Nuovo Spettrometro Digitale ad Alta Risoluzione Temporale e Frequenziale per Applicazioni Radioastronomiche

S. Montebugnoli, C. Bortolotti, S. Buttaccio, A. Cattani,
N. D' Amico, G. Grueff, A. Maccaferri, G. Maccaferri
A. Orfei, M. Roma, G. Tuccari, M. Tugnoli

IRA 225/96

File:spet.doc

INDICE

1) - PREMESSA.

In questa nota tecnica viene descritto il nuovo analizzatore di spettro digitale costruito nei laboratori della stazione radioastronomica di Medicina. Lo strumento descritto e` uno dei piu`avanzati nel suo genere ed apre nuove prospettive di indagine strumentale nel campo astrofisico ed in particolar modo dei fenomeni rapidamente variabili nel tempo, osservazioni di Maser in occultazione lunare, Pulsar e SETI.

2) - INTRODUZIONE.

Lo spettrometro digitale qui descritto fa uso di nuovi dispositivi DSP (Digital Signal Processor) LH9124 ad elevatissima velocita` della SHARP e relativo address generator LH9320. Il sistema e` completamente programmabile e, nella attuale versione, permette l' analisi spettrale di un segnale radioastronomico con banda fino a 20 MHz e risoluzione temporale di circa 30 ms. con un numero di canali che puo`variare da 1024 a 131072 (in potenze di 2). Allo stato attuale il sistema presenta una efficienza (time duty-cycle) relativamente bassa (~45% @ 6 MHz) ma e` previsto un upgrade che lo portera` ad una efficienza del 100% @ 12 MHz di banda di ingresso.

Lo spettrometro fu originariamente concepito come strumento a basso costo per la partecipazione della stazione radioastronomica di Medicina alle osservazioni del programma SETI della NASA. Con il passare del tempo ci si rese conto delle grandi potenzialita` che lo strumento poteva avere nel campo radioastronomico e questo porto` alla finalizzazione dello stesso alle osservazioni dell'impatto della cometa SL-9 con Giove avvenuto nel luglio 94. In quella occasione, sfruttando l'elevata risoluzione temporale, si è rivelata la presenza di una debole riga di emissione molecolare dell'acqua nella zona di impatto del frammento E con l'alta atmosfera di Giove. Tale rivelazione è stata possibile potendo compensare il marcato effetto Doppler introdotto dal rapido movimento relativo antenna/punto di impatto (Fig. 1). L'analisi di spettro, in campo radioastronomico, e`stata effettuata fino ad oggi mediante l'uso di autocorrelatori con il calcolo on-line della funzione di autocorrelazione e off-line della FFT di quest'ultima per ottenere lo spettro di potenza. Lo stato dell'arte dei dispositivi di calcolo (DSP) rende oggi possibile il calcolo on-line della FFT su un certo numero di dati, ottenuti campionando direttamente la radiofrequenza, ed il calcolo al volo del modulo quadro per ottenere lo spettro di potenza (Tav. 1). I vantaggi di questo secondo approccio sono rappresentati da una enorme compattezza del sistema, possibilita` di ottenere un numero di canali elevato (limitato solo da considerazioni di dinamica, velocita` di calcolo e capacita`di memoria) e maggiore risoluzione temporale. Il primo schema a blocchi del sistema di calcolo viene riportato in Fig. 2. In questo primo tentativo di architettura si pensava di usare, oltre che alle schede con lo Sharp a bordo per il calcolo della FFT, una scheda CSPI con 4xi860, per il computo del modulo quadro e della media degli spettri on line. In realta`, dopo una serie di considerazioni di tipo economico e funzionale, si e` deciso di usare una seconda scheda UltraDSP anche per il blocco che calcola la media e questa e` la configurazione che e`stata usata per effettuare le osservazioni

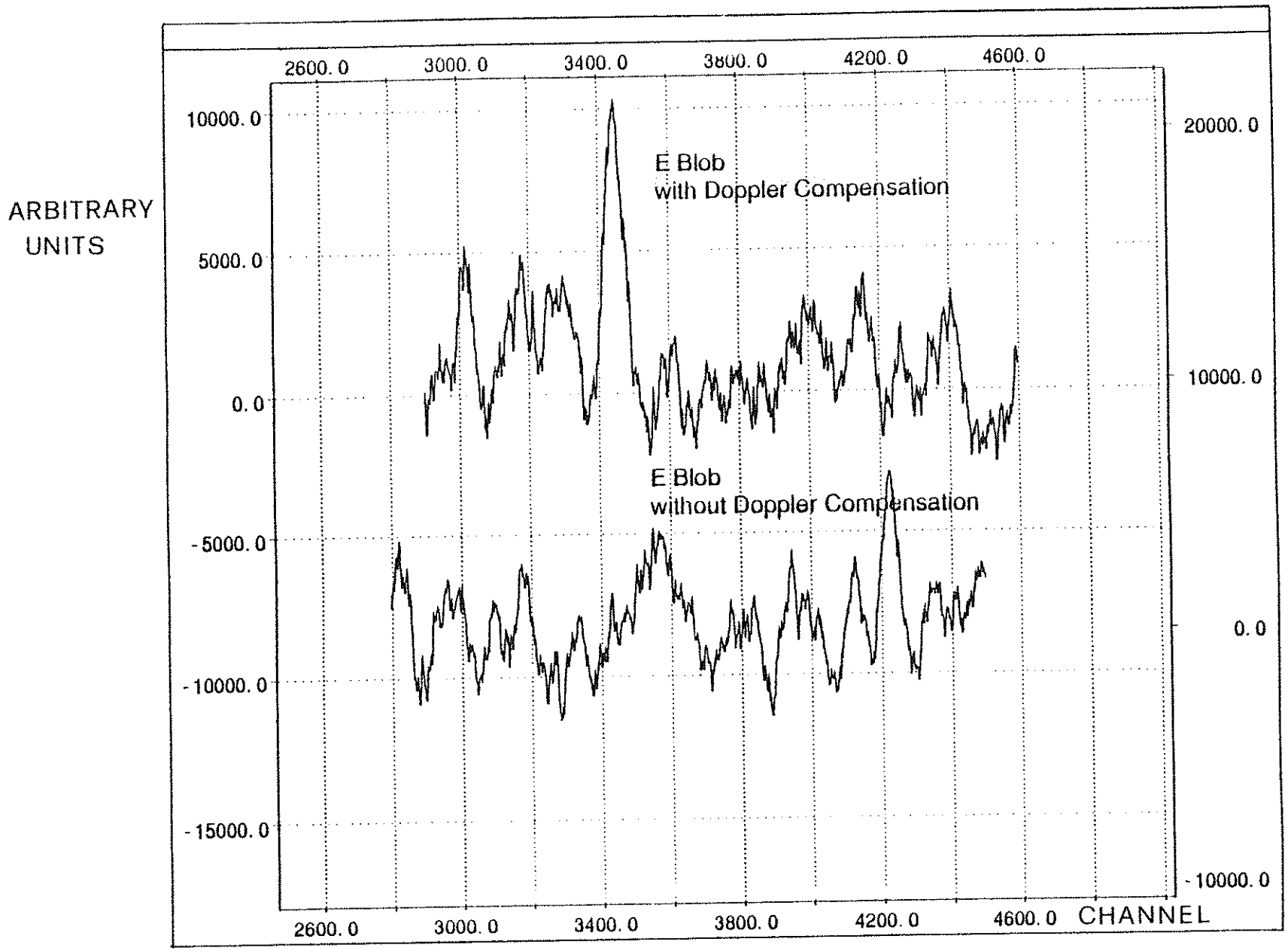
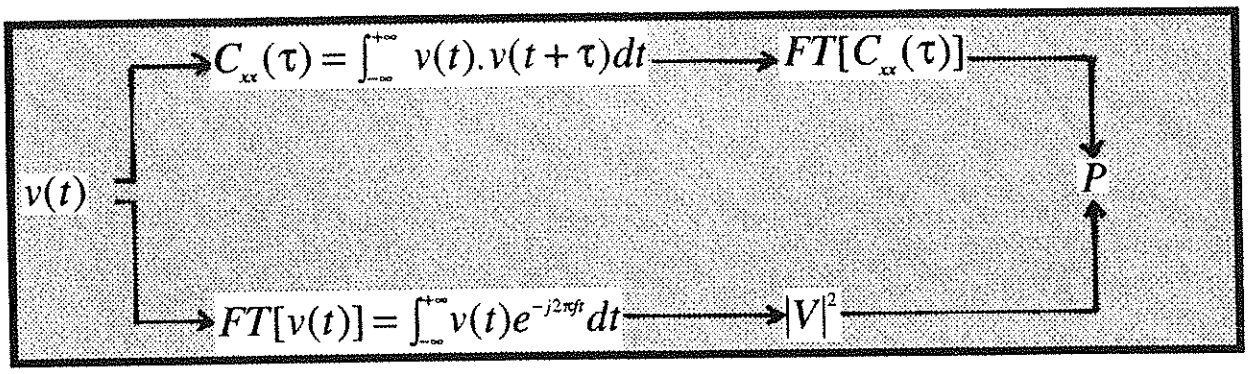

ARBITRARY
UNITS
2600. 0
3000. 0
3400. 0
3800. 0
4200. 0
4600. 0
10000. 0
5000. 0
0. 0
-5000. 0
-10000. 0
-15000. 0
20000. 0
10000. 0
0. 0
-10000. 0
E Blob
with Doppler Compensation
E Blob
without Doppler Compensation
CHANNEL

FIG. 1

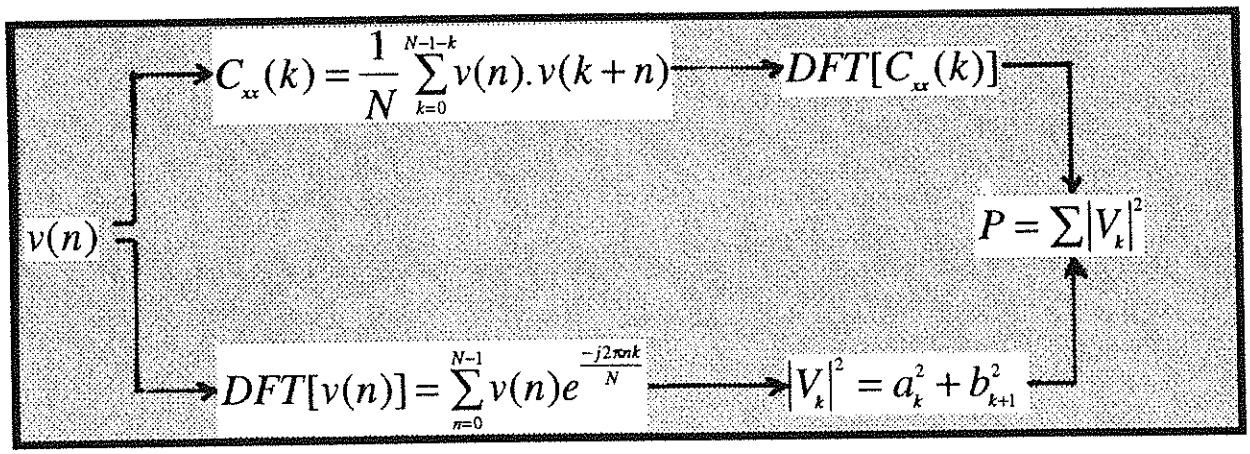
di Giove (Fig.2 a). E' tuttora in corso un up-grade del sistema che portera` a consistenti incrementi di prestazioni (questo sara` oggetto di uno degli ultimi capitoli di questo rapporto interno). In condizioni operative definitive si prevede una integrazione del sistema nel contesto globale di Medicina come riportato in Fig. 2c. Attualmente le due sedi (Antenna VLBI / Croce) sono collegate tramite un cavo coassiale sotteraneo e una certa quantita` di "doppini" per il controllo dello stato on e off della parabola, collegamento in rete etc.... E` gia` stata fatta la posa in opera di un cavo a 12 fibre ottiche ed e` previsto il collegamento tra le varie utenze Croce/Parabola. Un operatore dalla parabola VLBI dovrebbe essere in grado di osservare indifferentemente con l'autocorrelatore (ARCOS) a 2000 canali del gruppo di Arcetri e/o con lo spettrometro di Medicina. Si potrebbe, quindi, arrivare a potere osservare a banda larga con l'autocorrelatore una determinata sorgente e, contemporaneamente, effettuare uno zoom (temporale e/o frequenziale) con lo spettrometro su di una determinata porzione della stessa banda osservata. Questo potrebbe risultare molto utile in determinate situzioni osservative.

Non e` possibile riportare, in questo rapporto interno, gli schemi elettrici dei vari blocchi costitutuenti lo spettrometro sia per la enorme mole di documentazione che si sarebbe costretti ad accentrare in un solo volume sia perche` questi sono o saranno descritti in rapporti interni IRA singoli.

POWER SPECTRUM COMPUTATIONS



Analog Domain



Digital Domain

TAV. 1

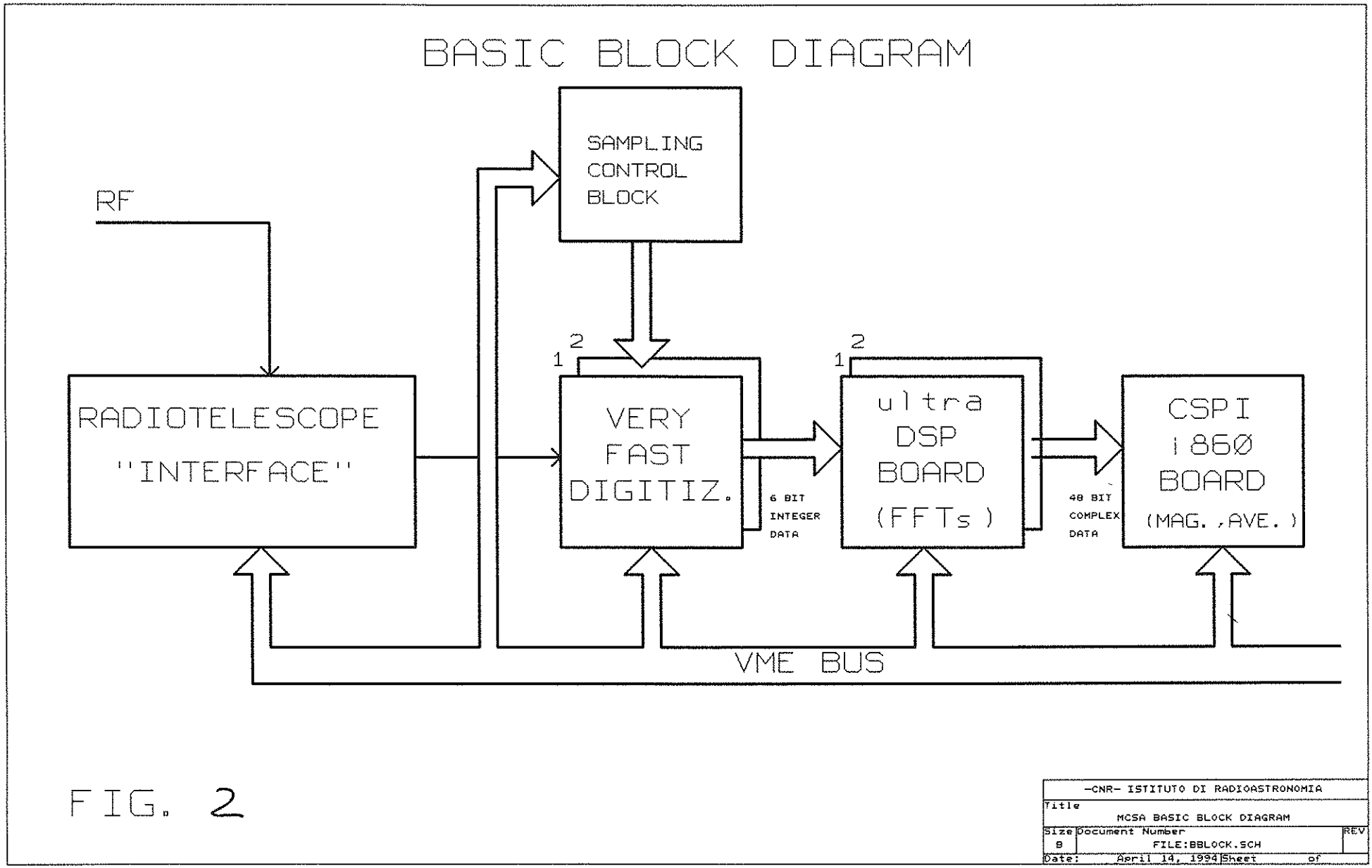

BASIC BLOCK DIAGRAM
SAMPLING CONTROL BLOCK
RF
RADIOTELESCOPE "INTERFACE"
1
2
VERY FAST DIGITIZ.
6 BIT INTEGER DATA
1
2
ultra DSP BOARD (FFTs)
48 BIT COMPLEX DATA
CSPI i860 BOARD (MAG., AVE.)
VME BUS
-CNR- ISTITUTO DI RADIOASTRONOMIA
Title
MCSA BASIC BLOCK DIAGRAM
Size B
Document Number
FILE:BBLOCK.SCH
REV
Date: April 14, 1994
Sheet of

FIG. 2

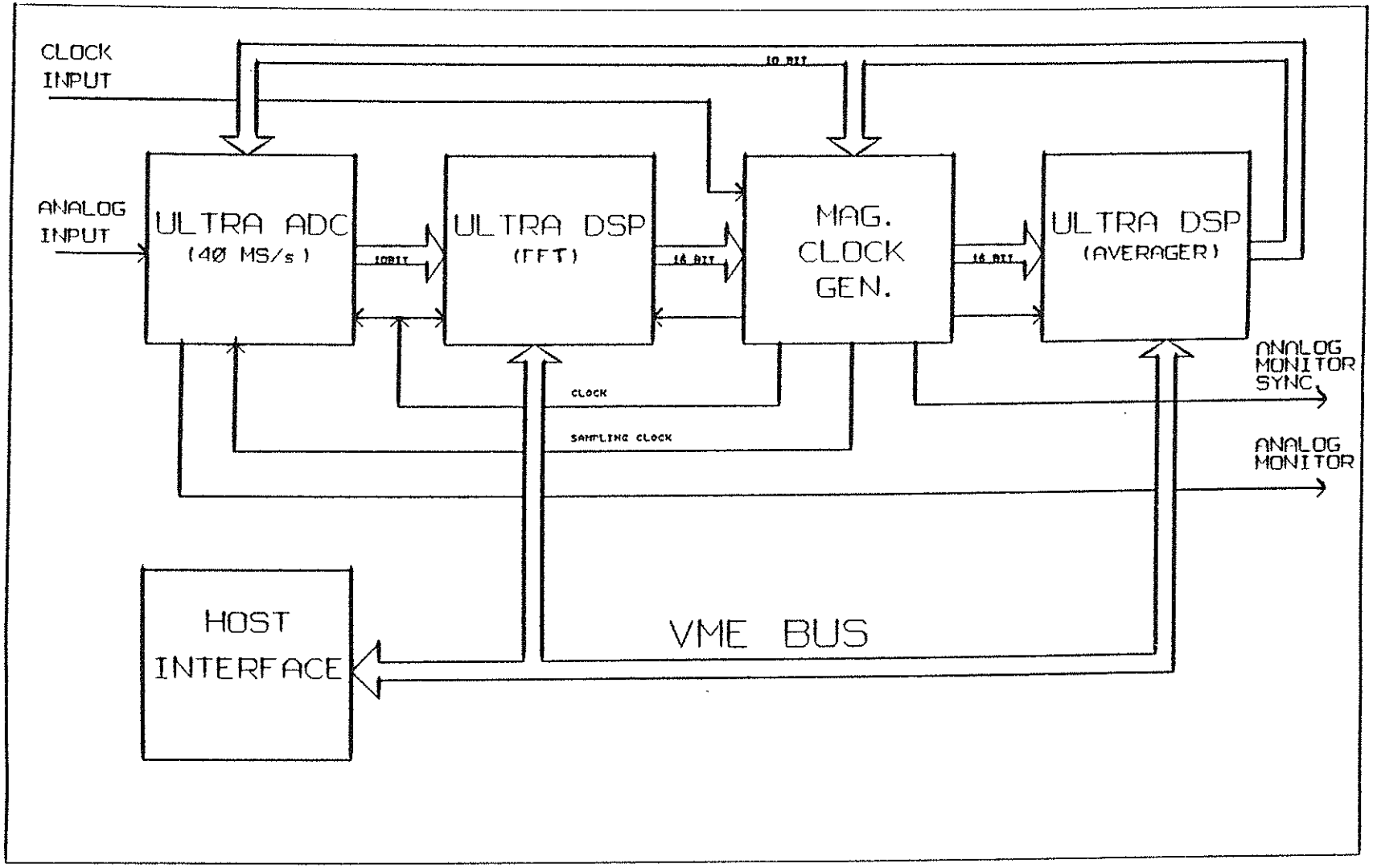


Fig. 2aDSP System Core

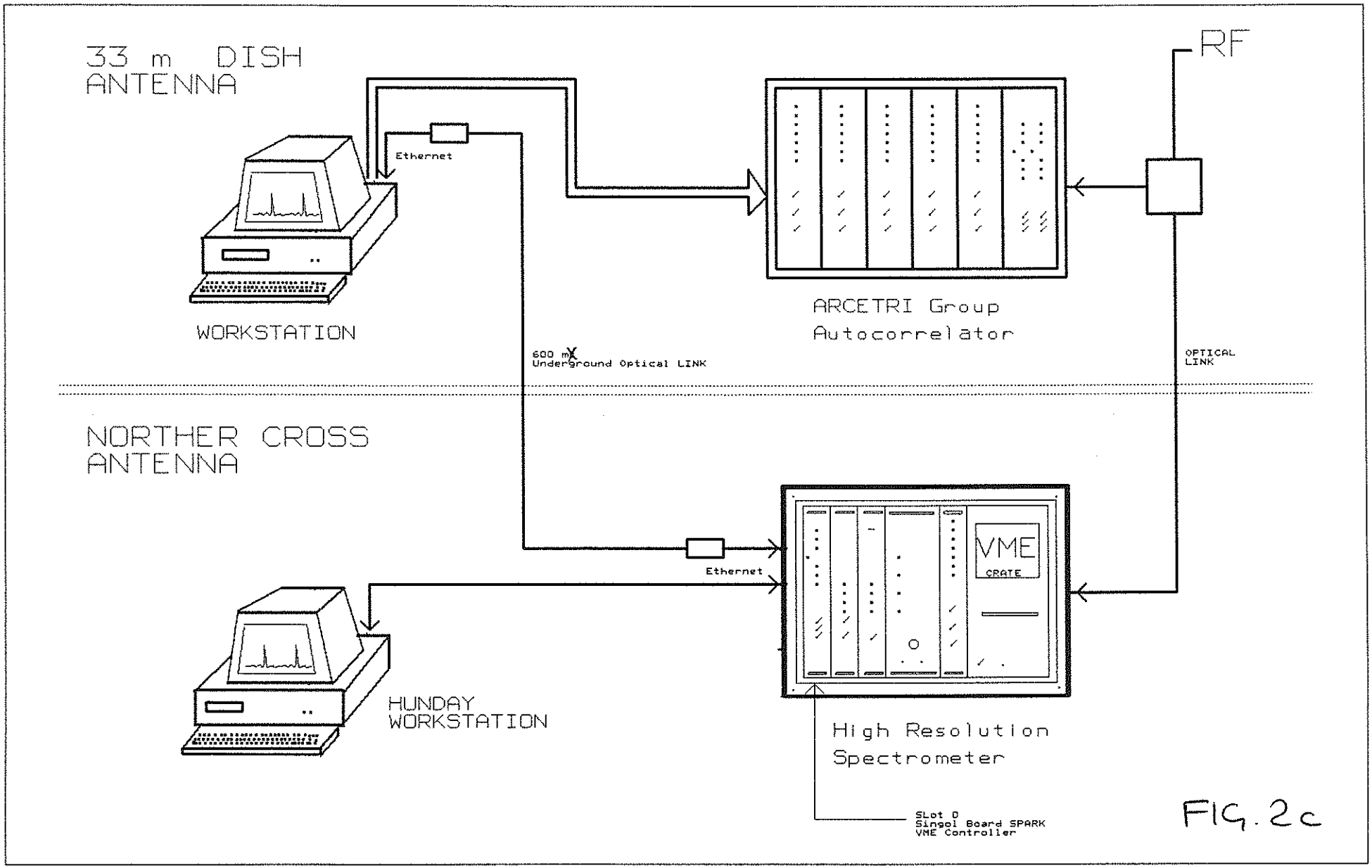

33 m DISH
ANTENNA
RF
Ethernet
WORKSTATION
ARCETRI Group
Autocorrelator
600 m
Underground Optical LINK
OPTICAL
LINK
NORTHER CROSS
ANTENNA
Ethernet
VME
CRATE
HUNDAY
WORKSTATION
High Resolution
Spectrometer
SLot 0
Singal Board SPARK
VME Controller

FIG. 2c

3) - IL CHIP SET SHARP LH9124/LH9320.

Prima di affrontare la descrizione dello schema a blocchi dello spettrometro e, in particolare, del sistema VME che contiene tutte le schede di conversione A/D, calcolo della FFT, modulo quadro e media, soffermiamoci un attimo sulla introduzione del chip set della Sharp LH9124/LH9320 che costituisce il cuore di tutto il sistema. Una descrizione completa del chip set in questione è stata fatta in un precedente rapporto interno IRA [6], al quale si rimanda chi fosse interessato ad una descrizione più esaudiente.
Il DSP LH9124, è un processore digitale di segnali ad altissime prestazioni che si presenta con una architettura completamente diversa da quella dei più comuni DSP rappresentando una unità computazionale ottimizzata per l' esecuzione di un certo numero di calcoli. Infatti, questo processore non è assolutamente in grado di generare gli indirizzi come tutti gli altri DSP standard, ma necessita di un address generator: l' LH9320 come appare in Fig. 3. Anche se non è immediato dare una classificazione per tale dispositivo, lo possiamo pensare come un "processore vettoriale" in quanto ogni istruzione ha come operando l' intera memoria o l' intero vettore contenente i dati da elaborare. Il singolo passo in cui il processo viene scomposto, e` eseguito da una singola istruzione che opera così o sull' intero array di dati che si presenta alla porta di ingresso, o sui risultati intermedi. Questo modo di operare e` particolarmente vantaggioso nel LH9124 per la presenza di ben 4 porte di I/O bidirezionali a 48 bit (24 reali e 24 complessi) programmabili (Fig. 4 a,b).
Generalmente i dati arrivano al DSP dal convertitore A/D attraverso il buffer di memoria collegato alla porta QR/QI. Attraverso la stessa porta, opportunamente programmata, escono i risultati della elaborazione verso ulteriori processi, se necessari. Nulla vieta di entrare con i dati attraverso altre porte sfruttando le varie possibilita` di direzione di flusso dei dati all'interno del chip come riportato in fig. 5a, b. Le porte AR/AI e BR/BI sono orientate all' handling dei risultati intermedi, mentre quella CA/CI all' ingresso dei "coefficienti" nel caso in cui l' operazione in corso abbia due operandi (ad es. i coefficienti W_N^k nei vari butterfly nel calcolo della FFT di N punti). Il flusso dei dati è caratterizzato, nella maggior parte dei casi, dall' ingresso al DSP dalla porta Q, scambio dei risultati parziali tra le porte A e B per tutta la durata dell' elaborazione, acquisizione dei coefficienti attraverso la porta C e uscita dei dati elaborati ancora attraverso la Q. Tramite la porta A o B è poi possibile prelevare dati, attraverso il

bus che il più delle volte è un bus VME. La configurazione a quattro porte di I/O rende particolarmente flessibile il sistema e, soprattutto, riduce al minimo circuiteria esterna necessaria. Questo e` dettato unicamente dal fatto che la gestione del flusso e direzione dei dati tra una porta e l' altra, è svolta internamente al chip. Ogni istruzione eseguita dal DSP è in pratica un passo dell' algoritmo in esecuzione in quel momento. Ad ogni istruzione (cioè operazione da effettuare sui dati) corrisponde un codice che viene introdotto dall' esterno attraverso il bus Function Code (FC[4:0]), mentre il percorso e la direzione del flusso dei dati tra una porta e l' altra viene programmato attraverso il Data Flow (DF[2:0]). Nel processore non e` prevista memoria interna per il programma come invece accade per i normali DSP e non dispone neppure dei classici registri che caratterizzano tutte le architetture dei microprocessori e DSP in generale (come ad esempio il program counter) per cui il codice relativo alle istruzioni ed ai modi di indirizzamento devono essere forniti, passo dopo passo, da un opportuno controllore esterno. E`in grado di operare con clock fino a 40 MHz e lavora con una aritmetica in virgola fissa a 24 bit con l' estensione "block floating point" che permette di conciliare la semplicità tipica dei dispositivi a virgola fissa con le grandi capacità dinamiche dei sistemi operanti in floating point. In aggiunta a quanto detto sopra, il dispositivo ha al suo interno due accumulatori da ben 60 bit ed istruzioni in grado di effettuare butterfly radix-2,radix-4 e radix-16 in un solo colpo e quindi in grado di accelerare il calcolo di una FFT, unitamente ad una grande semplicità di programmazione. L' LH9124 si presenta in contenitore PGA (Pin Grid Array) a 262 piedini. I segnali che usa per interfacciarsi con il mondo esterno sono raggruppati secondo le categorie che seguono:

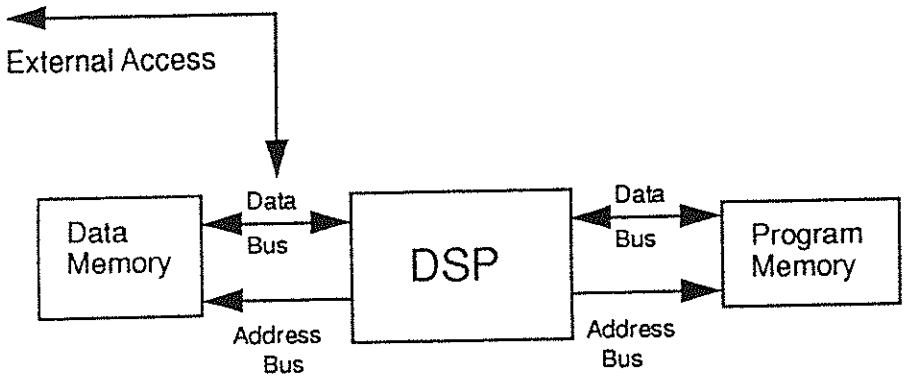
- ***Bus Dati***

- **QR[23:0]** Parte reale della porta bidirezionale Q. E' una porta bidirezionale a 24 bit che normalmente viene usata per l' I/O di dati reali con il mondo esterno.

- **QI[23:0]** Parte immaginaria della porta bidirezionale Q. E' una porta bidirezionale a 24 bit che normalmente viene usata per l' I/O di dati "immaginari" con il mondo esterno.

La porta Q viene normalmente detta porta di acquisizione.

CLASSICAL DSP SYSTEM



SHARP DSP SYSTEM

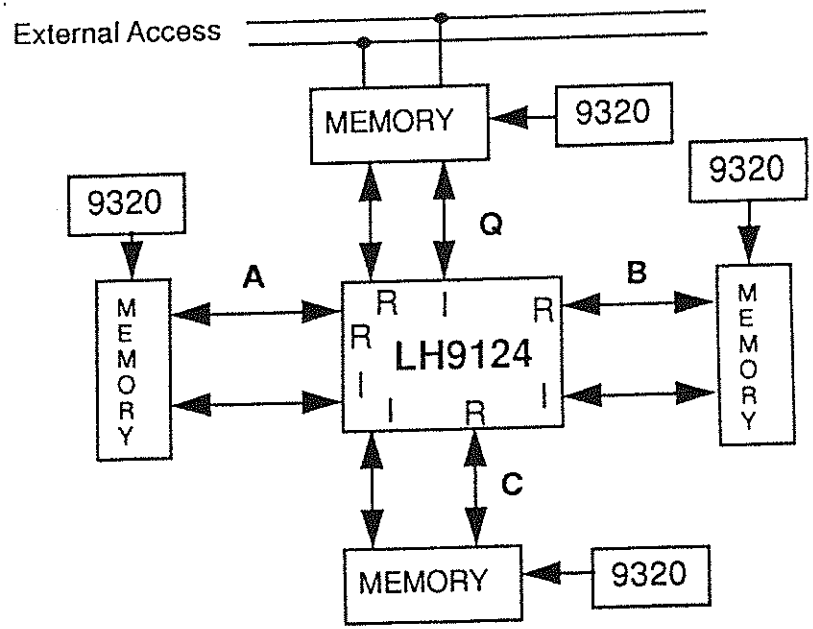


FIG. 3

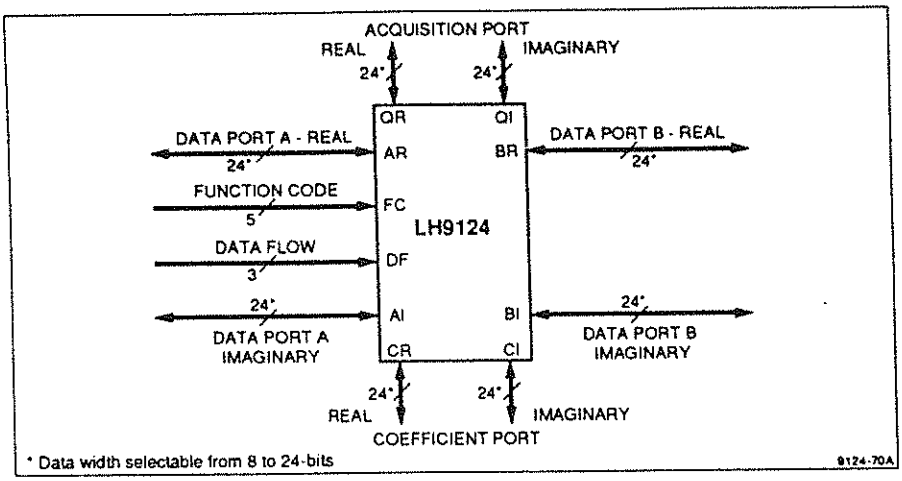

ACQUISITION PORT
REAL
IMAGINARY
24*
24*
QR
QI
DATA PORT A - REAL
AR
BR
DATA PORT B - REAL
24*
24*
FUNCTION CODE
5
FC
LH9124
DATA FLOW
3
DF
24*
AI
BI
24*
DATA PORT A
IMAGINARY
CR
CI
DATA PORT B
IMAGINARY
24*
24*
REAL
IMAGINARY
COEFFICIENT PORT
* Data width selectable from 8 to 24-bits
9124-70A

FIG. 4a

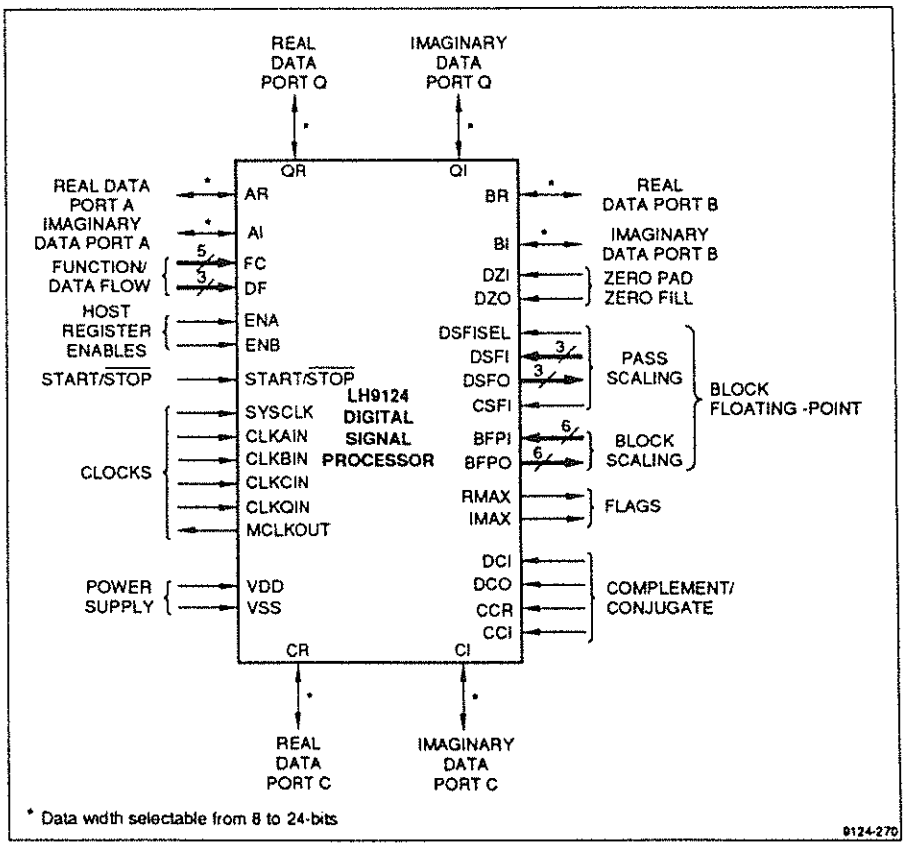

REAL
DATA
PORT Q
IMAGINARY
DATA
PORT Q
QR
QI
REAL DATA
PORT A
AR
BR
REAL
DATA PORT B
IMAGINARY
DATA PORT A
AI
BI
IMAGINARY
DATA PORT B
5
FUNCTION/
DATA FLOW
3
FC
DF
DZI
DZO
ZERO PAD
ZERO FILL
HOST
REGISTER
ENABLES
ENA
ENB
DSFISEL
DSFI
DSFO
CSFI
PASS
SCALING
BLOCK
FLOATING -POINT
START/STOP
START/STOP
LH9124
DIGITAL
SIGNAL
PROCESSOR
SYSCLK
CLKAIN
CLKBIN
CLKCIN
CLKQIN
MCLKOUT
CLOCKS
BFPI
BFPO
BLOCK
SCALING
RMAX
IMAX
FLAGS
POWER
SUPPLY
VDD
VSS
DCI
DCO
CCR
CCI
COMPLEMENT/
CONJUGATE
CR
CI
REAL
DATA
PORT C
IMAGINARY
DATA
PORT C
* Data width selectable from 8 to 24-bits
9124-270

FIG. 4b

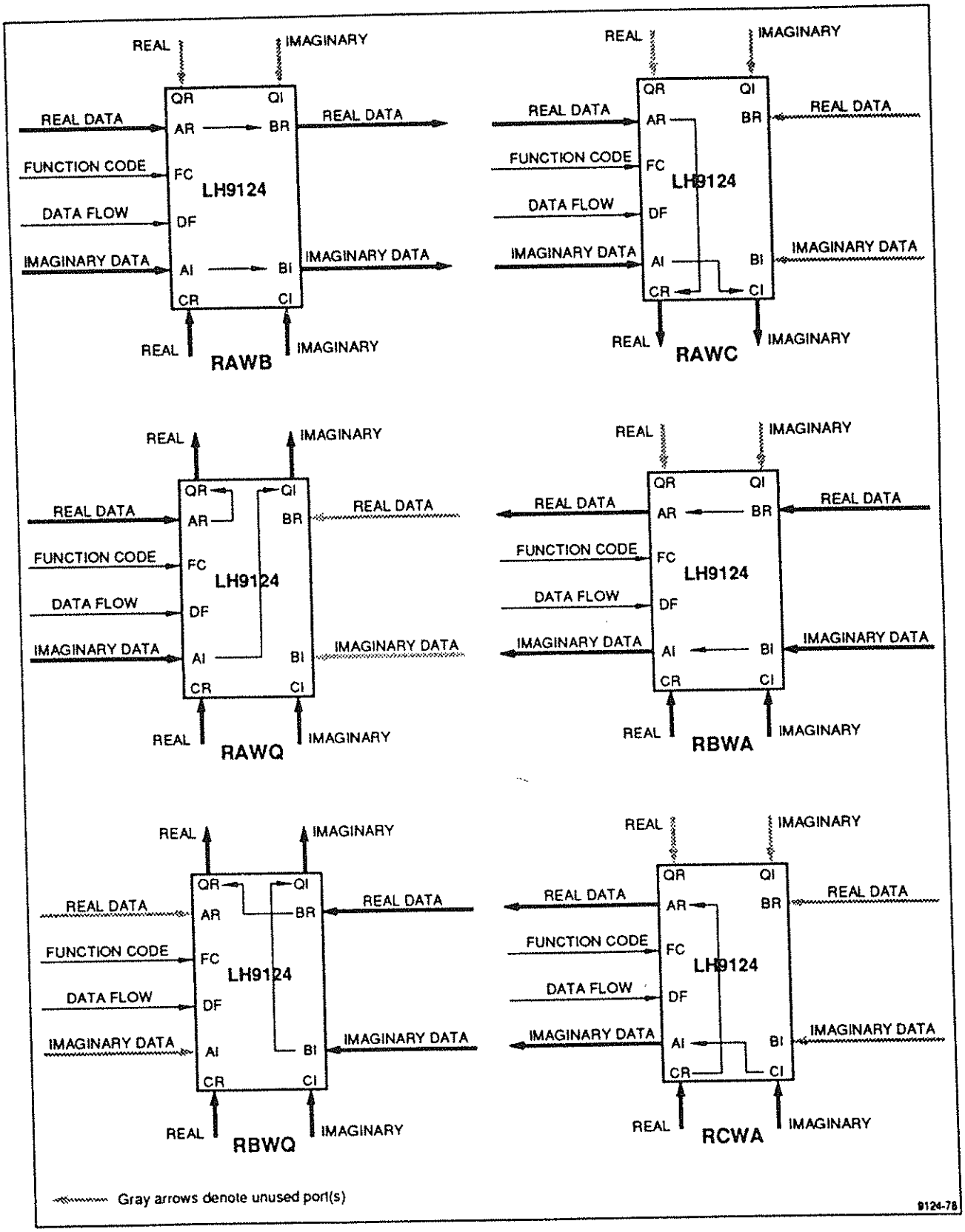

REAL
IMAGINARY
QR
QI
REAL DATA
AR
BR
REAL DATA
FUNCTION CODE
FC
LH9124
DATA FLOW
DF
IMAGINARY DATA
AI
BI
IMAGINARY DATA
CR
CI
REAL
IMAGINARY
RAWB
RAWC
RAWQ
RBWA
RBWQ
RCWA
Gray arrows denote unused port(s)
9124-78

FIG. 52

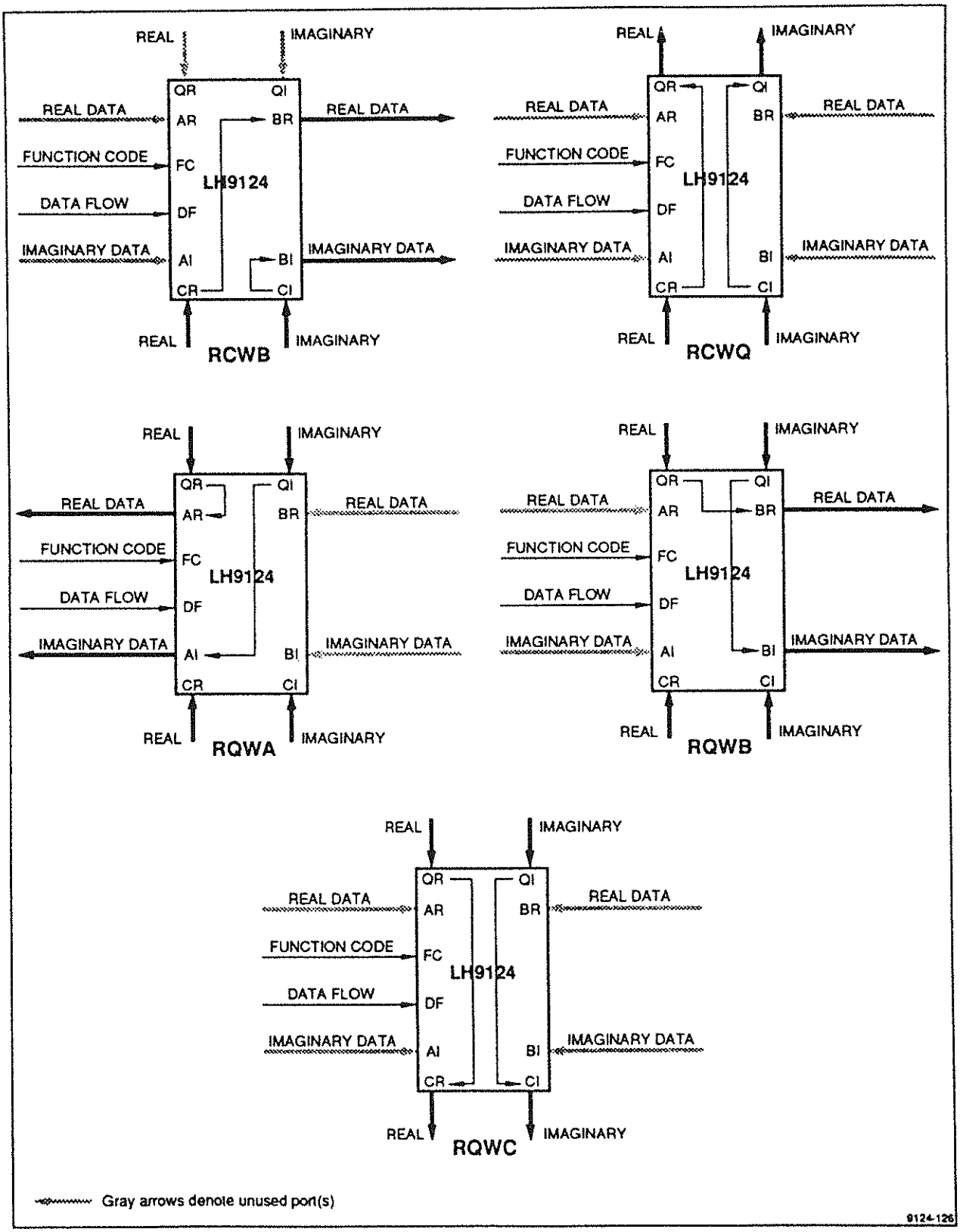

REAL
IMAGINARY
QR
QI
REAL DATA
AR
BR
REAL DATA
FUNCTION CODE
FC
LH9124
DATA FLOW
DF
IMAGINARY DATA
AI
BI
IMAGINARY DATA
CR
CI
REAL
IMAGINARY
RCWB
RCWQ
RQWA
RQWB
RQWC
Gray arrows denote unused port(s)
9124-126

FIG. 5b

- **AR[23:0]** Parte reale della porta bidirezionale A. Questo è un bus a 24 bit (logica positiva) usato per scambiare dati reali tra la porta A e la memoria esterna ad essa collegata.

- **AI[23:0]** Parte immaginaria della porta bidirezionale A. Questo è un bus a 24 bit usato per scambiare dati "immaginari" tra la porta A e la memoria esterna ad essa collegata.

- **BR[23:0]** Parte reale della porta bidirezionale B. E' un bus a 24 bit usato per scambiare dati reali tra la porta B e la memoria esterna ad essa collegata.

- **BI[23:0]** Parte immaginaria della porta bidirezionale B. E' un bus a 24 bit (logica positiva) usato per scambiare dati "immaginari" tra la porta B e la memoria esterna ad essa collegata.

Le porte A e B sono anche dette porte dati

- **CR[23:0]** Parte reale della porta bidirezionale C. Questo è un bus a 24 bit usato per lo scambio dei coefficienti (nel senso generale di "secondo operando") tra la porta C e la memoria esterna.

- **CI[23:0]** Parte immaginaria della porta bidirezionale C. Questo è un bus a 24 bit usato per lo scambio dei coefficienti "immaginari" tra la porta C e la memoria esterna.

La porta C viene detta porta dei coefficienti.

- ***Segnali di Controllo***

- **DF[2:0]- Data Flow:** INGRESSO, Logica positiva. Il codice presentato su questo bus indica il percorso che i dati devono effettuare in termini di quale porta costituisce la sorgente dei dati e verso quale porta questi devono andare. Esistono varie possibilità di percorso o flusso dei dati (vedere 5a,5b) a ciascuna delle quali corrisponde un preciso codice DF.

-**FC[4:0]- Function Code:** INGRESSO, logica positiva. Il codice presentato su questo bus determina l' istruzione che verrà eseguita nel prossimo passo. Il set completo viene riportato in tav. 3. Per come è costituito internamente il DSP, la presentazione del codice

relativo alla istruzione da eseguire deve essere fatta almeno un paio di cicli di clock prima dell' arrivo dei dati. Nel caso, ad esempio, del calcolo di una FFT si possono iniziare a leggere i dati nella memoria di I/O attraverso la porta Q, due cicli dopo che i segnali DF e FC sono stati settati.

-START/STOP- Start of pass: INGRESSO, logica positiva. Una volta definita attraverso il bus FC [4:0] l' istruzione che deve essere eseguita, lo stato "alto" di questa linea provoca lo start del passo. Questo stato deve persistere per tutta la durata del passo, ad esempio per tutta la durata della lettura dei dati fino al ciclo di clock prima di quello relativo alla lettura dell' ultimo dato. Con il sopraggiungere dello Stop (livello basso) vengono eseguite le ultime operazioni cioè la lettura dell' ultimo dato e le operazioni di troncamento ed arrotondamento del risultato.

- DCI- Data Complement/Coniugate Input: INGRESSO, logica positiva. Quando il livello di questo segnale di controllo è alto, viene complementato il segnale complesso in input. Un livello basso su questa linea, lascia inalterato il dato in ingresso.

- DCO- Data Complement/Coniugate Output: INGRESSO, logica positiva. Un livello alto in questa linea, porta a complementare i dati di uscita mentre un livello basso, lascia inalterati gli stessi.

- CCR- Coefficient Complement Real: INGRESSO, logica positiva. Il livello di questa linea, controlla il segno della parte reale dei coefficienti letti attraverso la porta C. CCR=1 significa che il segno viene cambiato. Per contro CCR=0 lascia il segno del coefficiente letto, inalterato.

- CCI- Coefficient Complementary Immaginary: INGRESSO, logica positiva. Questa linea di controllo si comporta come la CCR precedente, ma esclusivamente per la parte immaginaria dei coefficienti.

Le linee CCR e CCI sono molto utili per ridurre le dimensioni della memoria che contiene i coefficienti, come già visto in precedenza.

- DZI- Zero Input Data: INGRESSO, logica positiva. Un livello alto in questa linea di controllo, porta a sostituire il set di dati in igresso con

tutti 0. Un livello 0 permette la lettura del set dei dati in maniera normale.

- **DZO- Zero Output Data:** INGRESSO, logica positiva. Un livello alto su questa linea, forza tutti i dati in uscita a zero. Il livello zero permette ai dati di raggiungere la porta di destinazione con segno inalterato.

- **ENA- Enable A:** INGRESSO, logica positiva. Quando il livello di questa linea è alto, viene abilitata la lettura (al ciclo di clock successivo) dei bus FC[4:0] e DF[2:0].

- **ENB- Enable B:** INGRESSO, logica positiva. Un livello alto in questa linea, abilita (al ciclo successivo di clock) la lettura dei livelli dei segnali DCI, DCO, DSFI, DSFI, DSFISEL, BFPI[5:0].

Per questi due ultimi segnali devono essere soddisfatte alcune condizioni: il segnale ENA deve essere settato prima di ENB ed almeno due cicli di clock prima della attivazione del segnale Start/Stop.

- ***Gestione del Block Floating-Point e Fattore di Scala.***

- **DSFISEL- Data Scaling Factor Input Select:** INGRESSO. Le modalità con cui vengono trattati i dati all' interno del DSP, sono determinate dallo stato di questo segnale. Con DSFISEL=1 si abilita il trattamento "non automatico", nel senso che i dati vengono scalati verso sinistra di tanti posti quanti ne sono indicati nel bus DSFI (data scaling factor). Con il livello DSFISEL=0 si abilita il trattamento automatico dei dati nel formato Block Floating Point. Come esempio, se si vuole abilitare lo scaling dei dati in ingresso in modo non automatico di 2 posti a sinistra, si deve impostare DSFISEL=1 e DSFI[2:0]=010.

- **BFPI[5:0]- Block Floating Point Input:** INGRESSO, logica positiva. Quando è attivata l' aritmetica in Block Floating Point (DSFISEL=0), attraverso questo bus è possibile leggere il fattore di scala accumulato nel passo precedente, per poterne tenere conto nel passo che segue per evitare l' overflow.

- **BFPO[5:0]- Block Floating Point Output:** USCITA, logica positiva. Quando è attivata l' aritmetica in Block Floating Point (DSFISEL=0), il dato letto nel bus BFPO[5:0] alla fine del passo, rappresenta il numero di bit di cui è stato scalato il dato in ingresso fino a quel momento o, più precisamente, la somma di tutti i fattori di scala relativi ad ogni passo precedente a quello corrente compreso quello impostato sul bus DSFI[2:0]. Nel caso di un sistema a singolo processore, il bus BFPI[5:0] deve essere collegato con il bus BFPO[5:0], nel caso di un sistema a multiprocessore (più LH9124 collegati in cascata) si devono collegare le uscite BFPO del primo DSP agli ingressi BFPI del DSP successivo.

- **DSFI[2:0]- Data Scaling Data Input:** INGRESSO, logica positiva. Su questo bus viene impostato il fattore di scala relativo al passo in corso. Se si pone DSFISEL=1 (scaling manuale) il fattore di scala deve essere impostato manualmente. Se si pone DSFISEL=0 lo scaling avviene automaticamente, collegando il bus DSFI[2:0] a quello DSFO[2:0].

- **DSFO[2:0]- Data Scaling Factor Output:** USCITA, logica positiva. Su questo bus viene fornito il fattore di scala da applicare al passo successivo. Questo viene valutato sulla base di una stima in eccesso dei risultati dei passi precedenti.

- **CSFI- Coefficient Scaling Factor Input:** INGRESSO, logica positiva. Quando CSFI è alto, il coefficiente letto viene scalato di un bit verso a destra. Con CSFI=0, i coefficienti letti rimangono inalterati.

- ***Segnali di Temporizzazione.***

- **SYSCLK- System clock:** INGRESSO. Ingresso del segnale di clock per l' intero DSP LH9124.

- **CLKQIN- Clock Input for PORT Q:** INGRESSO. Questo ingresso gestisce un registro interno per la memorizzazione dei dati presenti sulla porta Q. Lo stesso registro viene letto al ciclo successivo di SYSCLK per dare poi il via alla elaborazione. Questo segnale di clock è separato dal segnale SYSCLK.

- **CLKAIN- Clock Input for PORT A:** INGRESSO. Simile al CLKQIN ma agisce sulla porta A.

- **CLKBIN- Clock Input for PORT B:** INGRESSO. Simile al CLKQIN ma agisce sulla porta B.

- **CLKCIN- Clock Input for PORT C:** INGRESSO. Simile al CLKQIN ma agisce sulla porta C.

- **MCLKOUT- Machine-Cycle Clock Output:** USCITA. Da questa uscita è possibile prelevare il segnale SYSCLK diviso per 4.

- ***Segnali di Stato.***

- **RMAX- Real Maximum:** USCITA, logica positiva. Questo segnale viene generato dalla comparazione dei sette bit più significativi del dato corrente con gli omologhi sette bit del dato immediatamente precedente. RMAX diventa alto quando la parte reale del dato corrente risulta essere maggiore di tutti i precedenti.

- **IMAX- Immaginary Maximum:** USCITA, logica positiva. Stessa funzione di RMAX, ma riferita alla parte immaginaria.

Le Istruzioni Base.

Il DSP LH9124 implementa 26 funzioni, raggruppabili nei seguenti insiemi:

- Digital Signal Processing. (DSP)
- Aritmetica Vettoriale. (Vector Aritmetic)
- Utilizzo Generale. (General Purpose)
- Aritmetica Complessa. (Complex Aritmetic)
- Logica Vettoriale (Vector Logical)

Vengono ora descritte brevemente le istruzioni base piu` importanti. Per una analisi piu` approfondita delle stesse, si rimanda al rapporto interno IRA197/95

Analisi delle Funzioni Base.

Il set delle funzioni del DSP è tale da permettere la soluzione di una vasta gamma di problematiche di elaborazione. Visto che in questa sede non è possibile descrivere dettagliatamente tutte le funzioni, si fara`solo un accenno sommario.

- **BFLY2** (Opcode: 02) Radix-2 Butterfly

Latenza=18

Questa funzione calcola due Butterfly radix-2 complesse alla volta, per cui necessita, prima di essere implementata, che siano già disponibili almeno quattro campionamenti del segnale. Nella BFLY-2 i coefficienti del primo stadio sono unitari per cui è necessario leggerli dalla porta C, si può quindi approffittare, quando richiesto, per effettuare la windowing dei campionamenti che entrano attraverso la porta Q

- **BFLY4** (opcode 01) Radix-4 Butterfly

Latenza=18

Funzione per il calcolo di un Butterfly Radix-4. La BFLY4 calcola con un solo stadio la FFT su quattro punti dove, invece, occorrerebbero 2 stadi BFLY2.

Come nel caso della BFLY2, i dati in ingresso sono considerati essere complessi, i coefficienti sono inseriti tramite la solita porta C e sono necessari almeno 4 dati campionati prima di fare partire il passo.
L' ordine di uscita dei dati rimane lo stesso di quello del passo BLFY2. La funzione BFLY4 richiede una particolare sequenza di indirizzamento dei dati e coefficienti, che viene fornita dal generatore di indirizzi LH9320. Sia nella BFLY2 che nella BFLY-4 i coefficienti del primo stadio sono tutti unitari e, trascurando i cicli di latenza, è veloce il doppio. Anche in questo caso, siccome nel primo stadio non è necessario leggere i coefficienti twiddle dalla porta C, si può approffittare, quando richiesto, per effettuare la windowing dei campionamenti che entrano attraverso la porta Q. Questo si ottiene semplicemente, e senza perdita aggiuntiva di tempo, moltiplicando gli stessi man mano che entrano per i corrispondenti coefficienti della finestra. In ambedue i casi (butterfly radix-2 e 4) è possibile quindi usare le funzioni BNWD2 e BNWD4, implementate nell' LH9124 che permettono di effettuare i suddetti butterfly contemporaneamente alla windowing.

- **BFLY16** (opcode 00) Radix-16 Butterfly
Latenza=68

Questa funzione calcola una butterfly radix-16 mediante il calcolo di due butterfly radix-4 adiacenti. L' architettura dell' LH9124 è ottimizzata per mettere a disposizione questa potente funzione che è in grado di eseguire la butterfly in 16 cicli per ogni 16 punti eguagliando in tempo di esecuzione, la BFLY2 e BFLY4. L' esecuzione della funzione BFLY16 porta il chip ad esprimere le sue massime prestazioni con 6 moltiplicazioni ed 11 addizioni per ogni ciclo di clock che, a 40 MHz, corrispondono a ben 680 milioni di operazioni in fixed point al secondo! Siccome l' esecuzione di questa funzione equivale alla esecuzione di due BFLY4 e quattro BFLY2, si può dire che fornisce prestazioni doppie e quadruple rispetto alle stesse. Nel calcolo di una qualsiasi FFT non è sempre possibile scegliere solo passi radix-16, perchè la scelta dei passi per la struttura ottimale è comunque legata al numero di punti da trattare. Da una attenta analisi della funzione, ci si rende anche conto che non è possibile avere una funzione di windowing nel primo stadio per il fatto che non tutti i coefficienti sono unitari per cui non esiste, purtroppo, una funzione BWND16. Se il primo stadio della FFT è un radix-16 ed è richiesta una windowing, occorre un passo addizionale

per tale operazione. Per questa funzione viene richiesto che l' address generator LH9320 fornisca un pattern di indirizzi ad hoc, per la lettura dei coefficienti dalla porta C.

- **BWND2** (opcode 05) Radix-2 Complex Window Butterfly
Latenza=20

La funzione implementa il windowing dei dati (primo stadio), moltiplicandoli per i coefficienti della finestra complessa man mano che arrivano dalla porta Q. Quando la particolare applicazione richiede tale windowing, questa funzione riduce i tempi che sarebbero richiesti da una architettura standard. I coefficienti della window entrano dalla porta C ed almeno 4 punti sono richiesti prima del lancio di tale funzione.

- **BWND4** (opcode 04) Radix-4 Complex Window Butterfly
Latenza=20

Questa funzione implementa il windowing dei dati (primo stadio), moltiplicandoli per i coefficienti di una funzione complessa di windowing, man mano che arrivano dalla porta di ingresso (lettura) dei dati Q, aumentando in tal modo la velocità. Anche in questo caso i coefficienti della window, arrivano dalla porta C. La parte reale è disponibile in ordine naturale di indici nella porta reale di destinazione scelta, la parte immaginaria è, allo stesso modo, posta sulla parte immaginaria della stessa porta.

- **BRFT** (opcode 07) Dual Real FFT Separation.
Latenza=18

Tale funzione esegue la separazione in uscita della FFT globale in due FFT reali e necessita di almeno 4 punti in ingresso. Per calcolare la FFT di due set di N dati reali come singolo set di N dati complessi (2N), si pongono i primi N dati reali sul bus reale ed i rimanenti N dati reali sul bus immaginario della catena di funzioni BFLYx o BFLYxx ed il risultato complesso, parcheggiato tramite una delle porte A o B, fornito alla funzione BFRT che ne rende le due FFT. In questa funzione la porta dei coefficienti C, viene ignorata. L' uscita della BRFT è formata da due FFT di N punti distinte, composte da coppie alternate di numeri complessi nel senso che i punti ad indice pari

costituiscono la FFT dei primi N punti reali entrati e quelli dispari la FFT dei rimanenti N punti reali entrati. Possiamo scrivere l' espressione delle due FFT presenti in uscita come:

$$FFT1 = [R(k) + R(N-k)] + j[I(k) - I(N-k)]$$
$$FFT2 = [I(k) + I(N-k)] - j[R(k) - R(N-k)]$$

con K=1,2,3,....N-1

La parti reali di FFT1 e FFT2 vengono trasferite nella parte reale della porta di destinazione in ordine naturale, quella immaginaria, allo stesso modo, nella parte immagina della stessa.

- **BFCT** (opcode 06) Fast Cosine Transform/Double Length (N Output)

Latenza=18

La funzione BFCT effettua il calcolo della FCT (Fast Cosine Transform). Ricombina in uscita i dati presentati all' ingresso del passo come risultato di una FFT complessa processata come FFT reale di 2N punti. I dati in uscita sono una sequenza ripetuta di coppie complesse che forniscono solo lo spettro positivo della trasformazione (i punti ad indice pari sono presentati nel bus reale e quelli dispari su quello immaginario). La parte reale dello spettro di 2N punti reali, viene presentata in uscita sulla parte reale della porta scelta, mentre quella immaginaria sulla parte immaginaria della stessa. L' ordine di presentazione è quello naturale e, ovviamente, vengono forniti in uscita N dati.

- **BFCT2** (opcode 0E) Fast Cosine Transform/Double Lenght (2N Output)

Latenza=18

Ricombina in uscita i dati complessi, presenti in ingresso, che provengono da una butterfly (BFLYx o xx) complesso di 2N dati reali forniti come array di N dati complessi. E' simile alla BFCT, fornisce in più lo spettro "riflesso" utile in eventuali IFFT. I dati in uscita vengono presentati nello stesso modo visto per la BFCT, ma il numero dei dati forniti è in questo caso 2N.

- **BCFIR** (opcode 08) Complex Finite Impulse Response Filter

Latenza=18

La funzione implementa un filtraggio di tipo FIR, non recursivo, sui dati in ingresso. Questo è il filtro comunemente conosciuto come Tapped Delay Line o filtro a struttura trasversale. In questo caso la risposta all' impulso desiderata (che determina le caratteristiche del filtro), deve essere caricata nella memoria dei coefficienti raggiungibile tramite la porta C. La parte reale dei risultati intermedi del FIR filter, è disponibile in uscita nella parte reale della porta prescelta ed in ordine naturale, la parte immaginaria nella parte immaginaria della stessa porta.

- **BDFIR** (opcode 09) Double Real FIR Filter.

Latenza=18

Questa funzione è del tutto simile alla precedente, solo che esegue il FIR filter su un doppio set di dati.

- **BRFIR** (opcode 0A) Real FIR FIlter.

Latenza=18

Esegue la stessa operazione di filtraggio di tipo FIR, ma su dati di ingresso reali.

- **CADD** (opcode 10) Complex Add

Latenza=18

La funzione CADD esegue la somma dei dati complessi e coefficienti.
I coefficienti sono inseriti tramite la porta C ed i dati da qualunque porta.. Occorrono almeno quattro punti per iniziare l' operazione e, ovviamente, quattro coefficienti: Il risultato è disponibile in uscita con lo stesso ordine dei dati in ingresso.

- **CMAG** (opcode 0C) Magnitude Squared.

Latenza=18

CMAG fornisce il modulo quadro di due numeri complessi dati in ingresso da qualsiasi delle tre porte disponibili (la porta dei coefficienti C non è ovviamente usata).

- **CMUL** (opcode 0D) Complex Multiply.

Latenza=18

Con questa funzione si ha la possibilità di moltiplicare un ingresso complesso con i coefficienti complessi provenienti dalla porta C. Questa funzione richiede almeno 4 numeri in ingresso. Le uscite hanno lo stesso ordine dei dati in ingresso.

- **CSUB** (opcode 11) Complex Subtract.

Latenza=18

La CSUB esegue la sottrazione tra i coefficienti complessi e i dati complessi in ingresso . Richiede almeno quattro numeri in ingresso: I dati in uscita hanno lo stesso ordine di come sono stati introdotti nel blocco.

- **VABS** (opcode 13) Vector Absolute Value.

Latenza=18

Determina il valore assoluto del dato in ingresso. La funzione richiede almeno 4 numeri in ingresso. I dati in uscita sono resi disponibili nello stesso ordine con cui sono stati inseriti.

- **VADD** (opcode 10) Vector Add.

Latenza=18

Somma coppie di dati in ingresso con coppie di coefficienti.
Occorrono almeno quattro punti per iniziare il processing dei dati in ingresso.
I risultati in uscita sono disponibili nello stesso ordine con cui si sono entrati i dati.

- **VMUL** (opcode12) Vector Multiply.

Latenza=18

La funzione permette il prodotto tra coppie di numeri reali in ingresso e coppie di coefficienti reali. I dati in uscita engono forniti nello stesso ordine con cui sono entrati.

- **VMXM** (opcode 1E) Vector Maximum and Minimum.
Latenza=20

Questa determina il vettore Max o Min dei dati forniti in ingresso. Dati in ingresso devono almeno essere 4. Le uscite sono fornite sul bus reale ed immaginario rispettivamente.

- **VSUB** (opcode 11) Vector Subtract.
Latenza=18

Sottrae coppie di coefficienti da coppie di dati presenti in ingresso. La funzione richiede un numero minimo di dati in ingresso pari a 4, per potere iniziare il processing. I risultati sono disponibili nell' ordine con cui sono forniti in ingresso.

- **VNAND** (opcode 18) Vector Logical Nand
Latenza=18

Funzione logica che esegue il NAND logico tra i dati in ingresso ed i coefficienti. Usando il segnale DCI=1, la funzione restituisce il NAND logico degli stessi. In ambedue i casi, l' ordine dei dati in uscita è lo stesso di quelli di ingresso.

- **VNOR** (opcode 1A) Vector Logical NOR
Latenza=18

Funzione che esegue il NOR logico tra i dati in ingresso ed i coefficienti. Anche in questo caso, come nel precedente, posizionando il segnale DCI=1, si ottiene l' OR logico. In ambedue i casi, l' ordine dei dati in uscita è lo stesso di quello di ingresso.

- **VPAS** (opcode 19) Vector Logical Pass.
Latenza=18

Questa semplice funzione, lascia passare le coppie di dati così come si presentano all' ingresso.

- **VXNOR** (opcode 1B) Vectro Logical Exclusive NOR.

Latenza=18

Esegue il NOR eslusivo logico tra i dati in ingresso ed i coefficienti (fig 36). Ponendo il livello del segnale DCI=1 la stessa funione fornisce l' OR esclusivo. In ambedue i casi, i dati si presentano in uscita con lo stesso ordine con cui entrano.

- **MOVC** (opcode 1D) Move Coefficient Data.

Latenza=18

Questa funzione è molto utile per spostare i coefficienti dalla porta C, a qualsiasi porta scelta tra le A,B,Q. Può essere usata per la verifica dei coefficienti prima dell' esecuzione di un qualsiasi algoritmo, se necessario. Durante l' esecuzione della funzione, eventuali dati alle porte A,B,Q vengono ignorati.

A'=A

con A complesso. I coefficienti spostati dalla porta C a quella di destinazione, sono disponibili su quest' ultima nello stesso ordine con cui sono stati mossi. La funzione richiede almeno 4 punti per potere iniziare il processing:

- **MOVD** (opcode 1C) Move Data.

Latenza=8 con RQWC e RAWC
Latenza=18 in tutti gli altri casi.

Questa funzione sposta dati da qualsiasi porta (A,B,C;Q) a qualsiasi porta (A,B,C;Q). In questo particolare caso si ha:

A=A'

con A e A' sono complessi. Questa è una funzione fondamentale per caricare dati e coefficenti. Questi ultimi, di norma, entrano nel relativo buffer collegato alla porta C secondo il percorso *Bus VME --> porta B*

--> *porta C*. Richiede che siano disponibili almeno quattro dati da spostare. I dati nella porta di destinazione, vengono presentati nello stesso ordine con cui sono stati prelevati dalla porta di provenienza. La stessa funzione è anche usata per introdurre i dati dalla porta Q.

4) - L' ADDRESS GENERATOR LH 9320.

Il DSP LH9124 e` stato disegnato per lavorare unitamente al suo generatore di indirizzi LH9320. Questo chip e` un address generator programmabile in grado di generare oltre 150 sequenze diverse su un range di indirizzamento che spazia tra 4 punti ed 1 milione di punti (20 bit) (Tav. 2). Sfruttando la sua architettura e` in grado di generare tutte le sequenze di indirizzamento richieste dalle funzioni implementate a bordo del DSP LH9124. Questo non richiede, come gli address generator standard, la programmazione degli indirizzamenti in quanto ha gia' implementato al suo interno potenti istruzioni che generano direttamente la sequenza per l' algoritmo richiesto; la maggior parte di queste sono dedicate al calcolo della FFT in tutte le sue forme. Il chip e`, per la sua architettura, ottimizzato per fornire i pattern per gli algoritmi:

- FFT (Fast Fourier Transform)
- DFT (Discrete Fourier Transfom)
- FIR filters (Finite Impulse Response)
- FCT (Fast Cosine Transform)
- DCT (Discrete Cosine Transform)

L'LH9320 puo' generare 150 diverse sequenze di indirizzamento orientate agli algoritmi:

1) *Digit Reverse for FFT*
2) *Two at a time real data only FFT Separation Pass.*
3) *Decimation/ auto-decimation.*
4) *Modulo Increment / decrement.*
5) *Overlap / discard*
6) *Radix-n (n=2,4,16) and mixed radix FFTs.*
7) *Double length (2N) real data only FFT FFT separation Pass.*
8) *FIR Filtering.*
9) *Interpolation addressing (index filling / index padding)*
10) *Circular buffering*

Il chip, contrariamente al DSP, ha un suo piccolo buffer di memoria in grado di contenere 32 istruzioni che anche se sembra essere di piccole dimensioni, e` piu' che sufficiente per il fatto che ogni istruzione indirizza un intero passo di programma. E' comunque

possibile, per algoritmi molto complessi, usare una memoria esterna al chip. Il dispositivo viene visto dal sistema di controllo come una qualsiasi periferica, grazie alla presenza di una interfaccia standard ad 8 bit ed e` in grado di generare tutti i segnali di controllo per la gestione dei processi di lettura e scrittura della memoria con i segnali di clock necessari al DSP per ricevere o fornire i dati attraverso la porta interessata. E` anche possibile generare indirizzamenti multicanale, che permettono al DSP di elaborare segnali diversi in sequenza, con l' ausilio di una logica di arbitraggio interna dei vari canali. Questo significa che da un solo canale e` possibile gestire 1 milione di indirizzi diversi, sino a 32 canali con ciascuno un massimo di 32 K indirizzi (32Kx32=1 M).

Il Chip LH9320.

Il chip appare con tutti i segnali di sistema e di interfaccia in Fig. 6 ed una sommaria descrizione dei pin è illustrata in Tav. 3 Vediamo ora di analizzarli brevemente uno per uno, premettendo che esistono due modi fondamentali:

- Funzionamento con memoria di programma interna (*Internal Memory Mode)*
- Funzionamento con memoria di programma esterna *(External Memory Mode)*

Nel caso delle schede che equipaggiano lo spettrometro si prevede l' uso della memoria di programma interna per cui affronteremo specificatamente questo caso.

A) Segnali di interfaccia con il sistema di controllo.

- DB[7:0] (Data Bus). Bus bidirezionale, a logica positiva, che permette la scrittura o lettura dei numerosi registri interni del chip; tale processo di scrittura o lettura avviene in due fasi: si scrive prima l'indirizzo del registro interessato ed immediatamente si scrive o si legge il dato relativo. Questo avviene a seconda del livello impostato in A0 , A1.

- A0,A1 (Host Control). Questi ingressi, attivi alti, oltre che gestire il modo di funzionamento, arbitrano anche lo scambio dei dati. Per sfruttare la memoria interna *(Internal Memory Mode)* si deve mantenere il livello del pin A1 basso, per cui il modo di funzionamento del chip e` determinato dal livello a cui e` portato il pin A0:

-A0=1 --> Il byte presente sul bus DB[7:0] e` interpretato come un indirizzo di un registro interno da scrivere.

-A0=0 --> Il byte presente sul bus DB[7:0] e` interpretato come un indirizzo di un registro interno da leggere.

- R/W (Read / Write). Ingresso, logica positiva. Questo segnale costituisce il clock vero e proprio per i registri interni dell' LH9320, alcuni dei quali (esempio il registro degli indirizzi) sono sensibili al fronte di salita ed altri (esempio il registro di programma) a quello di

Tav. 2

LH9320 Address Pattern Set Summary

MNEMONIC	DESCRIPTION
Fast Fourier Transforms (FFTs)	
BF2n (0 to 19)	Radix-2 data addresses
BF4n (0 to 9)	Radix-4 data addresses
BF16n (0 to 4)	Radix-16 data addresses
TF2n (0 to 19)	Radix-2 twiddle addresses
TF4n (0 to 9)	Radix-4 twiddle addresses
TF16n (0 to 4)	Radix-16 twiddle addresses
MXB24n (0 to 8)	Mixed radix (2,4) data addresses
MXB216n (0 to 3)	Mixed radix (2,16) data addresses
MXB416n (0 to 3)	Mixed radix (4,16) data addresses
MXB2416n (0 to 3)	Mixed radix (2,4,16) data addresses
MXT24n (0 to 8)	Mixed radix (2,4) twiddle addresses
MXT216n (0 to 3)	Mixed radix (2,16) twiddle addresses
MXT416n (0 to 3)	Mixed radix (4,16) twiddle addresses
MXT2416n (0 to 3)	Mixed radix (2,4,16) twiddle addresses
RBF0	Digit-reversed data address column 0
Separation Passes	
BRFTL	2-at-a-time real FFT separation pass, 2N, load
BRFTLS	2-at-a-time real FFT separation pass, N, load
BRFTU	2-at-a-time real FFT separation pass, 2N, unload
BRFTUS	2-at-a-time real FFT separation pass, N, unload
BFCTL	Fast cosine transform separation pass data addresses, 2N (long), load
BFCTT	Fast cosine transform separation pass twiddle addresses, 2N
BFCTUS	Fast cosine transform separation pass data addresses, N (short), load
BFCTU	Fast cosine transform separation pass data addresses, 2N, unload
BFCTUP	Fast cosine transform separation pass data addresses, 2N, unload using PO flag
BFCTUEP	Fast cosine transform separation pass data addresses, 2N, unload using early PO flag
BFCTULP	Fast cosine transform separation pass data addresses, 2N (long), unload using late PO flag
Decimation	
DECIM	Decimate
ADECIM	Auto-decimate

MNEMONIC	DESCRIPTION
Finite Impulse Response (FIR) Filters	
LPFIR1	Linear phase FIR, odd length, even symmetry
LPFIR2	Linear phase FIR, even length, even symmetry
LPFIR3	Linear phase FIR, odd length, odd symmetry
LPFIR4	Linear phase FIR, even length, odd symmetry
General Purpose Addressing/Utilities	
NOP	No operation
INC	Index increment
MODINC	Modulo increment
MODDEC	Modulo decrement
INTER	Interpolate/index fill
INTERP	Interpolate/index fill using PO flag
INTEREP	Interpolate/index fill using early PO flag
INTERLP	Interpolate/index fill using late PO flag
PADHIGH	Pad at end of sequence
PADHIGHP	Pad at end of sequence using PO flag
PADHIGHEP	Pad at end of sequence using early PO flag
PADHIGHLP	Pad at end of sequence using late PO flag
PADLOW	Pad at start of sequence
PADLOWP	Pad at start of sequence using PO flag
PADLOWEP	Pad at start of sequence using early PO flag
PADLOWLP	Pad at start of sequence using late PO flag
OVERLAP	Overlap
DISCARD	Discard
DISCARDP	Discard using PO flag
DISCARDEP	Discard using early PO flag
DISCARDLP	Discard using late PO flag
CMAG	Square of magnitude of a complex number
Circular Buffer Addressing	
CBUFFIR	Circular buffering for FIRs
CBUFFFT	Circular buffering for FFTs
Other	
CLRSIG	Clear signature
VIEWSIG	View signature

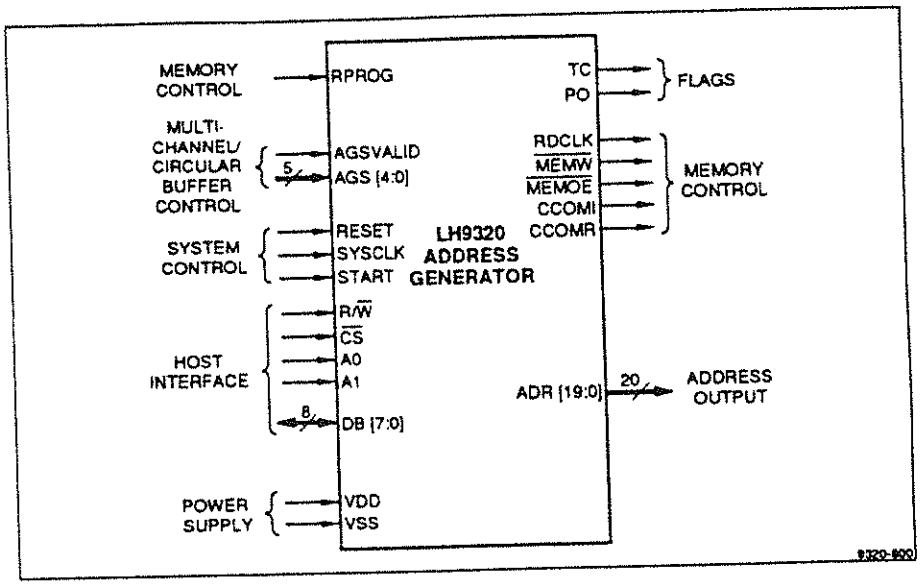


FIG. 6

LH9320 Signal-Pin Descriptions

SIGNAL *	DIRECTION **	SIGNAL NAME
DB[7:0]	I/O	Host Interface data bus
A1, A0	I	Host control
R/$\overline{W}$	I	Read/write host
$\overline{CS}$	I	Chip select
SYSCLK	I	System clock
START	I	Start of a pass
TC	O	Terminal count
ADR[19:0]	O	Address outputs
AGS[4:0]	I	Circular buffer channel sample request
AGSVALID	I	Circular buffer valid channel flag
CCOMR	O	Coefficient complement real
CCOMI	O	Coefficient complement imaginary
RESET	I	Chip reset
PO	O	Programmable output
$\overline{MEMW}$	O	Memory write
$\overline{MEMOE}$	O	Memory output enable
RDCLK	O	Read clock
RPROG	I	Programmable Pulsewidth for $\overline{MEMW}$
VDD	P	+ 5 volt power supply
VSS	P	Ground for the chip

* Numbers in brackets indicate the number of bits. For example, ADR[19:0] indicates 20 bits.
** I = Input, O = Output, I/O = Input and Output, P = Power.

TAV. 3

discesa. Questo porta a rendere disponibile un dato dopo un intero ciclo di clock (transizione basso --> alto e alto --> basso). La gestione, da parte del controllore di sistema, di questo segnale deve essere particolarmente curata per non introdurre conflitti sul bus per il fatto che questo segnale gestisce anche la direzione dei dati sullo stesso (R/W=1).

- **CS (Chip Select).** Ingresso, attivo basso. Un livello basso su questo pin, porta il chip ad essere abilitato a comunicare con il controllore attraverso il meccanismo appena descritto. Un livello alto su questo pin, porta il bus DB[7:0] in 3-state disabilitandolo.

B) Segnali di controllo.

- **SYSCLK (System Clock).** E` il clock di sistema (40 MHz), responsabile della sincronizzazione e controllo di tutte le operazioni. Tutti i segnali di uscita, vengono generati dai fronti di salita di tale clock.

- **RESET (Chip Reset).** Ingresso, attivo alto. Tramite questo pin, e` possibile resettare tutto il sistema ponendolo a livello alto; e` chiaro che durante il normale funzionamento, questo pin deve essere forzato a livello basso. Affinche` tale operazione di system reset sia effettiva, occorre attendere almeno due cicli di SYSCLK e due cicli di R/W.

- **START (Start of Pass).** Ingresso, attivo sul fronte di salita. Al fronte di salita di questo segnale viene dato il via alla generazione degli indirizzi della sequenza programmata. La prima sequenza in uscita viene fornita dopo 5 cicli di clock (latenza=5) dallo START, mentre le successive si ottengono senza, in pratica, nessuna latenza se il relativo segnale di START viene dato mentre e` ancora attiva la sequenza precedente.

C) Segnali di Controllo della Memoria.

- **MEMW (Memory Write).** Uscita, attiva bassa. Questo segnale controlla la scrittura dei dati sulla memoria indirizzata dallo stesso LH9320.

MEMW=1 --> dopo un reset, durante la lettura dati dal buffer.

MEMW=0 --> durante la scrittura del buffer di memoria.

Il livello di questo pin e` alto subito dopo un system reset e quando si leggono i dati dalla memoria. E` a livello basso quando si scrivono i dati sui buffer di memoria.

- **MEMOE (Memory Output Enable).** Uscita, attiva bassa. Il livello di questo pin, controlla la direzione del flusso dei dati verso o dalla memoria:

MEMOE=1 --> dopo un reset, scrittura del buffer.
MEMOE=0 --> lettura del buffer.

- **RDCLK (Read Clock).** Uscita. Questo segnale e` normalmente basso. Il fronte di salita indica che i dati presenti sulle RAM sono pronti per essere letti. Questo segnale viene usato per il pilotaggio degli ingressi CLKxIN del DSP LH9124.

- **CCOMR (Coefficient Complement Real).** Uscita, logica positiva.

- **CCOMI (Coefficient Complement Immaginary).** Uscita, logica positiva.

Ambedue i segnali, vengono usati in combinazione con alcune istruzioni per il pilotaggio diretto degli ingressi CCR e CCI dell' LH9124. Mediante l'uso di questi segnali, e` possibile ottenere coefficienti su 360 gradi, memorizzandone solo 90 gradi. Dopo un segnale di system reset e durante l'esecuzione di istruzioni che non hanno il controllo di CCOMR e CCOMI, questi vengono posti in 3-state.

- **RPROG (Programmable MEMW Pulsewidth).** Ingresso. Questo pin viene usato per definire la giusta temporizzazione per il segnale MEMW, in modo da garantire che rimanga a livello alto per tutta la durata della generazione della sequenza di indirizzi. Tipicamente, questo segnale viene portato a livello alto tramite una resistenza da $10K\Omega$ che consente di ottenere le temporizzazioni consigliate dalla Sharp.

D) Segnali di Flag.

- **TC (Terminal Count).** Uscita, logica positiva. Questo segnale indica la fine della sequenza di indirizzamento corrente. Questo segnale passa da livello basso a livello alto, un ciclo di clock prima della fine della sequenza per poi rimanervi fino al successivo segnale di START per la generazione della nuova sequenza di indirizzamento relativa al passo che segue. E` possibile ritardare di un ciclo la sopradetta transizione, agendo su di un particolare registro dell' Address Generator stesso.

- **PO (Programmable Output).** Uscita, logica positiva. Il segnale PO lavora solo in congiunzione ad alcune istruzioni e rende possibile la indicizzazione dei dati. E`, ad esempio, possibile attivare PO ogni M indirizzi generati per un tempo pari ad un certo numero di cicli di clock. Oppure si puo` programmare LH9320 in modo tale che PO sia attivo per un certo numero di cicli prima dell' inizio di una determinata sequenza, o lo stesso numero di cicli dopo la fine della stessa sequenza, ecc..... Pilotando in modo opportuno con PO i pin di LH9124 DCI,DCO,DZI o DZO, si possono effettuare decimazioni, interpolazioni o riempimenti (padding) con zeri dei dati da elaborare.

Per un piu` approfondito esame sull'uso del chip set SHARP si deve comunque fare riferimento al rapporto interno IRA [6]. Per terminare questa rapida descrizione del chip set, si riporta in Fig. 7 la configurazione base d'uso dei due chip nella versione singolo DSP e nelle Fig. 8 e 9 nella versione multi DSP.

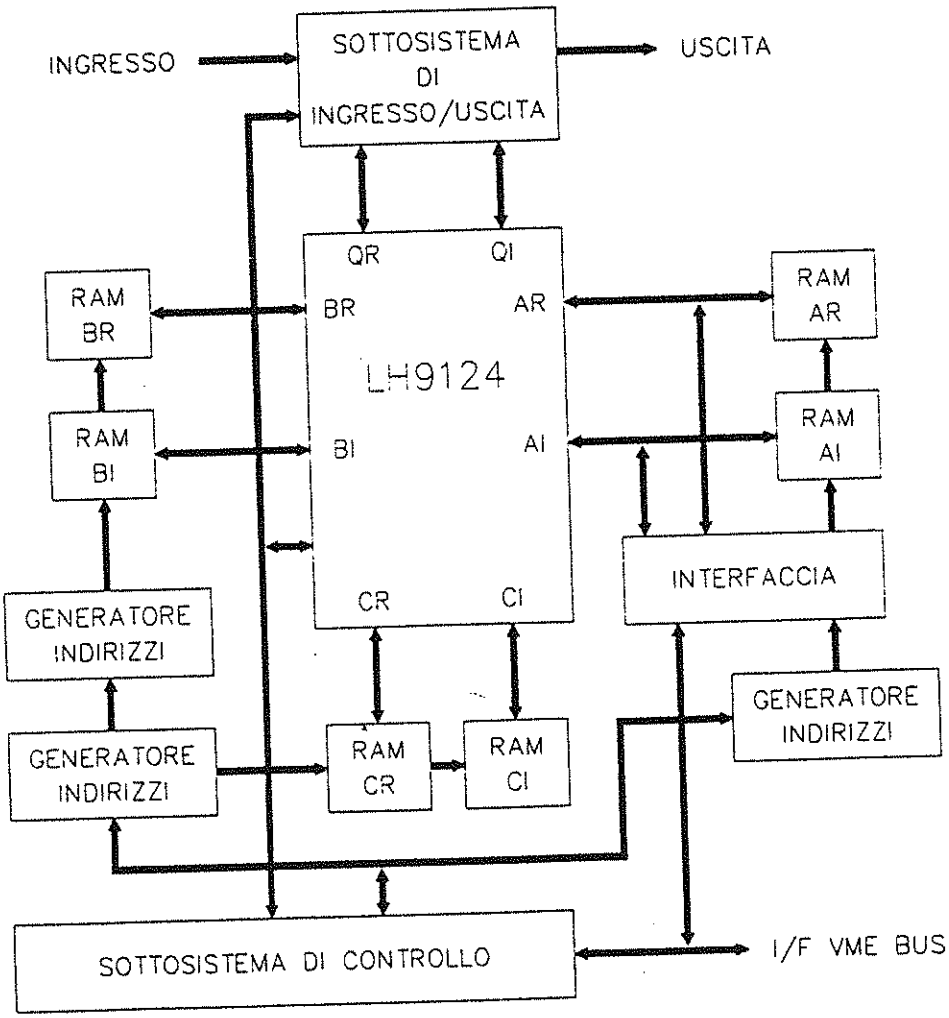

INGRESSO
SOTTOSISTEMA DI INGRESSO/USCITA
USCITA
QR
QI
BR
AR
LH9124
BI
AI
CR
CI
RAM BR
RAM BI
RAM AR
RAM AI
INTERFACCIA
GENERATORE INDIRIZZI
GENERATORE INDIRIZZI
GENERATORE INDIRIZZI
RAM CR
RAM CI
SOTTOSISTEMA DI CONTROLLO
I/F VME BUS

FIG. 7

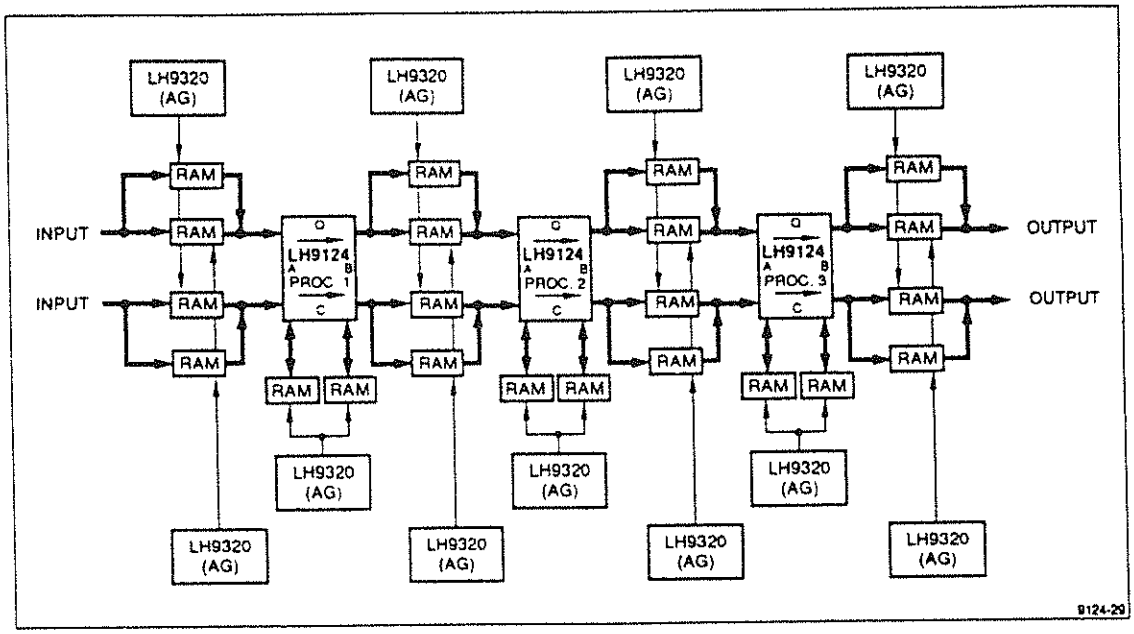

LH9320 (AG)
RAM
INPUT
INPUT
LH9124 PROC. 1
LH9124 PROC. 2
LH9124 PROC. 3
OUTPUT
OUTPUT
9124-29

FIG. 8

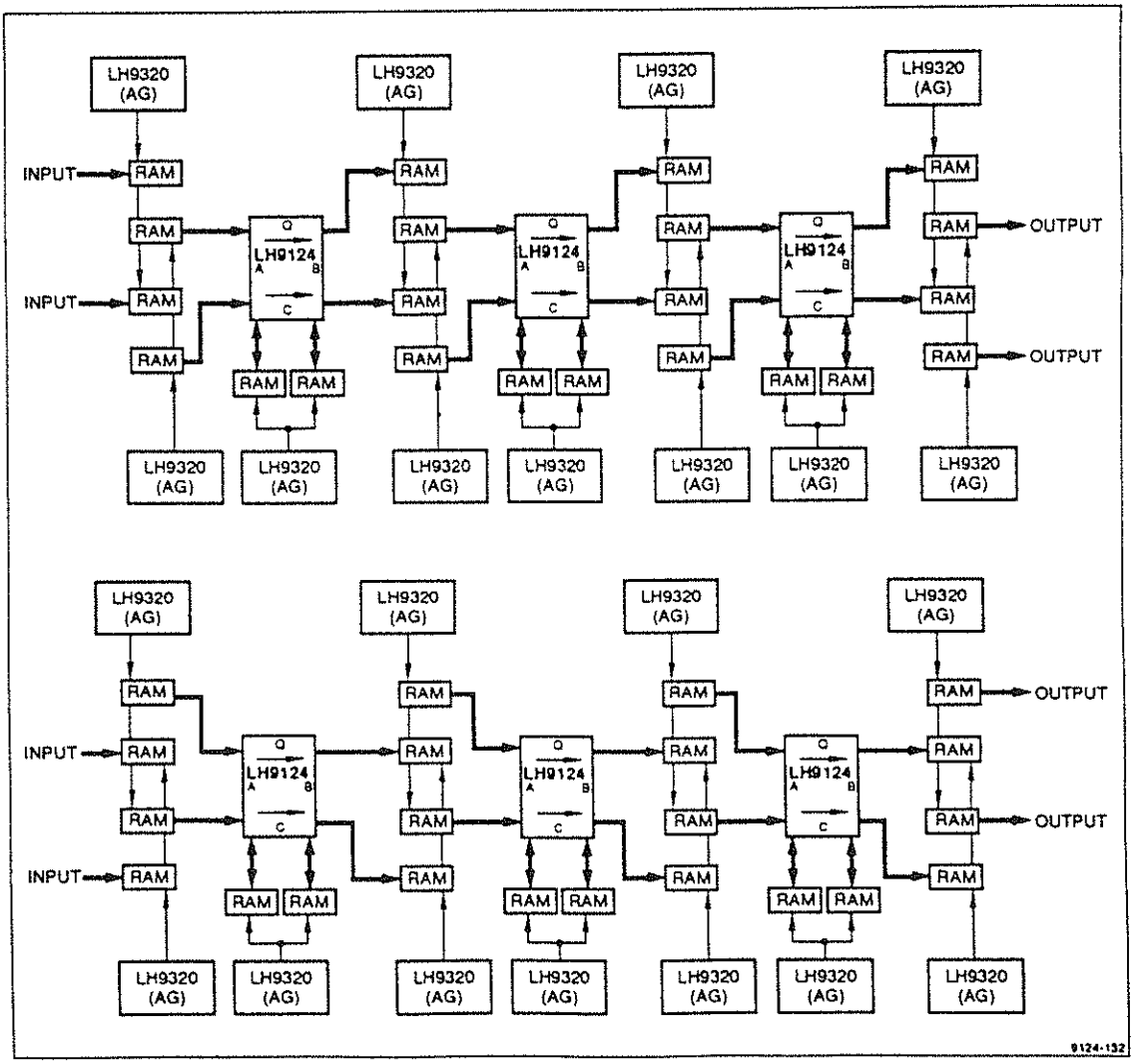

LH9320 (AG)
RAM
INPUT
INPUT
INPUT
LH9124
OUTPUT
OUTPUT
9124-132

FIG. 9

5) - IL SISTEMA.

Lo schema a blocchi del sistema viene riportato in Fig.10. Lo spettrometro è stato progettato per potere accettare in ingresso segnali provenienti sia dai moduli a larghezza di banda programmabile (banda video) del Mark III della parabola VLBI, sia direttamente dall' uscita delle medie frequenze larghe della croce con segnale centrato a 30 MHz (banda di 6 MHz) attraverso un phase shifter da 6 canali per il tracking del fascio E/W. Il segnale che arriva dal Mark III viene prima amplificato dal blocco (2) poi, se necessario, attenuato tramite un attenuatore programmabile localmente o remotamente dal calcolatore di gestione per portarlo al livello di ingresso richiesto dal convertitore A/D (blocco 4). Questo può essere letto su un indicatore posto direttamente sul pannello dello stadio di ingresso (10) oppure tramite il calcolatore. Per quanto riguarda il segnale a radio frequenza che proviene dalle medie larghe della Croce, questo entra nell' amplificatore (1) e poi nell' attenuatore programmabile (3) prima di essere convertito in banda base tramite il mixer. Per questo occore un oscillatore locale molto stabile, pulito e rapidamente sintonizzabile. A questo scopo si è progettato un generatore digitale programmabile (6) che fa uso di una evaluation board del DDS 1376 della Stanford Telecom americana [1]. Questo è in grado di fornire un segnale molto pulito, agganciato allo standard H-Maser della stazione e, soprattutto, in grado di spostarsi in frequenza in soli 200 ns ed a step minimi di 0.1 Hz. Questa ultima caratteristica è estremamente importante nel caso in cui sarà necessario osservare ad elevate risoluzioni e per lunghi periodi di integrazioni e quindi sia necessario compensare il Doppler shift introdotto dalla rotazione planetaria o qualsiasi altro tipo di accelerazioni. Un altro generatore del tipo DDS è stato approntato per essere usato come sampling generator programmabile per il convertitore A/D. Si è usato un tale tipo di generatore per la possibilità che questo offre di programmare il segnale di uscita in vasto range che, in questo caso, spazia da 1.1 Hz a 40 MHz a step di 1.1 Hz. Un terzo DDS (non ancora implementato) dovrà servire per la generazione di segnali monocromatici molto puliti e stabili per calibrazione, controllo e test su segnali che "driftano". Come nel caso dell' ingresso in banda base, vi è il blocco (5) che serve al controllo sia immediato che remoto del livello del segnale in ingresso al convertitore A/D. In tutti i casi in cui necessita la lettura dell' ampiezza dei segnali nei vari punti del sistema, si fa uso del blocco di controllo (16). Questo è composto da uno stadio di ingresso per ogni canale per l' elaborazione dei vari segnali e da un multiplexer analogico seguito da un convertitore a 16 bit gestito dalla Sparc di controllo tramite un CIO I/O Expander (14) [2] controllato a sua

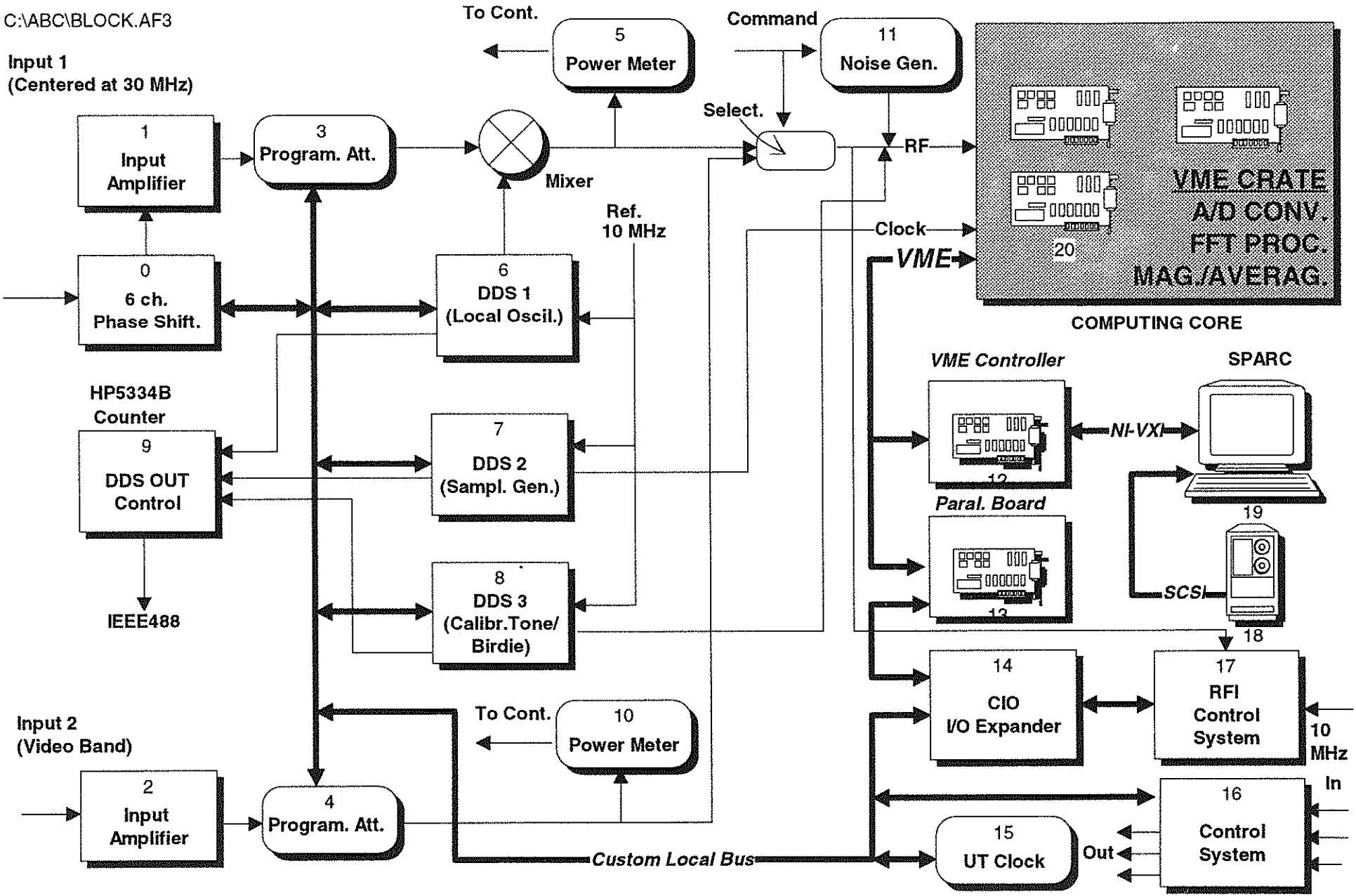

C:\ABC\BLOCK.AF3
Input 1
(Centered at 30 MHz)
1
Input
Amplifier
3
Program. Att.
To Cont.
5
Power Meter
Command
11
Noise Gen.
Select.
Mixer
RF
Ref.
10 MHz
Clock
VME
VME CRATE
A/D CONV.
FFT PROC.
MAG./AVERAG.
20
COMPUTING CORE
0
6 ch.
Phase Shift.
6
DDS 1
(Local Oscil.)
VME Controller
SPARC
NI-VXI
12
19
HP5334B
Counter
9
DDS OUT
Control
7
DDS 2
(Sampl. Gen.)
Paral. Board
SCSI
18
IEEE488
8
DDS 3
(Calibr.Tone/
Birdie)
14
CIO
I/O Expander
17
RFI
Control
System
10
MHz
Input 2
(Video Band)
To Cont.
10
Power Meter
2
Input
Amplifier
4
Program. Att.
Custom Local Bus
15
UT Clock
Out
16
Control
System
In

FIG. 10

volta attraverso un sistema di interfacciamento parallelo (13) in ambiente VME. Tramite lo stesso CIO I/O Expander vengono anche letti gli orologi di stazione (UT e Siderale), impostati i valori richiesti sugli attenuatori programmabili, impostati i valori di lavoro sui vari DDS (oscillatore locale, Sampling Generator e test) e controllo del blocco (17) per il monitoraggio delle interfernze (RFI). Il path dei dati in I/O può quindi essere riassunto come segue:

-in uscita (scrittura sui vari devices): Sparc (19) , bus NI-VXI, scheda National di controllo del VME nella slot n. 1 (12), bus VME, scheda parallela VME, CIO I/O Expander (14) ed infine, attraverso il bus custom fino al device (xx) selezionato.

- in ingresso (lettura dai vari devices): dal blocco in oggetto (xx) attraverso il bus custom fino al CIO I/O Expander, scheda parallela VME, bus VME, scheda di controllo National (slot 1) ed infine Sparc station.

Tramite la CIO è possibile configurare tutto il sistema e in più controllare che la configurazione impostata risponda a quella voluta. Infatti tutti i vari blocchi costituenti lo spettrometro sono stati progettati in modo tale da rendere possibile la lettura, su qualsiasi porta di output, di quello che si è appena scritto. Il calcolatore di controllo esegue la comparazione tra il valore impostato e quello letto, questi ovviamente devono essere perfettamente uguali. Questa filosofia di controllo sembra essere pedante, complicata e costosa ma l' esperienza fatta nell' utilizzo del sistema dimostra quanto questa sia utile a tutti i livelli, sia operativo che diagnostico. La Sparc di gestione di tutto il sistema, è una AXIL da 85 MHz collegata ad una unità nastro da 8 mm tipo Gigastore da 2 Gbyte. Nella stessa fig.10 dello schema a blocchi generale, si nota il blocco (20) che costituisce il cuore dell' intero sistema e che comprende la conversione A/D, le schede superveloci per il calcolo on line della Fast Fourier Tranform, del relativo modulo quadro ed infine, l' integrazione on-fly di un certo numero di spettri prima che il risultante spettro mediato venga presentato alla Sparc per potere poi successivamente essere immagazinato nella unita' nastro. Prima di terminare la descrizione sommaria dello schema a blocchi, soffermiamoci un attimo sul sistema di controllo delle interferenze a cui si è accennato in precedenza (17). Il segnale a radio frequenza proveniente dal radiotelescopio (sia in banda base che a 30 MHz) viene mandato a questo blocco per essere convertito da un A/D veloce tipo flash a 6 bit (100 MSampl./Sec. max). Vengono acquisiti "N" punti (di norma 1024 o 2048) su cui si calcola la FFT, il modulo quadro per ogni canale e la media di un certo numero di spettri (50/100)

mediante la Sparc AXIL. Il risultato viene poi visualizzato tramite DADiSP per avere sempre una panoramica dello stato delle radio interferenze. Tutte queste operazioni vengono effettuate dalla Sparc AXIL senza interagire minimamente con il crate VME contenente le schede di calcolo veloce che costituiscono lo spettrometro. Nella versione attuale lo strumento presenta le seguenti caratteristiche:

- System Controller:	40 MHz Sparc Comp.
- Input Bandwidth:	up to 20 MHz
- Typical Efficiency:	45% @ 6 MHz
- Number of channels:	512,..131072 (power of 2)
- Average	1/256 @ step 1
	256/64K @ step 256
- Output Format:	32 bit Float. Point
- Windows:	Hann., Hamm.,
	Kaiser-Bessel, etc..
- DSPs used:	2 x Sharp LH9124
- Mass Storage device:	~2 Gb 8 mm tape

Come si vedra` in seguito, l'attuale versione verra` migliorata in termini di time duty-cycle incrementando la banda di ingresso dagli attuali 6 MHz (45%) a 12 MHz (100%). E` anche previsto un incremento notevole del numero di canali. Non e` ancora chiaro se usare un FFT processor con maggiore quantita` di memoria (FFT calcolate su piu` punti) o se fare precedere al blocco FFT un banco di filtri digitali di tipo polifase. Tutti i programmi di gestione del sistema sono stati scritti in "C".

6) - DESCRIZIONE DEI BLOCCHI SINGOLI

Abbiamo descritto nel precedente paragrafo lo schema a blocchi del sistema, affrontiamo ora una descrizione sommaria dei blocchi singoli più significativi da un punto di vista funzionale. Una approfondita descrizione di questi ultimi è già stata fatta in una serie di rapporti interni (vedere bibliografia) mentre per i rimanenti si procederà alla descrizione, sempre all' interno della stessa serie, in un prossimo futuro. Per quello che riguarda il software di gestione dei vari blocchi, verra` riportato alla fine di ogni descrizione del blocco singolo il listato del relativo sorgente. Come già accennato all' inizio, il sistema è stato finalizzato alle osservazioni dell' impatto Giove/SL-9, per cui sia la realizzazione che un primo collaudo sono stati fatti a tempo di record (vista la scadenza a breve termine) per potere essere, in un qualche modo, operativi alla data dell' inizio degli impatti. Nella scrittura del software non si sono ricercate, quindi, le finezze o le forme più eleganti ma piuttosto quelle più rapide che permettessero di arrivare al controllo dello spettrometro nel modo più completo possibile in tempo utile per l'impatto. Tutto il software di controllo è stato scritto in linguaggio C. E` probabile che per l'integrazione dei moduli scritti nel programma finale di gestione dello spettrometro si presenti la necessita` di rivederli parzialmente o totalmente.

- ***Phase Shifter.***

Lo stadio di ingresso (Fig.11a) a 30 MHz del sistema e` cositituito dal blocco 0 [7] di Fig.10. Questo banco di sfasatori si rende necessario, se si opera con il ramo E/W, per aumentare di circa un fattore 6 il tempo di integrazione mediante il tracking del fascio B. Il banco e` realizzato in tecnica strip-line con switches allo stato solido estremamente veloci (50 ns.) tipo YSW-2-50DR della Mini Circuit. Ogni canale ha 6 stadi singoli di sfasamento (6 bit) e quindi in grado di fornire uno step di fase minimo di 360/64=5.6 gradi. Una apposita interfaccia ha il compito di gestire la inserzione o meno degli step sotto il controllo del calcolatore di sistema attraverso il blocco CIO che verra` descritto in seguito. Ogni canale del banco di filtri offre la possibilita`di prelevare il segnale di uscita sia direttamente che attraverso un accoppiatore direzionale (attenuato di 10 dB) per ulteriori controlli od elaborazioni. Particolare cura e`stata prestata nella progettazione per avere variazioni di ampiezza il piu` piccolo possibile (in centro banda) su tutte le 64 combinazioni (mediamente < 0.06 dB), precisione di fase dello step singolo (< 0.1 gradi), return loss > 24 dB, perdita di inserzione di 2.75 dB e precisione sulla massima fase di +/- 0.4 gradi.
Viene riportato qui di seguito il modulo software usato per il rifasamento del beam B del ramo E/W (phase.c). Le fasi che il modulo usa sono memorizzate in un file (phase.dat) e sono quelle fornite dal programma "Widecal".

C:\ABC\PHASE.AF3

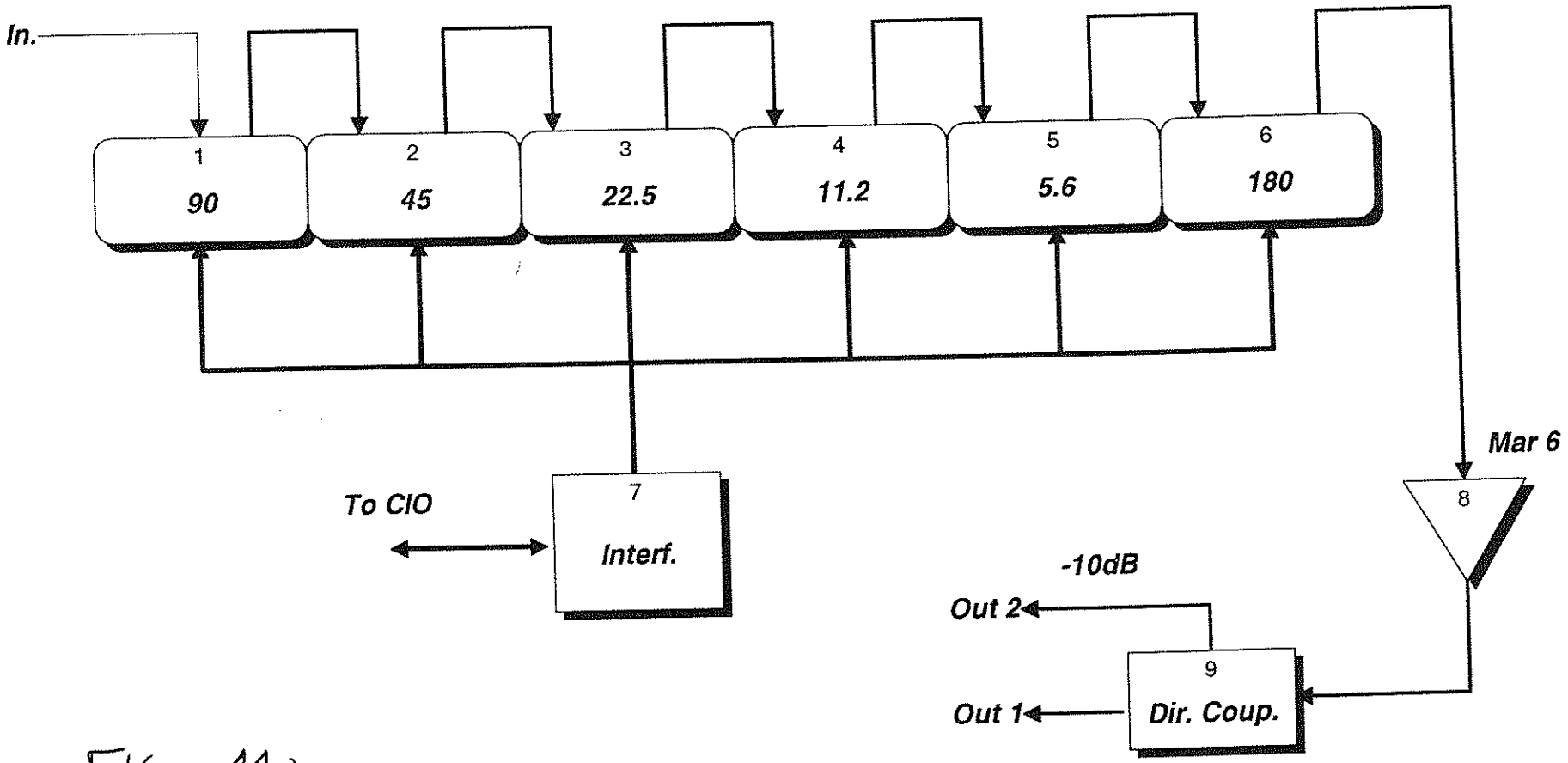


FIG. 11a

- ***Stadi di Ingresso.***

La sezione di ingresso RF del sistema è rappresentato dai blocchi (1) e (2) di Fig.10. Il blocco (1) è lo stadio di ingresso a 30 MHz dedicato ai segnali che provengono dalla Croce del Nord ed il cui schema a blocchi appare in Fig. 11b. Questo è composto da sommatori che formano, una volta aggiustata la fase, i fasci E/W e N/S. Esiste la possibilità di equalizzare i segnali somma dei rami prima della conversione, per poi poterli usare o come rami separati o sommati insieme per operare in fascio globale. E' prevista una ulteriore possibilità di variare in ampiezza il segnale in uscita e di misurarlo con un sistema che rivela, integra e scala di un opportuno fattore. Si ha in tal modo una indicazione sulla potenza del segnale inviato al sistema di elaborazione digitale. Una volta che il livello del segnale e` stato impostato a quanto richiesto viene convertito in banda base con DDS2. E' possibile iniettare una marca di calibrazione di ampiezza nota, tramite il DDS 3 (8) di Fig.10 e/o una marca di rumore anch' essa calibrata. Il sistema è completamente controllabile sia localmente che remotamente tramite il "custom local bus" che fa capo al CIO I/O Expander ed al controllore di sistema. Come visibile nella stessa Fig. 10, i livelli nei vari punti vengono controllati da un sistema di acquisizione dati locale che verra` descritto in seguito.
Lo stadio di ingresso RF (2) (Fig.10) è adatto a segnali già convertiti in banda base tipo quelli che provengono dal sistema programmabile Mark III della parabola VLBI. Questo stadio contiene 3 canali di ingresso perfettamente identici e dà l' opportunità di scegliere l' uscita da mandare al sistema di conversione A/D. Con un sistema di misura simile al precedente si ha, per ogni canale, un valore indicativo della potenza. E' prevista per ogni canale la possibilità di attenuare o meno il segnale RF, operando sugli attenuatori digitali o in modo locale o in remoto tramite il calcolatore di sistema attraverso lo stesso bus locale visto in precedenza. Lo stesso stadio rende poi disponibili i segnali di ingresso già condizionati anche (-6 dB rispetto alla uscita principale) su uscite secondarie per potere essere usati per controlli vari.
Segue il listato del modulo software di controllo dello stadio descritto (file: rfcontr.c).

```
/*  RADIO FREQUENCY BLOCK CONTROL*/
/* Program Name: rfcontr.c */
#include <stdio.h>
#include "nivxi.h"
#include "spet.h"

void att_cont(which_att,att)
char which_att;
uint16 att;


      {
        uint16 ret;
        float re;


        switch(which_att)
        {
        case'1':
                    ret=subcio_in(cio_rf_read,r_ew_att,cio_rf_add);
                    re=ret;
                    re=re/10.;
                    printf("Previous Inserted Value= %.1f \n",re);
                  subcio_out(att, cio_rf_data,w_ew_att,cio_rf_add);
                    ret=subcio_in(cio_rf_read,r_ew_att,cio_rf_add);
                      re=ret;
                      re=re/10.;
                    printf("Written Value= %.1f \n", re);
                    if(ret != att)
                    {
                    printf("Be careful, error in write or read!!\n");
                    printf("Check  the LOC/REM Switch!!\n");
                    }
                    break;

        case'2':
                    ret=subcio_in(cio_rf_read,r_ns_att,cio_rf_add);
                    re=ret;
                    re=re/10.;
                    printf("Previous Inserted Value= %.1f \n",re);
                  subcio_out(att, cio_rf_data,w_ns_att,cio_rf_add);
                    ret=subcio_in(cio_rf_read,r_ns_att,cio_rf_add);
                      re=ret;
                      re=re/10.;
                    printf("Written Value=  %.1f  \n", re);
                    if(ret != att)
                    {
                    printf("Be careful, error in write or read!!\n");
                    printf("Check  the LOC/REM Switch!!\n");
                    }
                    break;

        case'3':
                    ret=subcio_in(cio_rf_read,r_glob_att,cio_rf_add);
                    re=ret;
                    re=re/10.;
                    printf("Previous Inserted Value= %.1f \n",re);
                  subcio_out(att, cio_rf_data,w_glob_att,cio_rf_add);
                    ret=subcio_in(cio_rf_read,r_glob_att,cio_rf_add);
                      re=ret;
                      re=re/10.;
                    printf("Written Value=  %.1f  \n", re);
                    if(ret != att)
                    {
                    printf("Be careful, error in write or read!!\n");
                    printf("Check  the LOC/REM Switch!!\n");
                    }
                         }

      }


void onoff_rf(on_off)
```

```
int on_off;

 {
   uint16 ret;
          subcio_out(on_off, cio_rf_data,w_sing_sw,cio_rf_add);
          ret=subcio_in(cio_rf_read,r_sing_sw,cio_rf_add);
          if(ret != on_off)
          printf("Be careful, error in write or read!!\n");
            printf(" Valore entrato %d \n", on_off);
}


 main()

{
int response;
uint8 result;
int on_off;
uint8 retval;
char which_att;
float val;
uint16 att;
char flag;


resetvmio12();

continue_prog: result=InitVXIlibrary();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/
enable('2');
enable('3');

loop: printf("============================================================\n ");
printf("Switch->0   Attenuator->1 \n");
scanf("%d", &response);
if(response==0)
{
 printf("Wich sw to turn ON ?  \n");
 printf("Enter in Binary Mode  ( 1->1on 2->2on 8->3on ect.. ) \n");
scanf("%d", &on_off);
onoff_rf(on_off);
goto loop;
}


cont: printf("Wich attenuator? (E/W->1 N/S->2 GLOB->3) \n");
 scanf("%1s", &which_att);
        if(which_att>'3') goto cont;


again: printf("Wich Attenuation value? (0->25.5 dB format XX.x) \n");
scanf("%f", &val);
           if(val > 25.5)
                          {
                           printf("Entered Value greater than 25.5 \n");
                           printf("Enter Again the Value \n ");
                           goto again;
                           }

val = val*10.;
att=val;
att_cont(which_att,att);


result=CloseVXIlibrary();

goto continue_prog;


}
```

```
/*  RADIO FREQUENCY BLOCK CONTROL*/
/* Program Name: rfcontr.c */
#include <stdio.h>
#include "nivxi.h"
#include "spet.h"


 onoff_rf(which_sw, on_off)
char which_sw;
uint8 on_off;


{

        switch(which_sw)
        {
        case'1':
                subcio_out(on_off, cio_rf_data,w_sing_sw,cio_rf_add);
                   break;

        case'2':
              subcio_out(on_off, cio_rf_data,w_sing_sw,cio_rf_add);

        }

}


 main()

{
uint8 result;
uint8 on_off;
uint8 retval;
char which_att;
char which_sw;
float val;
uint16 att;

resetvmio12();

continue_prog: result=InitVXIlibrary();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/
enable('2');
enable('3');

printf("Wich ? (Noise Source->1 VLBI Ant.->2 \n");
scanf("%1s", &which_sw);
printf("ON->1  OFF->0 \n");
scanf("%d", &on_off);
onoff_rf(which_sw, on_off);


result=CloseVXIlibrary();

goto continue_prog;

}
```

- ***DDS 1 (Oscillatore Locale)*** [4].

L' oscillatore locale della parte di interfacciamento Croce / Spettrometro (6) deve possedere la stabilità, purezza spettrale, step di variazione minimo e velocità di sintonizzazione richieste dal tipo di impiego. La grande stabilità e purezza spettrale sono richiesta dal fatto che lo spettrometro potrebbe essere usato ad elevate risoluzioni (1 Hz o meno) per cui l' oscillatore non deve assolutamente "scivolare" all' interno della larghezza di canale. Per questo motivo anche l' oscillatore locale della Croce sarà sostituito da un oscillatore locale tipo DDS sincronizzato, come quello in questione, dallo standard di frequenza di stazione. Lo step minimo di variazione è richiesto essere molto piccolo (< 0.2 Hz) e veloce da impostare per potere compensare la variazione Doppler dovuta alla rotazione planetaria quando si opera ad elevate risoluzioni frequenziali e a frequenze elevate (22 GHz o più). I tempi di impostazione della frequenza selezionata nel DDS usato sono dell' ordine di 250 nS. Si intuisce, per quanto detto sopra, che un PLL non sarebbe stato assolutamente adatto perchè, ad esempio, sarebbe stato molto difficile raggiugere step di 0.1 Hz e, soprattutto, modificare la frequenza in poche centinaia di nS. I DDS, in generale, pur non rappresentando un diretto rimpiazzo dei PLL presentano alcune caratteristiche non riscontrabili nei PLL, tipo:

- Minimo Settling Time (Fraz. di uS).
- Range operativo di molte ottave.
- Alta risoluzione in frequenza (< 1 uHz tipico).
- Coerenza di fase al variare della frequenza.
- Possibilità di generare segnali in quadratura su molte ottave.
- Basso rumore di fase.

Lo schema a blocchi di un generico DDS è rappresentato in Fig. 12. Senza entrare nel merito del funzionamento ([4]) possiamo dire che in un qualche modo il valore della fase impostato definisce l' indirizzo della look up table che contiene il relativo valore numerico dell' ampiezza da inviare al convertitore D/A (12 bit).
Nel caso dell' oscillatore locale in questione, si è fatto uso di una evaluation board della Stanford Telecom della famiglia STEL 137X (nel nostro caso 1376) il cui schema a blocchi viene riportato in Fig. 13. Questa scheda è equipaggiata con un DDS in grado di operare fino a 35 MHz con step minimi di 0.1 Hz e sincronizzabile con il riferimento a 10 MHz proveniente dal H-Maser della stazione. Il

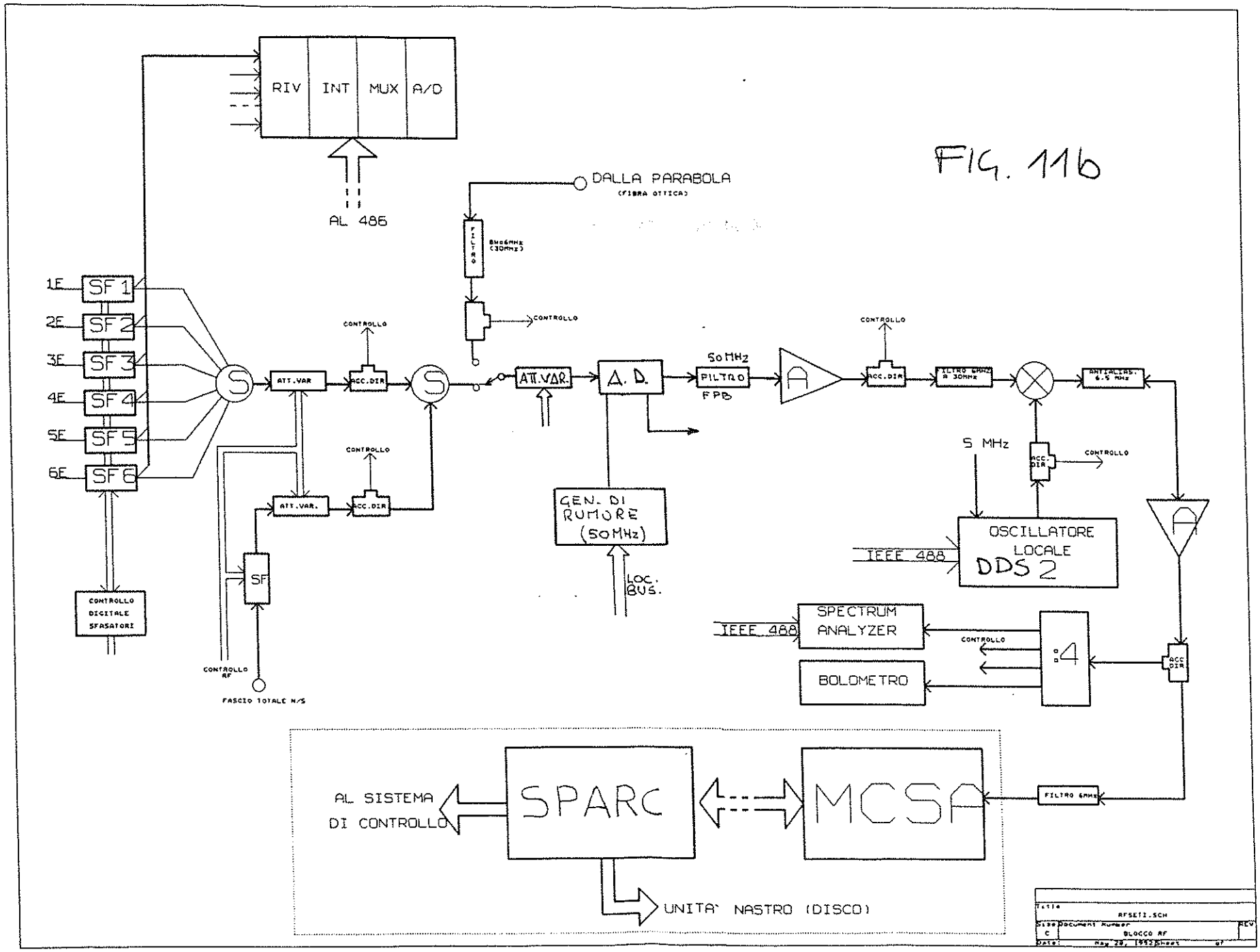

RIV
INT
MUX
A/D
AL 486
DALLA PARABOLA
(FIBRA OTTICA)
FILTRO
8÷6MHz (30MHz)
CONTROLLO
1E
2E
3E
4E
5E
6E
SF 1
SF 2
SF 3
SF 4
SF 5
SF 6
S
ATT.VAR
ACC.DIR
ATT.VAR.
A. D.
50 MHz
FILTRO
FPB
A
FILTRO 6MHz a 30MHz
ANTIALIAS 6.5 MHz
5 MHz
GEN. DI RUMORE (50MHz)
LOC. BUS.
OSCILLATORE LOCALE DDS 2
IEEE 488
SF
CONTROLLO DIGITALE SFASATORI
CONTROLLO RF
FASCIO TOTALE N/S
SPECTRUM ANALYZER
BOLOMETRO
÷4
FILTRO 6MHz
AL SISTEMA DI CONTROLLO
SPARC
MCSP
UNITA' NASTRO (DISCO)
RFSETI.SCH
BLOCCO RF
May 28, 1992

FIG. 116

modulo STEL 1376 ospita, a sua volta, il NCO (Numerically Controlled Oscillator) STEL 1176 (Fig.14) che ha il compito principale di generare la sinusoide campionata (la funzione di campionamento non è altro che il clock) multipla di 0.1 Hz. Questa è correlata alla fase che viene impostata tramite una parola di 35 bit (8 cifre e 3/4 BCD). La frequenza ottenibile in uscita Fo è data da:

$$F_0 = \frac{F_c \cdot \Delta\varphi}{8.1 \cdot 10^{-6}}$$

con Fo=frequenza di uscita e Fc=frequenza di clock.
Viene riportato in Fig.15 a,b un plottato dello spettro del segnale in uscita che si riferisce, in particolare, ad un segnale impostato di frequenza pari a 26.5 MHz. e dove si può vedere come le spurie siano a livelli di -59 dBc.
Lo schema a blocchi dell' intero oscillatore locale viene riportato in Fig.16. La parola relativa alla frequenza da programmare ed i segnali di controllo necessari per l' impostazione della stessa vengono caricati nella STEL 1370 o tramite "contraves" tipo BCD e pulsanti (local Mode) o tramite porte di scrittura collegate al local bus e gestite dal calcolatore di controllo attraverso la CIO e la scheda parallela VME. Come in tutti i rimanenti blocchi che costituiscono lo spettrometro, anche qui è stata implementata la possibilità di leggere sempre il dato appena scritto su qualsiasi porta; questo per tenere sotto controllo il tratto Sparc/porte di scrittura. Questo percorso è, in realta, abbastanza lungo e "tortuoso" in quanto una volta che il dato è partito dalla Sparc, raggiunge il Crate VME attraverso il linK National Ins. NI-VXI per poi, attraversando una scheda parallela VME, raggiungere il CIO I/O Expander e, successivamente, la sezione di decodifica dell' oscillatore locale ultima meta prima di arrivare alla porta indirizzata. L'uscita del DDS 1 e` poi filtrata ed amplificata. Tramite un accoppiatore direzionale questa viene mandata all'uscita e ad un frequenzimetro HP 5334B gestibile via IEEE488.
Viene riportato qui di seguito il listato del programma che, nella attuale versione, gestisce il setting dell' oscillatore digitale (file: dds.c).

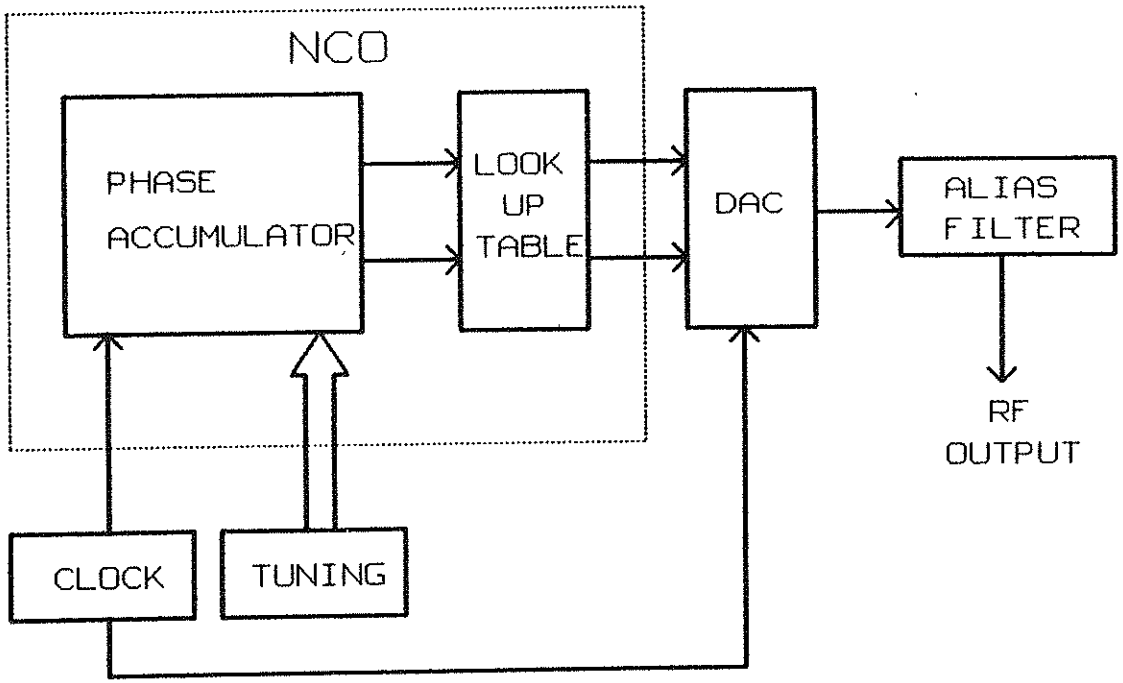


FIG. 12

-CNR- ISTITUTO DI RADIOASTRONOMIA		
Title: BASIC DDS SYSTEM		
Size: B	Document Number: FILE:DDSR1.SCH	REV:
Date: May 13, 1993	Sheet: of	

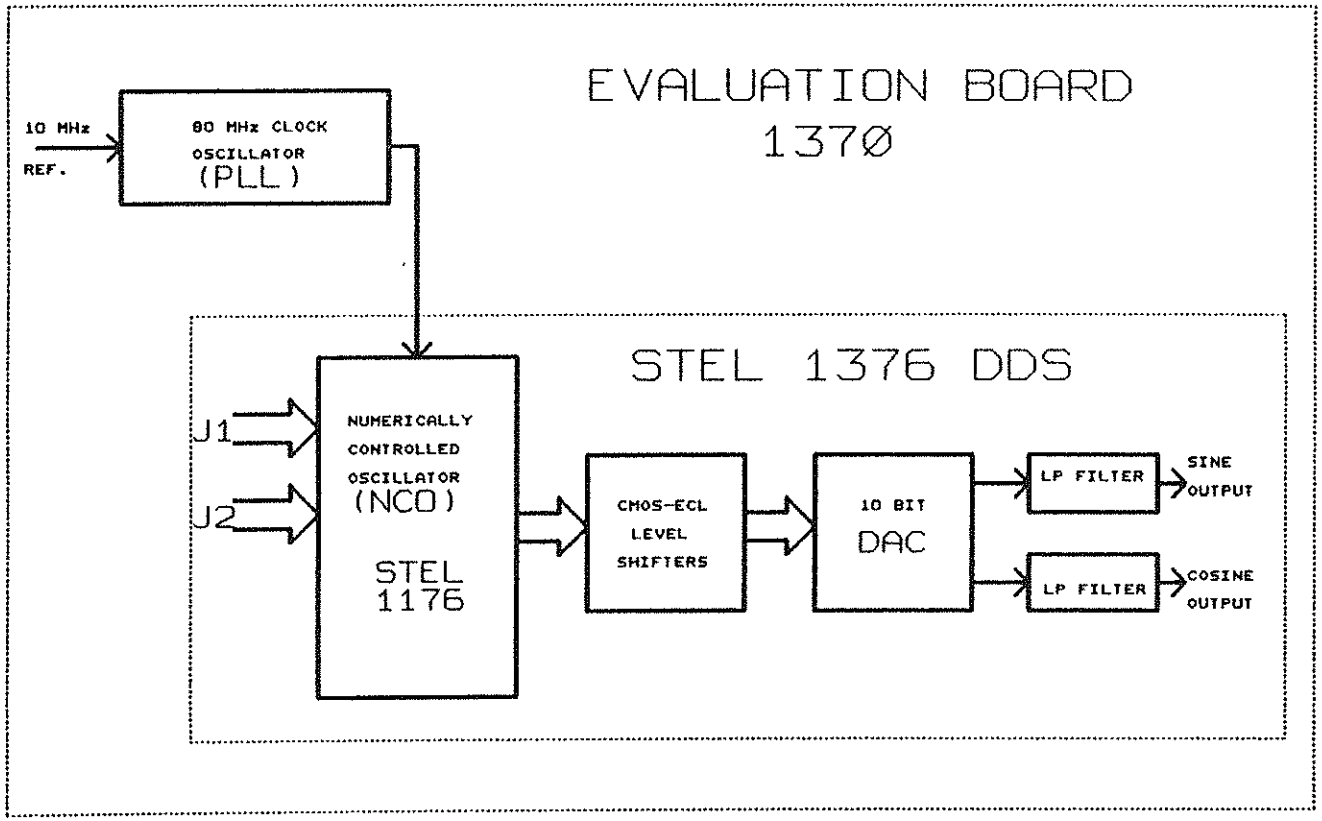


FIG. 13

-CNR- ISTITUTO DI RADIOASTRONOMIA		
Title: 1370 EVALUATION BOARD		
Size: B	Document Number: FILE:EV.SCH	REV:
Date: May 13, 1993	Sheet:	of

STEL-1176 BLOCK DIAGRAM

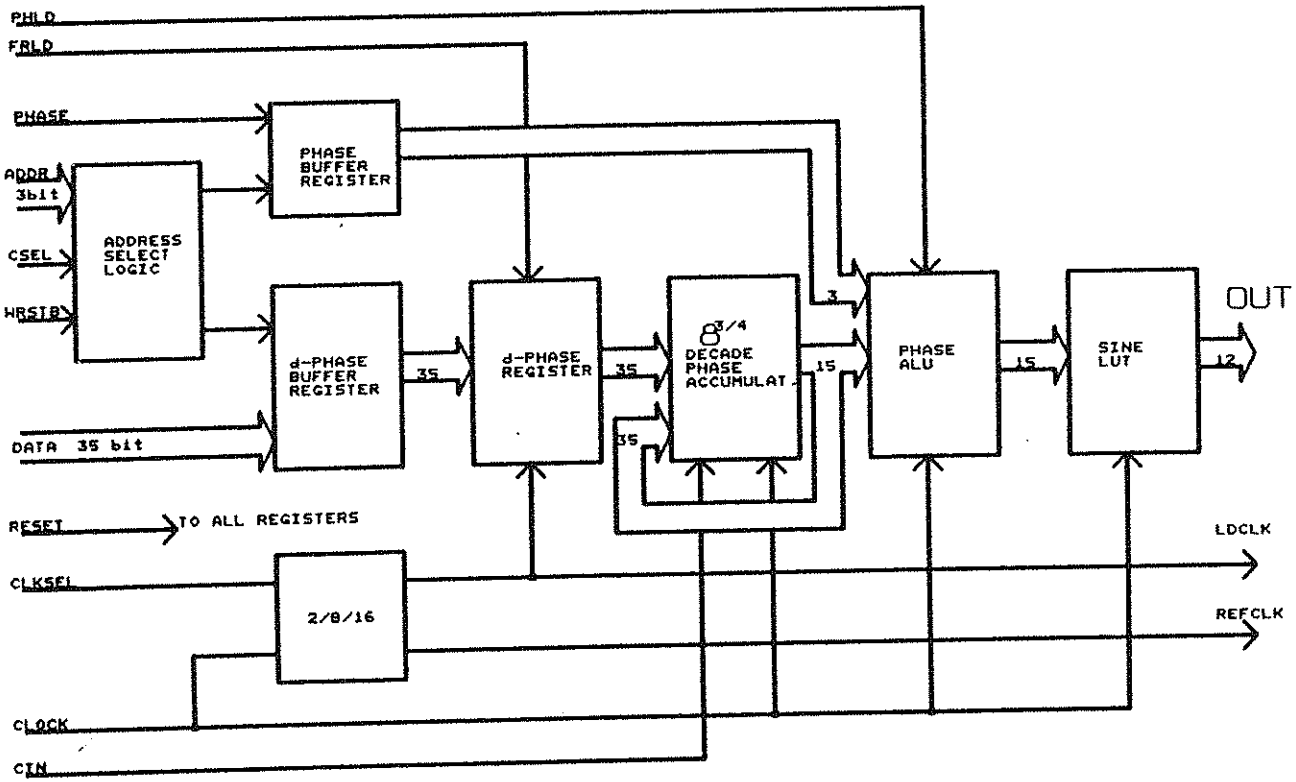


FIG. 14

-CNR- ISTITUTO DI RADIOASTRONOMIA		
Title: STEL-1176 BLOCK DIAGRAM		
Size: B	Document Number: FILE:1176.SCH	REV
Date: May 13, 1993	Sheet of	

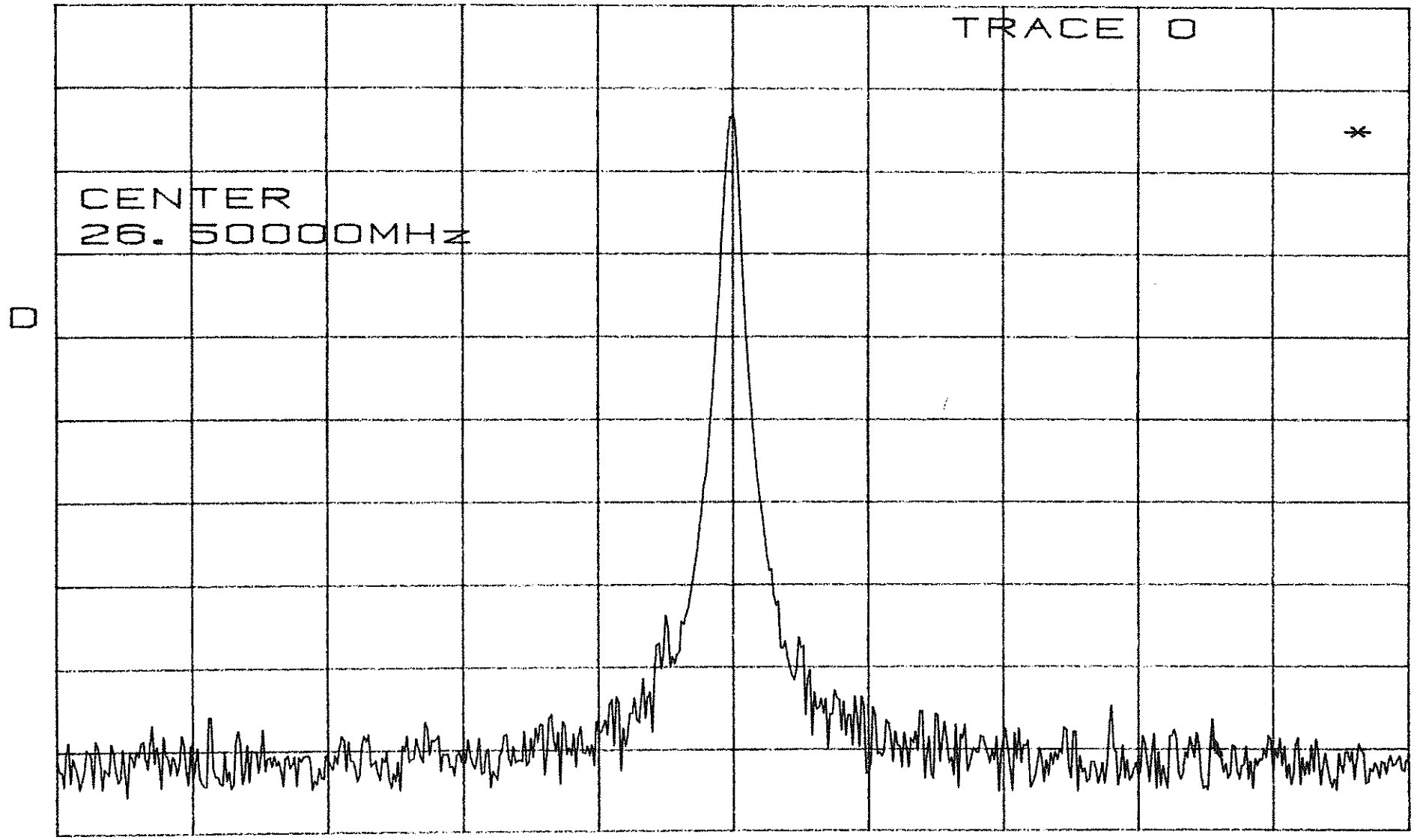

*ATTEN 50dB
RL 20.0dBm
10dB/
TRACE 0
*
CENTER
26.50000MHz
D
CENTER 26.50000MHz
SPAN 20.00kHz
RBW 100Hz
VBW 300Hz
SWP 5.0sec

FIG. 15Q

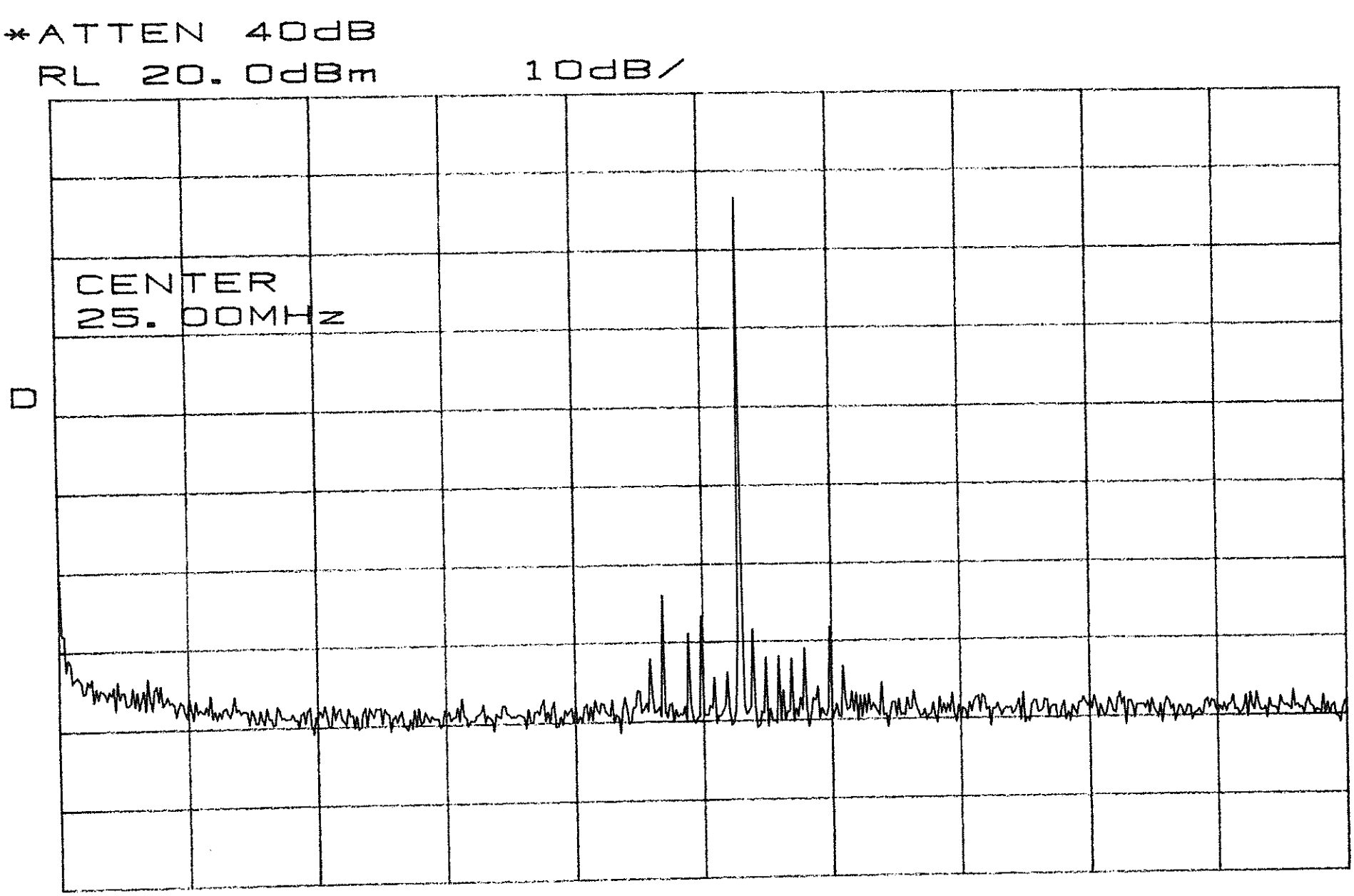
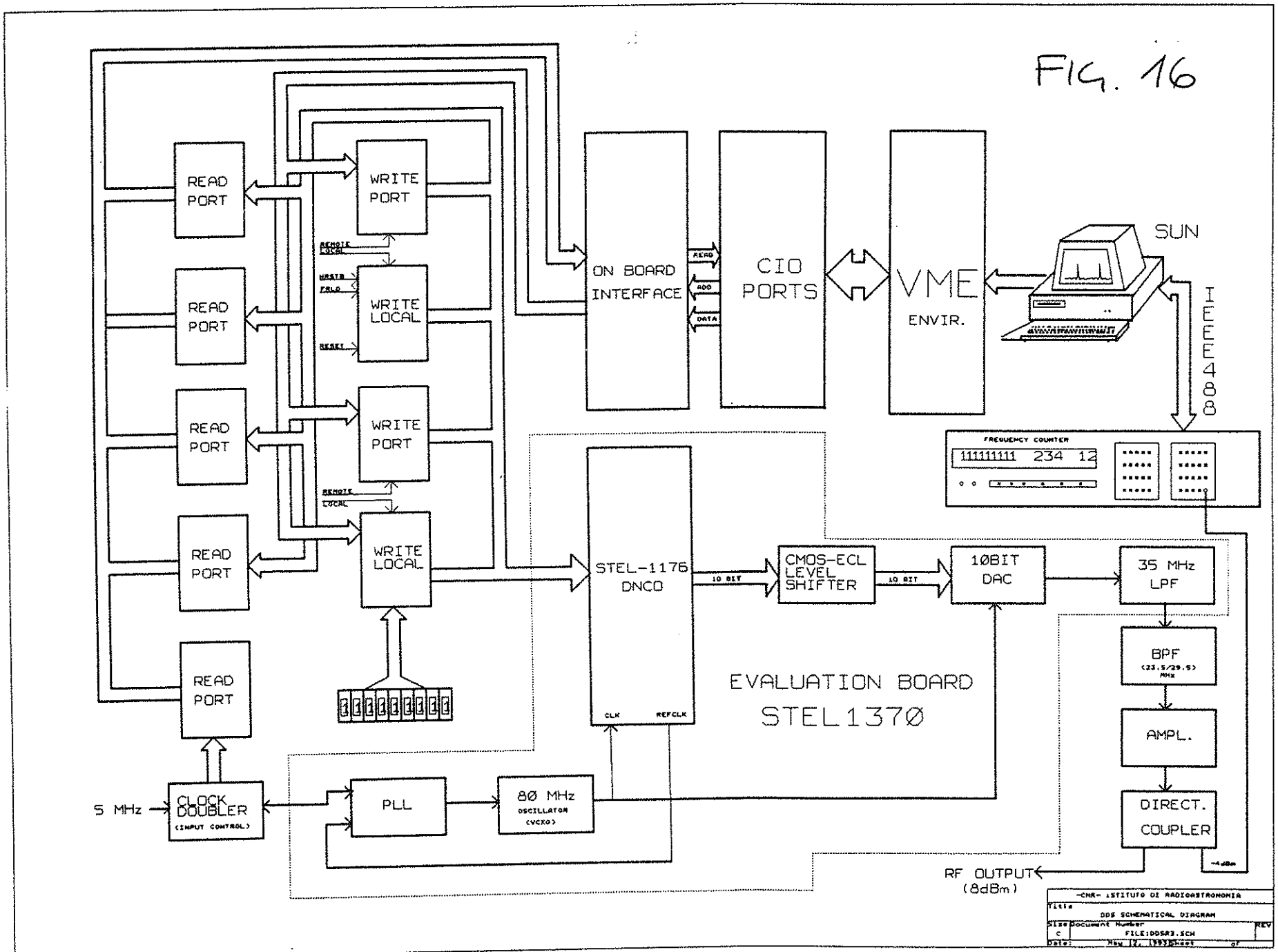

*ATTEN 40dB
RL 20.0dBm
10dB/
CENTER
25.00MHz
D
CENTER 25.00MHz
SPAN 50.00MHz
*RBW 10kHz
VBW 30kHz
SWP 2.0sec

FIG. 15b

FIG. 16


READ PORT
READ PORT
READ PORT
READ PORT
READ PORT
WRITE PORT
WRITE LOCAL
WRITE PORT
WRITE LOCAL
REMOTE LOCAL
WRITE FOLD
RESET
REMOTE LOCAL
ON BOARD INTERFACE
READ
ADD
DATA
CIO PORTS
VME ENVIR.
SUN
IEE488
FREQUENCY COUNTER
111111111 234 12
STEL-1176 DNCO
10 BIT
CMOS-ECL LEVEL SHIFTER
10 BIT
10BIT DAC
35 MHz LPF
BPF (23.5/29.5) MHz
AMPL.
DIRECT. COUPLER
CLK
REFCLK
EVALUATION BOARD
STEL1370
5 MHz
CLOCK DOUBLER (INPUT CONTROL)
PLL
80 MHz OSCILLATOR (VCXO)
RF OUTPUT (8dBm)
-CNR- ISTITUTO DI RADIOASTRONOMIA
Title
DDS SCHEMATICAL DIAGRAM
Size C
Document Number
FILE:DDSR3.SCH
REV
Date: May 12, 1993
Sheet of

```
/*  DIRECT DIGITAL SYNTHETIZER CONTROL*/
/* Program Name: dds.c */
#include <stdio.h>
#include "nivxi.h"
#include "spet.h"
#include <math.h>

void digitiz (port,value)
uint8 port;
uint8 value;

      {
        int16 ret;

                    subcio_out(value, cio_dig_data, port, cio_dig_add);
                    ret=subcio_in(cio_dig_read, port, cio_dig_add);
                    if(ret != value)
                    {
                    printf("Be careful, error in write or read!!\n");
                    printf("Check  the LOC/REM Switch!!\n");
                    }

      }


 main()

{
int response, i;
uint8 result;
int16 ret;
uint8 dataw;
float res;
int m;
FILE *inpu;
uint8 dmil, mil, cm, dm, im, ic, id, iu;
uint8 dataw1, dataw2, dataw3, dataw4;
float fdummy, freq ;
int ddummy;
uint32 dummy;
inpu=fopen("/home/stelio/cprog/ad.dat","w+");
resetvmio12();


continue_prog: result=InitVXIlibrary();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/
enable('2');
enable('3');


initvmio12('0','b','o','n'); /*port b of IC 6 becomes out (Clock Read)*/
initvmio12('1','a','i','n'); /*port b of IC 7 becomes in (data+flags)*/
initvmio12('1','b','i','n'); /*port b of IC 7 becomes in (flags)*/
enable('0');
enable('1');


/* Reset  --_- */
dataw=0;
subcio_out (dataw,cio_dig_data,refgen_reset,cio_dig_add);
dataw=1;
subcio_out (dataw,cio_dig_data,refgen_reset,cio_dig_add);



/*=====================================================================*/
/* SETTING OF THE REFERENCE GENERATOR BLOCK */

/*inp: printf("RESULUTION? (1/200 KHz) \n");
scanf("%d",&res);*/
```

```
res=0.1;
m=1000/res;


/*printf("Requested Resolution= %.3f KHz \n",res);*/


im=m/1000;
ic=(m-im*1000)/100;
id=(m-im*1000-ic*100)/10;
iu=m-im*1000-ic*100-id*10;

ddummy=im*1000+ic*100+id*10+iu;
fdummy= 1000./ddummy;
/*printf("Real Resulution= %.3f KHz \n", fdummy);*/

dataw1=iu+id*16;
dataw2=ic+im*16;

digitiz (refgen_lsb,dataw1);
digitiz (refgen_msb,dataw2);

/* Status Control   */
  ret=subcio_in (cio_dig_read, refgen_status, cio_dig_add);
/*  printf("status of Refgen. = %u \n", ret);*/

    if((ret & 0x1) == 1 )
     printf("REF. FREQ. BOARD IS IN LOCAL STATUS \n");

    if((ret & 0x2) == 0 )
      printf("5 MHz  failure \n");

/*=========================================================================*/
/*SETTING OF THE PLL FREQUENCY */

/*printf("OUTPUT PLL FREQUENCY (Freq.) ? (10.200--> 17.200 MHz (XX.xxx MHz)  \n");
printf("Out Frequency= Resolution(KHz) x Entered Freq.(MHz) (MHz)  \n");

scanf("%f", &freq);*/
fdummy=14.;
fdummy=fdummy*1000;

fdummy=fdummy/res;
ddummy=fdummy;

dm=ddummy/10000;
im=(ddummy-dm*10000)/1000;
ic=(ddummy-dm*10000-im*1000)/100;
id=(ddummy-dm*10000-im*1000-ic*100)/10;
iu=(ddummy-dm*10000-im*1000-ic*100-id*10);

fdummy=(dm*10000+im*1000+ic*100+id*10+iu)/1000.;
/*printf("Entered Frequency = %.3f \n", fdummy);*/

dataw1=iu+id*16;
dataw2=ic+im*16;
dataw3=dm;

digitiz (pll_lsb,dataw1);
digitiz (pll_msb1,dataw2);
digitiz (pll_msb2,dataw3);

/*=========================================================================*/
/*SETTING OF THE PROGRAMMABLE DIVIDER */

/*printf("Number of point for the FT ? \n");
scanf("%d", &dummy);*/

dummy=2048;
dmil=dummy/10000000;
mil=(dummy-dmil*10000000)/1000000;
cm=(dummy-dmil*10000000-mil*1000000)/100000;
dm=(dummy-dmil*10000000-mil*1000000-cm*100000)/10000;
im=(dummy-dmil*10000000-mil*1000000-cm*100000-dm*10000)/1000;
ic=(dummy-dmil*10000000-mil*1000000-cm*100000-dm*10000-im*1000)/100;
```

```
id=(dummy-dmil*10000000-mil*1000000-cm*100000-dm*10000-im*1000-ic*100)/10;
iu=(dummy-dmil*10000000-mil*1000000-cm*100000-dm*10000-im*1000-ic*100-id*10);


/* data to write */
dataw1=iu+id*16;
dataw2=ic+im*16;
dataw3=dm+cm*16;
dataw4=mil+dmil*16;

/* data writing */
digitiz (pdiv_lsb,dataw1);
digitiz (pdiv_msb1,dataw2);
digitiz (pdiv_msb2,dataw3);
digitiz (pdiv_msb3,dataw4);

/* Load and Start  --_-*/
ddummy=1;
subcio_out(ddummy, cio_dig_data, pdiv_ls, cio_dig_add);
ddummy=0;
subcio_out(ddummy, cio_dig_data, pdiv_ls, cio_dig_add);
ddummy=1;
subcio_out(ddummy, cio_dig_data, pdiv_ls, cio_dig_add);


/* PLL Status Control    */
   ret=subcio_in (cio_dig_read, pll_status, cio_dig_add);

/*   printf("PLL Status = %u \n", ret);*/

    if((ret & 0x1) == 0 )
   printf("PLL unlocked. Ceck Input Parameters! \n");



/* PROG. DIV. Status Control    */
 ret=subcio_in(cio_dig_read, pdiv_status, cio_dig_add);
/*printf("Prog. Div. Status = %u \n", ret);*/

     if((ret & 0x1) == 1 )
   printf("Board in Local Status! \n");

   if((ret & 0x2) == 2 )
   printf("Be Careful, Data FIFO Overflow!! \n");

i=1;
/*Read _-_ */

/*outvmio12('0','b',128);*/
do{
outvmio12('0','b',128);
outvmio12('0','b',0);
result=invmio12 ('1','a');
result=(result & 63);
/*outvmio12('0','b',128);*/
fprintf(inpu,"%u \n",result);
}while(i++<dummy);



/* Reset  --_-
dataw=0;
subcio_out (dataw,cio_dig_data,refgen_reset,cio_dig_add);
dataw=1;
subcio_out (dataw,cio_dig_data,refgen_reset,cio_dig_add); */



result=CloseVXIlibrary();

}
```

- ***DDS 2.***

Il convertitore A/D, all'interno del crate VME, ha bisogno di impulsi di trigger per lo start-conversion ad un valore di frequenza tale da soddisfare il teorema di Nyquist per la banda da convertire. Il convertitore Ultra ADC della Valley Tech. usato, puo` operare fino a sampling rate di 40 MHz per cui il generatore (7) deve essere in grado di fornire tali impulsi di trigger. Il range di operazione del convertitore spazia da circa 100 KHz a 40 MHz per cui il generatore di trigger deve essere, oltre che programmabile localmente e remotamente come il precedente, in grado di operare su tale gamma di frequenze (molte ottave); per cui la naturale soluzione e` rappresentata da un DDS anche se in questo caso non e` richiesta una grande risoluzione frequenziale o grande velocita`di impostazione della frequenza.
Il DDS usato in questo caso e` montato a bordo di una evaluation board della Stanford Telecom modello STEL-2272+110. Lo schema a blocchi del generatore viene riportato in Fig. 17 mentre un disegno della scheda viene mostrato in Fig. 18 . Si nota che il NCO impiegato in questo caso e` il STEL-2172, che opera a 28 bit ed e` in grado di offrire risoluzioni di 1.1 Hz in un range che spazia da 0 a 110 MHz con clock di 300 MHz. Quest'ultimo viene fornito tramite un PLL sincronizzato con la 10 MHz proveniente dallo standard H-Maser della stazione. Anche in questo caso, come nel precedente, e` possibile impostare la parola relativa alla frequenza voluta sia tramite calcolatore, con meccanismo simile a quello gia` descritto per il DDS 1, che localmente tramite contraves. Da notare che in questo caso la parola impostata localmente sui contraves, deve essere in binario. Questo non rende molto agevole tale operazione ma considerando che questa avviene raramente e comunque solo in regime di manutenzione, la cosa risulta accettabile. Come per il DDS 1, si e`dovuto progettare ad hoc una scheda in grado di comandare, quando in remote mode, la evaluation board tramite lo stesso controllore (Sparc) di sistema. In Fig. 19 viene riportato lo schema a blocchi del generatore di sincronismi DDS 2 mentre in Fig. 20 viene riportato lo spettro di un segnale a 30 MHz.
Qui di seguito e` riportato il sorgente del modulo software usato, nella attuale versione dello spettrometro, per la gestione del generatore di trigger (file: samdds.c).

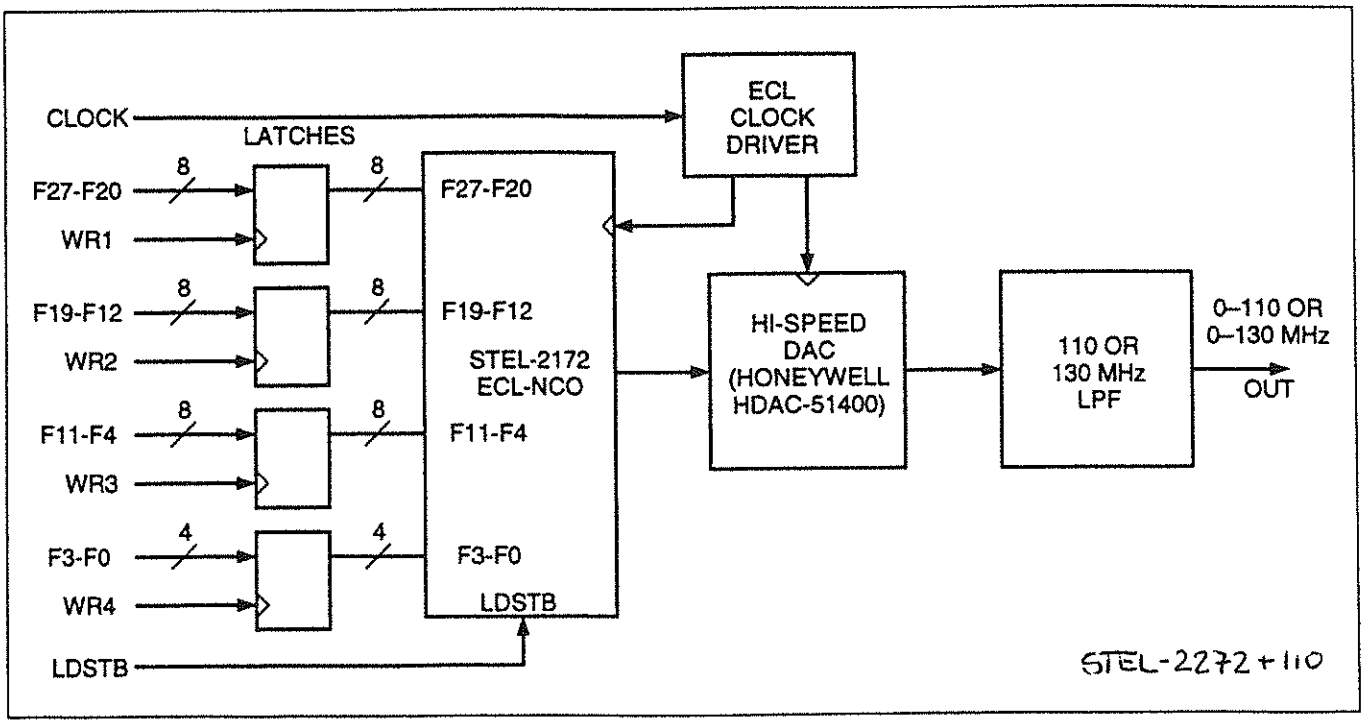


FIG. 17

MECHANICAL OUTLINE

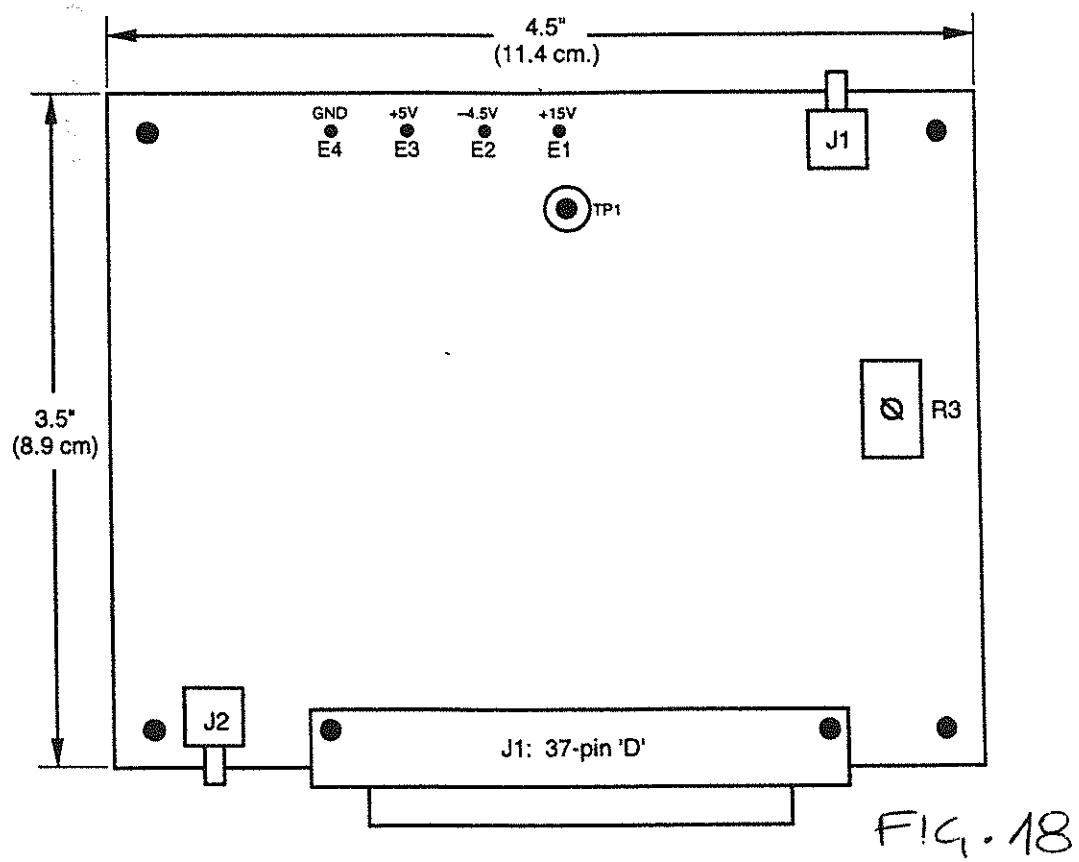


FIG. 18

Notes:
1. Corner mounting holes are located 0.2" from corners and are 0.125" diameter.
2. 'D' connector J1 protrudes approx. 0.35" beyond edge of board.
3. Test point TP1 is connected to digital ground.
4. Turning R3 clockwise decreases the output level.

C:\ABC\DDS2.AF3

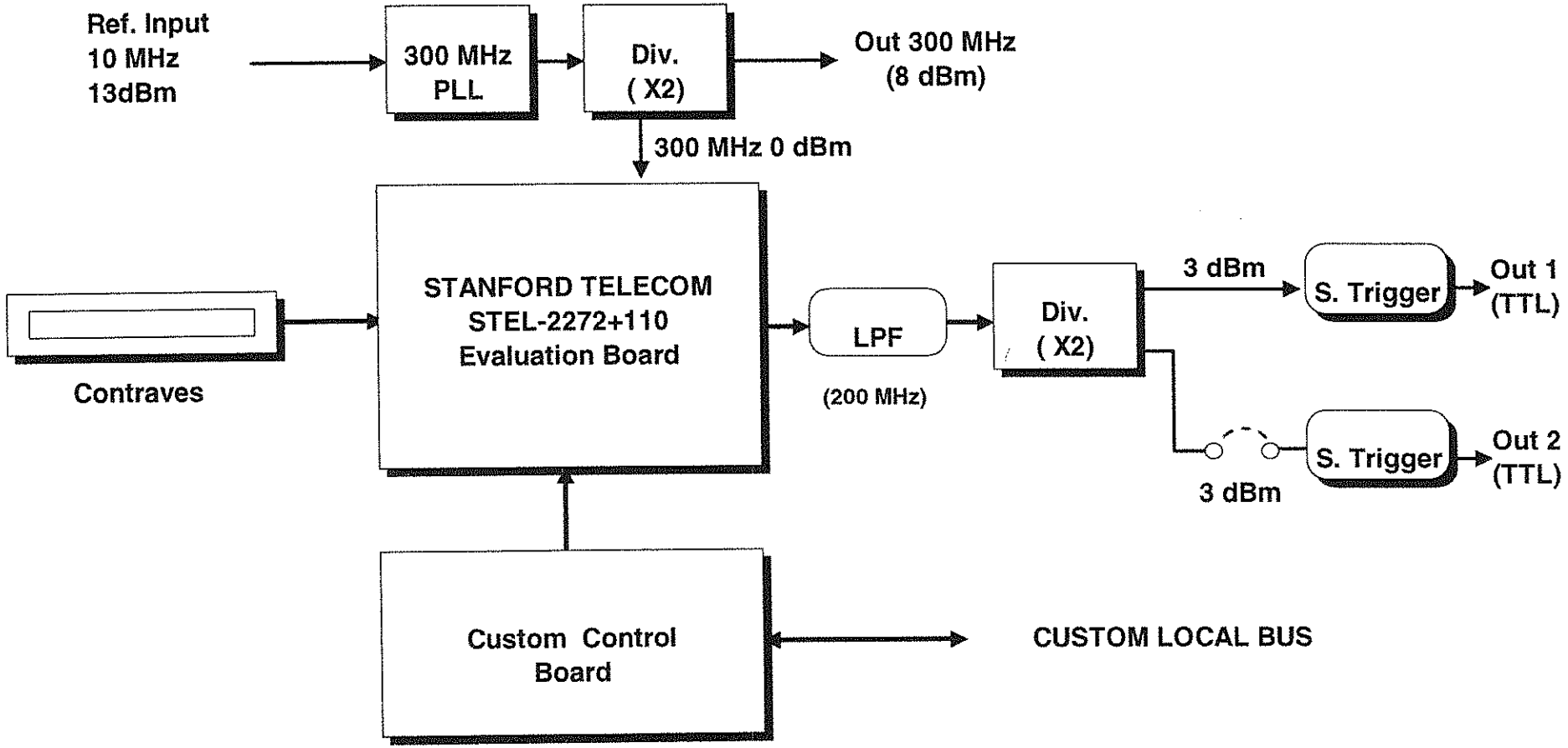


FIG. 19

TYPICAL SPECTRUM

Center Frequency:	67.89 MHz
Frequency Span:	0-100 MHz
Reference Level:	+6 dBm
Resolution Bandwidth:	3 KHz
Scale:	Log, 10 dB/div
Output frequency:	67.89 MHz
Clock frequency:	300.0 MHz

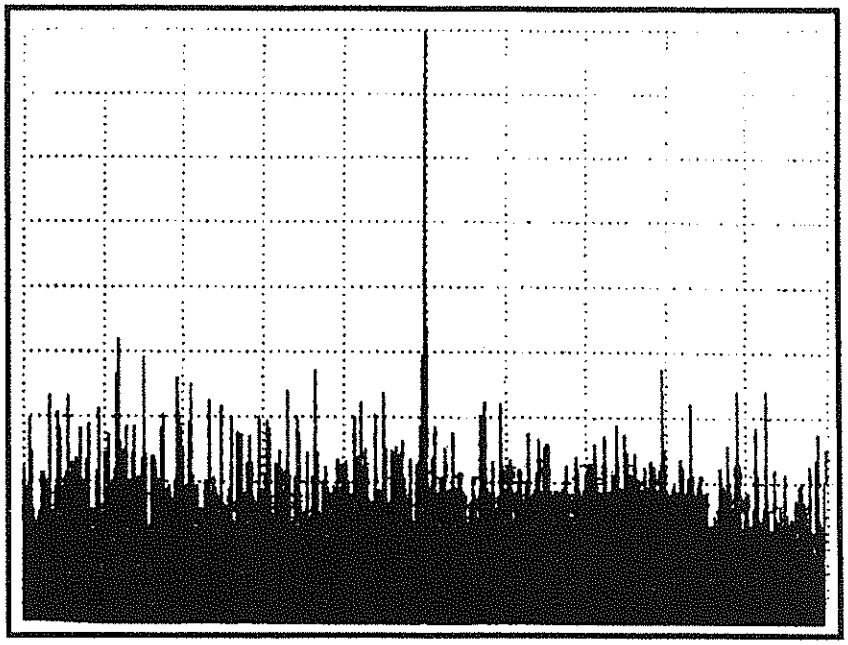


FIG. 20

```
/*  SAMPLING GENERATOR CONTROL*/
/* Program Name: dig.c */
#include <stdio.h>
#include "nivxi.h"
#include "spet.h"
#include <math.h>

void contdds (port,value)
uint8 port;
uint8 value;

      {
        int16 ret;

                    subcio_out(value, cio_samdds_data, port, cio_samdds_add);
                    ret=subcio_in(cio_samdds_read, port, cio_samdds_add);
                    if(ret != value)
                    {
                    printf("Be careful, error in write or read!!\n");
                    printf("Check  the LOC/REM Switch!!\n");
                    }

      }


 main()

{
int response, i;
uint8 result;
int16 ret;
uint8 dataw;
float res;
int m;
FILE *inpu;
uint8 dataw1, dataw2, dataw3, dataw4;
float fdummy, freq ;
int ddummy;
uint32 dummy;
inpu=fopen("/home/stelio/cprog/ad.dat","w+");
resetvmio12();


continue_prog: result=InitVXIlibrary();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/
enable('2');
enable('3');


initvmio12('0','b','o','n'); /*port b of IC 6 becomes out (Clock Read)*/
initvmio12('1','a','i','n'); /*port b of IC 7 becomes in (data+flags)*/
initvmio12('1','b','i','n'); /*port b of IC 7 becomes in (flags)*/
enable('0');
enable('1');


/* Reset  _-_*/
dataw=0;
subcio_out (dataw,cio_samdds_data,samdds_cont,cio_samdds_add);
dataw=2;
subcio_out (dataw,cio_samdds_data,samdds_cont,cio_samdds_add);
dataw=0;
subcio_out (dataw,cio_samdds_data,samdds_cont,cio_samdds_add);



/* Status Control  */
 /* ret=subcio_in (cio_samdds_read, read_samddsf1, cio_dig_add);

    if((ret & 0x1) == 0 )
```

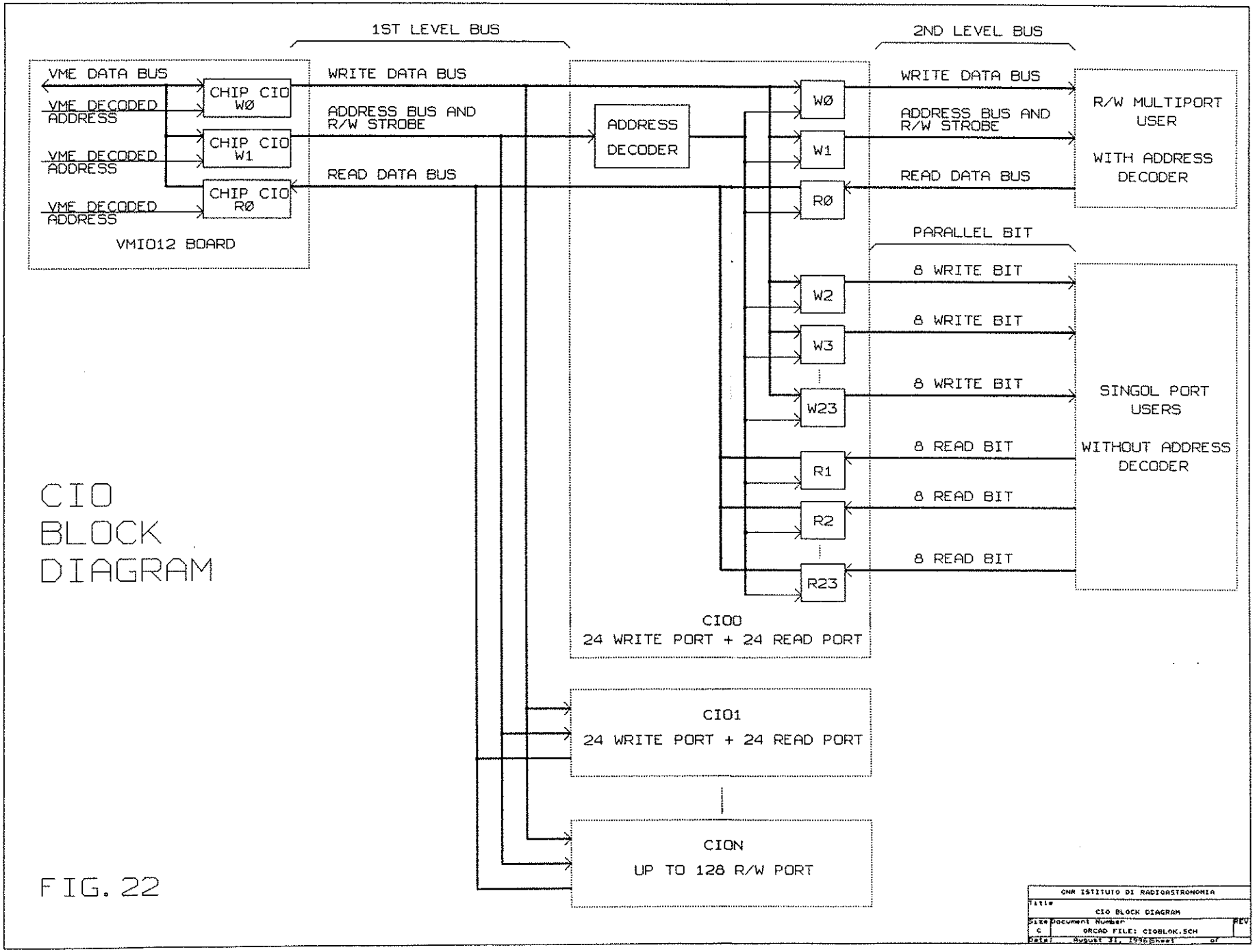
```
     printf("DDS SYSTEM IS IN LOCAL STATUS!! \n");

    if((ret & 0x2) == 1 )
      printf("5 MHz  failure \n");*/

/*=========================================================================*/
/*SETTING OF THE SAMPLING FREQUENCY */

printf("SAMPLING REQUENCY (Freq.) ? (0.1-->110.000 MHz (XXX.xxxxxx MHz)  \n");
scanf("%f", &freq);
dummy=freq*1000000/1.117587;

dataw4=dummy/16777216;
dataw3=(dummy-dataw4*16777216)/65536;
dataw2=(dummy-dataw4*16777216-dataw3*65536)/256;
dataw1=(dummy-dataw4*16777216-dataw3*65536-dataw2*256);


contdds (samdds1,dataw1);
contdds (samdds2,dataw2);
contdds (samdds3,dataw3);
contdds (samdds4,dataw4);


/* FREQ. LOAD --_-*/
dummy=0;
subcio_out(dummy, cio_samdds_data, samdds_cont, cio_samdds_add);
dummy=1;
subcio_out(dummy, cio_samdds_data, samdds_cont, cio_samdds_add);
dummy=0;
subcio_out(dummy, cio_samdds_data, samdds_cont, cio_samdds_add);


result=CloseVXIlibrary();

}
```

- ***DDS 3.***

Il generatore DDS 3 (8) non e` ancora stato realizzato. Questo sara`, in pratica, una replica del DDS 1 e sara` progettato utilizzando una evaluation board della Stanford tipo STEL-1376. Il compito principale di questo sintetizzatore sara` quello di generare segnali monocromatici in grado di spostarsi in frequenza con una legge definibile e programmabile per il test del software e/o hardware di compensazione del Doppler. Come nel caso dei DDS descritti in precedenza, anche il DDS 3 usera` un PLL per la generazione del clock interno che a sua volta sara` sincronizzato dal riferimento H-Maser di stazione. Anche la gestione di questo DDS sara`affidata alla Sparc che controlla il crate VME attraverso il solito sistema CIO I/O expander. Lo schema a blocchi del DDS 3 sara` simile a quello del DDS 1.

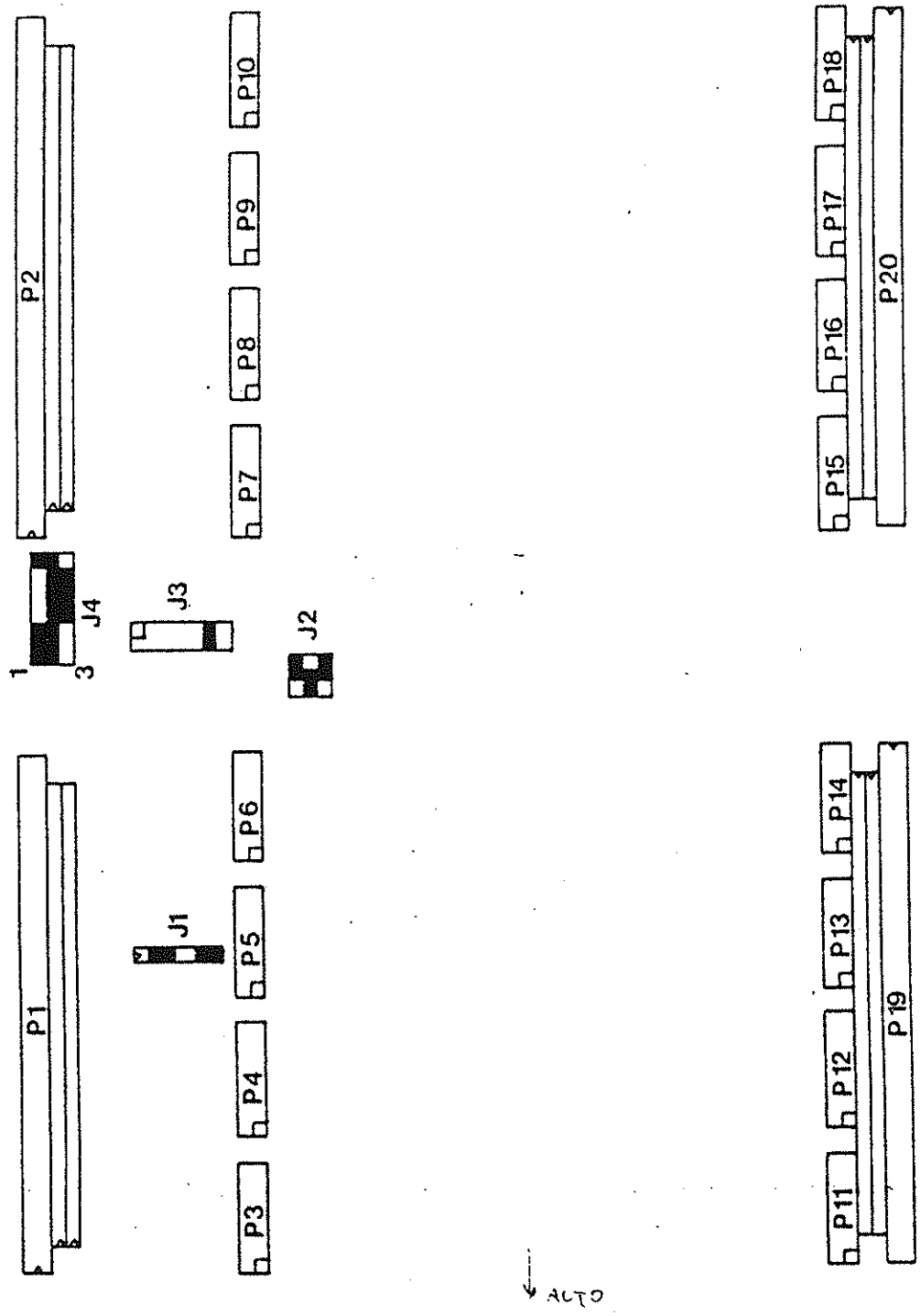
- ***Scheda Parallela I/O (VME)*** [5] **e *CIO*** [2].

Questa blocco (12) costituisce il ponte tra il crate VME-Sparc e lo spettrometro attraverso il CIO I/O expander che verra` descritto nel prossimo paragrafo. La scheda impiegata e` un modello PCB VMIO12 della OR tedesca e usa chip CIO molto versatili e programmabili del tipo Z8536. Su questa e`possibile montare fino a 8 moduletti tipo Piggy-Back che possono essere scelti all'interno di una vasta gamma di funzioni. Ha a bordo 12 contatori a 16 bit (all'interno degli Z8536) programmabili che comunque in questa applicazione non vengono utilizzati. Il tempo di accesso alla scheda e` di circa 375 nS e le dimensioni sono standard VME 6U. In Fig. 21a, b, c vengono riportati le disposizioni dei Jumper e Piggy-Back, disposizione dei CIO chips ed infine, dei Piggy-Back usati e relativi indirizzi. Lo schema a blocchi dell' intero sistema di interfacciamento viene riportato in Fig. 22. Come si puo` notare sulla scheda PCB VMIO12 sono necessarie solo 3 porte per la gestione del CIO I/O expander, 2 di scrittura e 1 di lettura.
Esse costituiscono il bus di I/O di primo livello composto da un bus dati in scrittura a 8 bit, un bus dati in lettura a 8 bit ed un bus di controllo composto da 7 bit di indirizzamento ed 1 di selezione operazione di scrittura e lettura. Questa architettura permette di gestire verso l'esterno fino a 128 porte di scrittura e 128 di lettura, ciascuna ad 8 bit alle quali possono essere collegati direttamente i vari utilizzatori. Ad esempio l'orologio è interfacciato direttamente al CIO tramite 6 porte di lettura. Altri dispositivi quali ad esempio il sistema di controllo dello stadio di ingresso a radiofrequenza, i phase shifter etc.., dato il consistente numero di porte richiesto, sono collegate al CIO tramite un bus di secondo livello che ricalca la struttura del bus di primo livello fra VMIO12 e CIO, utilizzando 2 porte di uscita ed una di ingresso del CIO. Come anticipato prima la struttura CIO permette una espansione fino a 128 porte di ingresso e 128 di uscita, in realtà a causa delle dimensioni fisiche dei connettori sono state costruite due unità CIO, ciascuna delle quali fornisce 24 porte di ingresso e 24 di uscita, in caso di necessità sullo stesso bus di primo livello possono venire collegate altre unità CIO fino a ricoprire le 128 porte disponibili. Viene riportato il sorgente del modulo software che permette al calcolatore di gestione di accedere alla scheda parallela VME, passando attraverso il bus NI-VXI (file: vmio12.c). Viene riportato anche il file <include> di nome "spet.h" che contiene tutti gli indirizzi sia del CIO1 che del CIO2 che viene usato in tutti i moduli di controllo. Viene inoltre riportato il modulo per il controllo del CIO (file: ciotest.c).



CIO
BLOCK
DIAGRAM

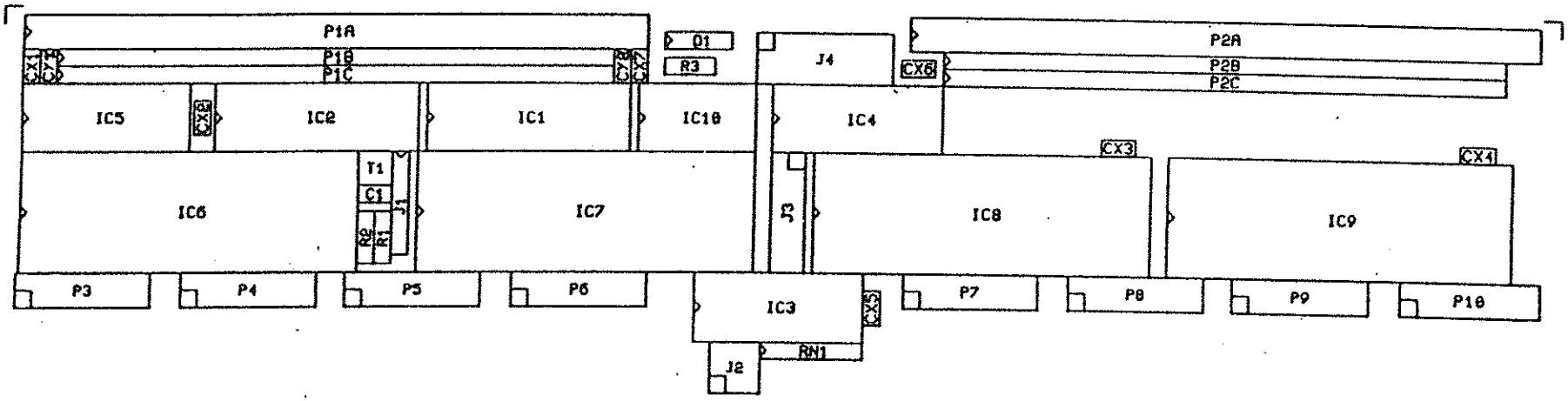
FIG. 22



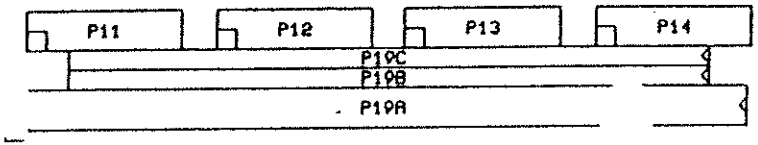
P1 = VMEbus connector
P2 = no function
P3-10 = Z-8536 I/O lines
P11-18 = connection to P19, P20
P19,20 = I/O connector

J1 = AM2 select
J2 = INTACK level
J3 = IRQ level
J4 = address base A15-A08

FIG. 21a



LATO ALTO



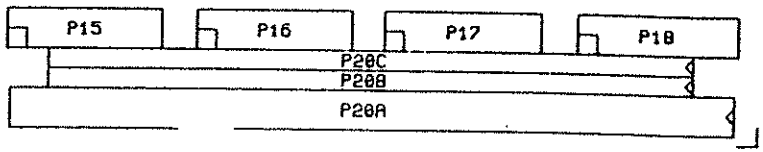


Fig. 21b

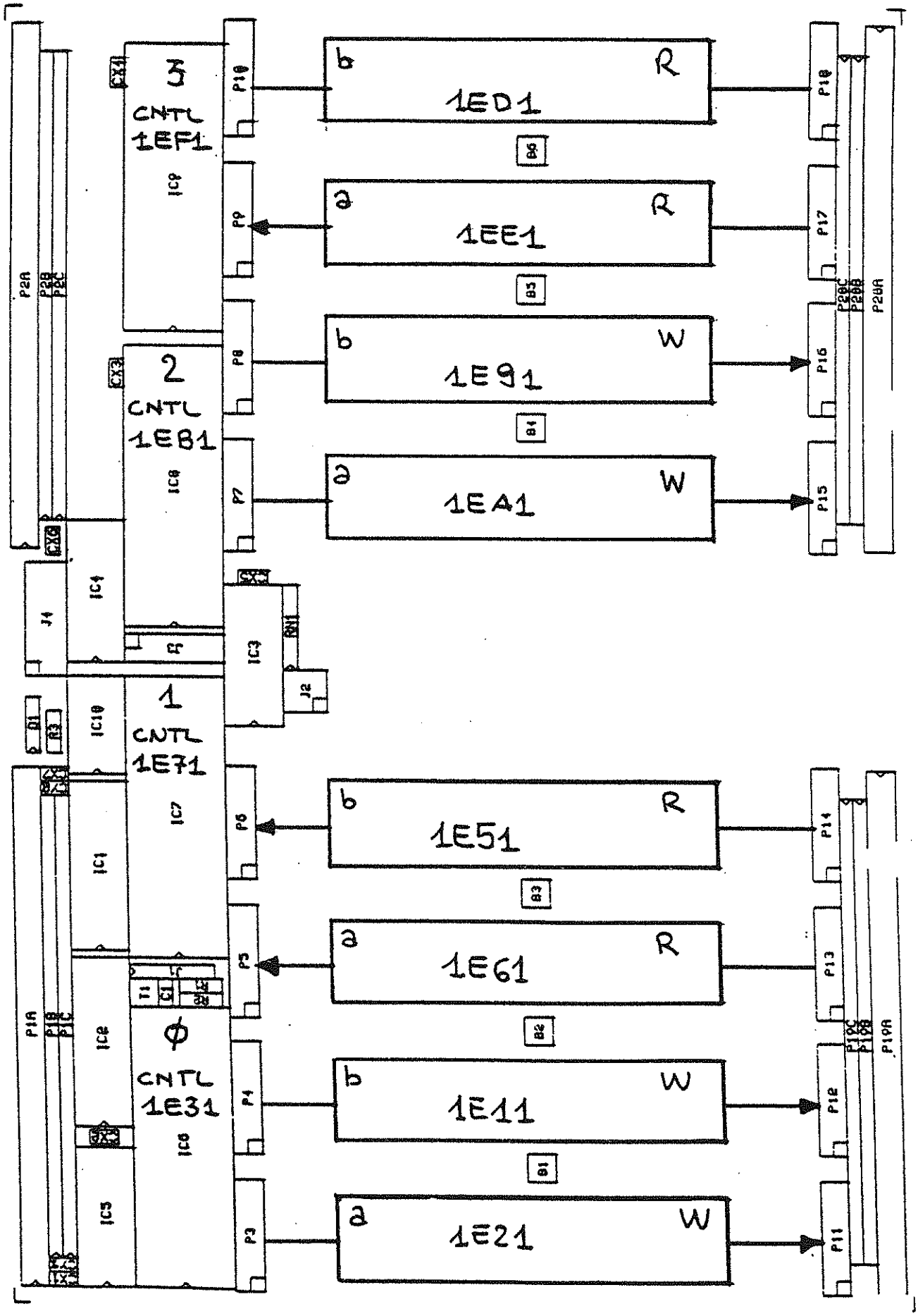

3
CNTL
1EF1
2
CNTL
1EB1
1
CNTL
1E71
Φ
CNTL
1E31
b
1ED1
R
a
1EE1
R
b
1E91
W
a
1EA1
W
b
1E51
R
a
1E61
R
b
1E11
W
a
1E21
W

FIG. 21C

```
  /*  Program Name: vmio12.c  */
  /*  December 27 / 1993  */


#include  <stdio.h>
#include  "nivxi.h"


/*----------------------ResetCIO*/
void resetvmio(chip)
short chip;

{
int16  retvalue;
uint32 cntl;
uint16 accessparms;
uint16 width;
uint32 value;

accessparms = 1;
width = 1;


{
        switch (chip)
        {
        case '0': cntl= 0x1e31;
                   break;

        case '1': cntl=0x1e71;
                   break;

        case '2': cntl= 0x1eB1;
                   break;

        case '3': cntl=0x1eF1;
         }

  value=0;
  retvalue=VXIout (accessparms,cntl,width,value);
  value=1;
  retvalue=VXIout (accessparms,cntl,width,value);
  value=0;
  retvalue=VXIout (accessparms,cntl,width,value);
 }

}

/*-------------------------------------------------*/
void resetvmio12()
{
        resetvmio('0');
        resetvmio('1');
        resetvmio('2');
        resetvmio('3');

}
/* ----------------------------------------enable*/
void enable(chip)   /* output/input enable for IC6-9*/
short chip;
{
        int16 ret;
        uint32 cntl;
        uint16 accessparms;
        uint16 width;
        uint32 value;

accessparms = 1;
width = 1;


{
```

```
        switch (chip)
        {
        case '0': cntl= 0x1e31;
                  break;

        case '1': cntl=0x1e71;
                  break;

        case '2': cntl= 0x1eB1;
                  break;

        case '3': cntl=0x1eF1;
         }

  value=0x01;
  ret=VXIout (accessparms,cntl,width,value);
  value=0x94;
  ret=VXIout (accessparms,cntl,width,value);
}

}



/*---------------------------- Init of chips 8536 ------------ */
void initvmio12 (chip,channel,dir,inv)
        short chip;
        char dir;
        char channel;
        char inv;
{
        int16 retval;
        uint32 address;
        uint32 cntl;
        uint16 accessparms;
        uint16 width;
        uint32 value1;
        uint32 value2;
        uint32 value3;
        uint32 value;

accessparms = 1;
width = 1;



 {
         switch (chip)
         {
         case '0': cntl= 0x1e31;
                   break;

         case '1': cntl= 0x1e71;
                   break;

         case '2': cntl= 0x1eB1;
                   break;

         case '3': cntl= 0x1eF1;
         }

         switch (channel)
         {
         case 'a': value1= 0x22;
                   value2= 0x23;
                   break;

         case 'b': value1= 0x2a;
                   value2= 0x2b;
         }
         switch (dir)
         {
         case 'o': value3=0;
                   break;
```

```
        case 'i': value3=0xff;
                  if (channel=='c')
                     value3=0x0f;

        }
        switch (inv)
        {
        case 'y': value=0xff;
                  break;

        case 'n': value=0;
                  if (channel=='c')
                     value3=0x0f;

        }


  retval=VXIout (accessparms,cntl,width,value1);
  retval=VXIout (accessparms,cntl,width,value);
  retval=VXIout (accessparms,cntl,width,value2);
  retval=VXIout (accessparms,cntl,width,value3);

 }
}
/*============================================================*/
/*============================out data ==================*/
void outvmio12 (chip,channel,value)
        char chip;
        char channel;
        uint32 value;
{
        int16 retval;
        uint32 address;
        uint16 accessparms;
        uint16 width;

accessparms = 1;
width = 1;


{
        switch (chip)
        {
        case '0': switch (channel)
                  {
                   case 'a':address=0x1e21;
                            break;
                   case 'b':address=0x1e11;
                            break;
                   case 'c':address=0x1e01;
                   }
                  break;

        case '1': switch (channel)
                  {
                   case 'a':address=0x1e61;
                            break;
                   case 'b':address=0x1e51;
                            break;
                   case 'c':address=0x1e41;
                   }
                  break;

        case '2':switch (channel)
                  {
                   case 'a':address=0x1ea1;
                            break;
                   case 'b':address=0x1e91;
                            break;
                   case 'c':address=0x1e81;
                   }
                  break;
        case '3':switch (channel)
                  {
```

```
                    case 'a':address=0x1ee1;
                             break;
                    case 'b':address=0x1ed1;
                             break;
                    case 'c':address=0x1ec1;
                    }
          }

  retval=VXIout (accessparms,address,width,value);
  }
 }

/*=================================================================*/
/*============================ in data ============================*/
uint8 invmio12 (chip,channel)
        char chip;
        char channel;

{
        int16 retval;
        uint16 *punta;
        uint32 address;
        uint16 accessparms;
        uint16 width;
        uint16 val;
        uint8 value;


accessparms = 1;
width = 1;
{
        switch (chip)
        {
        case '0': switch (channel)
                  {
                   case 'a':address=0x1e21;
                            break;
                   case 'b':address=0x1e11;
                            break;
                   case 'c':address=0x1e01;
                   }
                  break;

        case '1': switch (channel)
                  {
                   case 'a':address=0x1e61;
                            break;
                   case 'b':address=0x1e51;
                            break;
                   case 'c':address=0x1e41;
                   }
                  break;

         case '2':switch (channel)
                  {
                   case 'a':address=0x1ea1;
                            break;
                   case 'b':address=0x1e91;
                            break;
                   case 'c':address=0x1e81;
                   }
                  break;
         case '3':switch (channel)
                  {
                   case 'a':address=0x1ee1;
                            break;
                   case 'b':address=0x1ed1;
                            break;
                   case 'c':address=0x1ec1;
                   }
         }
    }
    retval=VXIin (accessparms,address,width,&value);
    return (value);
```

```
}


/*-------------- CIO out Function --------------*/

void cio_out(add,dwr)
 uint8 add;
 uint8 dwr;


{
outvmio12('2','a',dwr);

add=(add | 0x80);
outvmio12('2','b',add);
add=(add & 0x7f);
outvmio12('2','b',add);
add=(add | 0x80);
outvmio12('2','b',add);

}



/*-------------- CIO in Function --------------*/

uint8 cio_in (add)
 uint8 add;

{
int16  result;

/*add+128 */
add=(add | 0x80);
outvmio12('2','b',add);

result=invmio12('3','a');

return(result);
}



/* ----------------- subcio_out --------------------*/

void subcio_out (dataw,adw,adata,addcio)

uint8 dataw;
uint8 adw;
uint8 adata;
uint8 addcio;

 /*dataw --> Setisys data value                      */
 /*adw   --> Cio port for Setisys data               */
 /*adata --> Setisys port address                    */
 /*addcio--> Cio port for Setisys address            */


{
  cio_out (adw,dataw);

  adata=(adata | 0x80);
  cio_out (addcio,adata);

  adata= (adata & 0x7f);
  cio_out (addcio,adata);

  adata=(adata | 0x80);
  cio_out (addcio,adata);
}

/* ----------------- subcio_in ----------------------*/
 subcio_in (adr,adata,addcio)

uint8 adr;
```

```
uint8 adata;
uint8 addcio;

 /*adr    --> Cio port for Setisys data read                    */
 /*adata --> Setisys port address                               */
 /*addcio--> Cio port for Setisys address                       */

{

int16 result;
   adata=(adata | 0x80);
   cio_out (addcio,adata);

   result=cio_in (adr);

 return(result);

}
```

```
/*    Include file for Spectrometer I/O ports Blocks          */

/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*                  CIO N.1 :  PORTS 0 --> 23                  */
/*||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/


/* PHASE SHIFTER block */
/*----CIO R/W PORTS NUMBERS----*/
#define cio_pshift_read 0
#define cio_pshift_data 0
#define cio_pshift_add 1
/*=====================WRITE====================*/
#define w_pshift_1 0 /* phase shifter n. 1  */
#define w_pshift_2 1 /* phase shifter n. 2  */
#define w_pshift_3 2 /* phase shifter n. 3  */
#define w_pshift_4 3 /* phase shifter n. 4  */
#define w_pshift_5 4 /* phase shifter n. 5  */
#define w_pshift_6 5 /* phase shifter n. 6  */
#define w_pshift_7 6 /* phase shifter n. 7  */
#define w_pshift_8 7 /* phase shifter n. 8  */
/*=====================READ=====================*/
#define r_pshift_1 0 /* phase shifter n. 1  */
#define r_pshift_2 1 /* phase shifter n. 2  */
#define r_pshift_3 2 /* phase shifter n. 3  */
#define r_pshift_4 3 /* phase shifter n. 4  */
#define r_pshift_5 4 /* phase shifter n. 5  */
#define r_pshift_6 5 /* phase shifter n. 6  */
#define r_pshift_7 6 /* phase shifter n. 7  */
#define r_pshift_8 7 /* phase shifter n. 8  */


/* Input RADIO FREQUENCY block */
/*----CIO R/W PORTS NUMBERS----*/
#define cio_rf_read 1
#define cio_rf_data 2
#define cio_rf_add 3
/*=====================WRITE====================*/
#define w_e/w_att 0  /* E/W input attenuators    */
#define w_n/s_att 1  /* N/S input attenuators    */
#define w_glob_att 2 /* GLobal attenuator        */
#define w_sing_sw 3  /* Single switch            */
/*=====================READ=====================*/
#define r_e/w_att 0  /* E/W input atteniators    */
#define r_n/s_att 1  /* N/S input attenuators    */
#define r_glob_att 2 /* Global attenuators       */
#define r_sing_sw 3  /* Single switch            */


/* Digital Direct Sinthetyzer (DDS) */
/*----CIO R/W PORTS NUMBERS----*/
#define cio_dds_read 2
#define cio_dds_data 4
#define cio_dds_add 5
/*=========================WRITE====================*/
#define w_dds_1 0 /* Hz/100 and units of Hz (2 nibbles)*/
#define w_dds_2 1 /* 10xHz and 100xHz (2 nibbles)      */
#define w_dds_3 2 /* kHz and 10xKHz  (2 nibbles)       */
#define w_dds_4 3 /* 100xKHz and MHz (2 nibbles)       */
#define w_dds_5 4 /* 10xMhz (1 nibble) */
#define w_dds_reset 5 /* DDS remote Reset */
/*=========================READ=====================*/
#define r_dds_1 0 /* Hz/100 and units of Hz (2 nibbles)*/
#define r_dds_2 1 /* 10xHz and 100xHz (2 nibbles)      */
#define r_dds_3 2 /* kHz and 10xKHz  (2 nibbles)       */
#define r_dds_4 3 /* 100xKHz and MHz (2 nibbles)       */
#define r_dds_5 4 /* 10xMhz (1 nibble) */
#define r_dds_status 5 /* bit 0-->remote   bit 1-->5MHz fail */


/* Local Control Data Acquisition Dystem  (DAS)  */
/*----CIO R/W PORTS NUMBERS----*/
#define cio_ad_read 3
#define cio_ad_data 6
#define cio_ad_add 7
```

```
/*=========================WRITE=========================*/
#define w_ad_stconv 0  /* bit 0  active High _-_ */
#define w_ad_mux 1 /* bit 0-->5 mux , bit 6 -->gain control*/
/*=========================READ==========================*/
#define r_ad_lsb 0 /* A/D converter LSB */
#define r_ad_msb   /* A/D converter MSB */
#define r_ad_busy  /* end of conversion */


/*|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/
/*                  CIO N.2 :  PORTS 24 --> 47                 */
/*|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||*/

/ Programmable High Speed Digitizer  */
/*----CIO R/W PORTS NUMBERS----*/
#define cio_dig_read 24
#define cio_dig_data 25
#define cio_dig_add  26

/* Reference frequency generator */
/*=========================WRITE=========================*/
#define w_refgen_lsb 0     /* units and tens  (2 nibbles)        */
#define w_refgen_msb 1     /* hundreds and thousands (2 nibbles) */
#define w_refgen_reset 2   /* Reset  (bit 0)                      */
/*=========================READ==========================*/
#define r_refgen_lsb 0     /* units and tens  (2 nibbles)        */
#define r_refgen_msb 1     /* hundreds and thousands (2 nibbles) */
#define r_refgen_status 2 /* bit 0 /rem,  bit 1 /5MHz fail       */

/*  Phase Locked Loop (PLL) -Sampling Generator- */
/*=========================WRITE=========================*/
#define w_pll_lsb 8         /* units and tens (2 nibbles)        */
#define w_pll_msb1 9        /* hundreds and thousands (2 nibbles) */
#define w_pll_msb2 10       /* tens of thousands (1 nibble)      */
/*=========================READ==========================*/
#define r_pll_lsb 8         /* units and tens (2 nibbles)        */
#define r_pll_msb1 9        /* hundreds and thousands (2 nibbles  */
#define r_pll_msb2 10       /* tens of thousands (1 nibble)      */
#define r_pll_status 11     /* Status bit 0 /rem,  bit 1 /lock   */

/*Programmable Divider  */
/*=========================WRITE=========================*/
#define w_pdiv_lsb 16         /* units and tens (2 nibbles)        */
#define w_pdiv_msb1 17        /* hundreds and thousands (2 nibbles) */
#define w_pdiv_msb2 18        /* tens of th. and units of milion (2 nibbles) */
#define w_pdiv_msb3 19        /* tens of milion (1 nibble)         */
#define w_pdiv_ls 20          /* Remote Load/Start                 */
/*=========================READ==========================*/
#define r_pdiv_lsb 16         /* units and tens (2 nibbles)        */
#define r_pdiv_msb1 17        /* hundreds and thousands (2 nibbles) */
#define r_pdiv_msb2 18        /* tens of th. and units of milion (2 nibbles) */
#define r_pdiv_msb3 19        /* tens of milion (1 nibble)         */
#define r_pdiv_status 20      /* Status  bit 0 /rem                */

/* UT System Clock */
/*----CIO R/W PORTS NUMBERS----*/
#define cio_ucl_read1 25 /* sec./100 and sec/10  (2 nibbles)      */
#define cio_ucl_read2 26 /* seconds and tens of seconds           */
#define cio_ucl_read3 27 /* minutes and tens of minutes           */
#define cio_ucl_read4 28 /* hours and tens of hours               */


/* SIDERAL System Clock */
/*----CIO R/W PORTS NUMBERS----*/
#define cio_sid_read1 29 /* sec./100 and sec/10  (2 nibbles)      */
#define cio_sid_read2 30 /* seconds and tens of seconds           */
#define cio_sid_read3 31 /* minutes and tens of minutes           */
#define cio_sid_read4 32 /* hours and tens of hours               */
```

```
/*  READ / WRITE CONTROL PROGRAM*/
/* Program Name: ciotest.c */
#include <stdio.h>
#include "nivxi.h"
#include "spet.h"


 main()

{
uint16 result;
uint16 ret;
int cio_write, ind;
int cio_read;
int user_add;
int user_data, data_read;
char res;


result=InitVXIlibrary();
resetvmio12();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/
enable('2');
enable('3');



cont: printf("CIO WRITE PORT  \n");
ret=scanf("%d",&cio_write);


printf("CIO READ PORT  \n");
ret=scanf("%d",&cio_read);


printf("Manual or Automatic (1->255) ?  (m/a) \n");
scanf("%s", &res);


switch (res)
        {
        case 'm':cont1:printf("DATA to write in the USER PORT  \n");
                 scanf("%u", &user_data);
                 cio_out(cio_write, user_data);
                 ret= cio_in(cio_read);
                 if(user_data != ret)
                 printf("BE CAREFULL Error in writing or reading data \n");
                 printf("write / read value %u -> % u \n", user_data, ret);
                 goto cont1;
                 break;

         case'a':for (ind=0; ind <=255; ++ind)
                {
                 cio_out(cio_write, ind);
                 ret= cio_in (cio_read);
                 if(ind != ret)
                 printf("BE CAREFULL Error in writing or reading data \n");
                 printf("write / read %u -> % u \n", ind, ret);
                 }
         }
goto cont;
result=CloseVXIlibrary();


}
```

- ***Sistema di Controllo.***

Questo blocco (16) (Fig.10) ha il compito fondamentale di controllare i punti piu` importanti del sistema. La misura del livello nei punti piu` cruciali puo` essere fatta sia per controllare che non vi siano malfunzionamenti sia per sincerarsi che il sistema sta operando nelle corrette condizioni. Lo schema a blocchi semplificato appare in Fig. 23, da qui si puo` vedere che sono presenti 3 blocchi con 8 ingressi ciascuno in a.c. Il segnale in ingresso in questi moduli (banco N.1, N.2 e N.3) viene rivelato, amplificato, integrato ed infine mandato ad un multiplexer analogico a 32 canali che seleziona, su controllo del gestore di sistema, quale canale mandare al convertitore A/D da 16 bit. Il banco N.4 accetta in ingresso segnali in d.c. con range 0 / 5 Volt. Il multiplexer e` formato da switches analogici tipo DG507ACJ caratterizzati da una bassa resistenza di ON. Il chip usato come convertitore e` un 16 bit della Burr-Brown modello ADC71JG dotato di buona linearita` e precisione. Per facilita` di controllo della scheda convertitore sono stati messi sul pannello frontale dei diodi led corrispondenti allo start conversion, indirizzamento ed ai vari bit del convertitore. Il sistema e` montato all' interno di un cestello da 6U in cui trovano alloggio le schede che accettano segnali da rivelare (a.c.), quelle in d.c, l'interfaccia locale, l'alimentatore e la scheda del convertitore stesso. Al momento il software di questo blocco non e` ancora stato scritto.

C:\ABC\AD.AF3

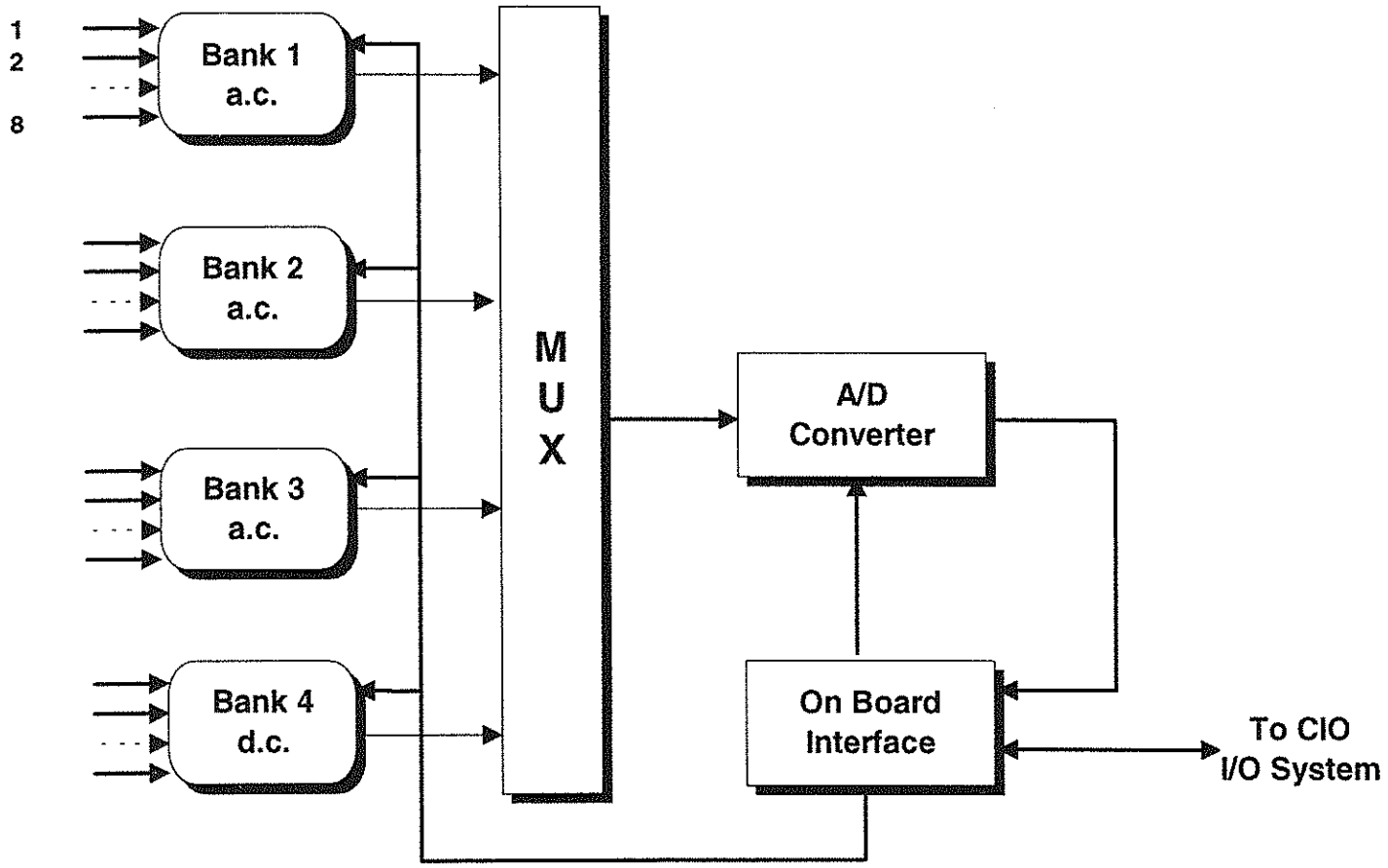


FIG. 23

- ***Sistema "Controllo Interferenze"*** [3].

Il blocco (17) per il controllo delle radio interferenze e` stato espressamente progettato per tenere continuamente monitorata la banda di ricezione del radiotelescopio senza minimamente interagire con il normale funzionamento dello spettrometro. In Fig.23a e` visibile lo schema a blocchi della acquisizione dati programmabile. Il blocco 1 fornisce, una volta programmato, la frequenza di riferimento per il PLL (blocco 2) che genera la frequenza di sampling ad un valore predefinibile all' interno del range 6/13 MHz (Fig.23b). Il blocco 3 che segue, ha il compito, una volta programmato, di mandare al convertitore A/D veloce gli impulsi di start conversion. In altri termini, se si vuole scomporre la banda di ingresso in N canali, si programma il divisore per fornire, alla frequenza richiesta da Nyquist, 2N punti. Il blocco convertitore usato si basa sulla evaluation board della TRW tipo TDC1029C che monta un flash A/D converter TDC1029J7C. Questo chip puo` operare fino a frequenze di sampling di 100 MHz massimi con 6 bit di risoluzione ed un range di ingresso di 1 Vpp max.
I dati in uscita dal convertitore vengono mandati alla Sparc attraverso una FIFO 64K x 9 bit ad alta velocita` che, attraverso una interfaccia per Sbus installata direttamente all'interno della Sparc di gestione, permette l'acquisizione in snap-shot mode fino a 64536 punti (32768 canali). La Sparc in questione e` una Axil con 85 MHz di clock e costituisce la consolle di comando della Sparc FORCE-3 che controlla lo spettrometro dallo slot 0 del VME e nello stesso tempo, su di un'altra finestra, controlla la situazione interferenze come detto sopra. Sono stati scritti tutta una serie di blocchi software che istruiscono la "macchina" sui parametri da impostare. Piu` precisamente, e` possibile localmente o attraverso il custom local bus impostare:

- sul blocco 1 lo step con cui "sintonizzare" la sample rate.
- sul blocco 2 la frequenza del PLL adatta alla banda da tenere sotto controllo.
- sul blocco 3 il numero di impulsi da inviare all'A/D conveter.

In questo modo all' uscita del sistema si rendono disponibili un numero prefissato di dati su cui, una volta acquisiti, si calcola con la Sparc Axil la trasformata di Fourier, il modulo quadro per ogni canale ed infine la media di un certo numero di spettri. Alla fine con una serie di macro scritte in ambiente DADiSP si importano i dati automaticamente per la visualizzazione. Il ciclo appena descritto vienne attivato una volta al minuto.

Nell' appendice verranno riportati i vari blocchi software (in linguaggio C) scritti espressamente per la gestione del sistema. Il sistema digitizzatore/acquisizione/calcolo della FFT e` stato testato collegando il sistema alla parabola sintonizzata sulla banda dei 22 GHz, osservando con 2048 canali un maser molto forte (W51) (Fig.24a). Un' altra prova di funzionamento qualitativo e` stata effettuata collegando lo stesso sistema all' uscita delle medie frequenze larghe (6 MHz) della Croce e producendo uno spettro con 2048 canali della banda di ricezione. Come si puo` notare, le interferenze dovute a ponti militari di Monte Grande 1, 2 e 3 sono facilmente riscontrabili.

Lo shema a blocchi del modulo software per la gestione del controllo delle interferenze viene riportato in Fig. 24c. Nella versione migliorata il sistema verra` gestito dalla Sparc che operera` come consolle di sistema (AXIL 85 MHz) e sulla quale e` installato DADiSP. Dalla directory /data/dsp si lancia:

Xdadisp -c=rfi

Da DADiSP viene lanciato il modulo "rfi" che a sua volta lancia il programma "interf.c" che ha il compito di impostare il numero richiesto di punti da acquisire, di calcolare la FFT, il modulo quadro e la media del numero di spettri impostato. Come si puo` notare questo modulo fa largo uso di macro scritte in ambiente DADiSP.

100 Ms DIGITIZER:BLOCK DIAGRAM

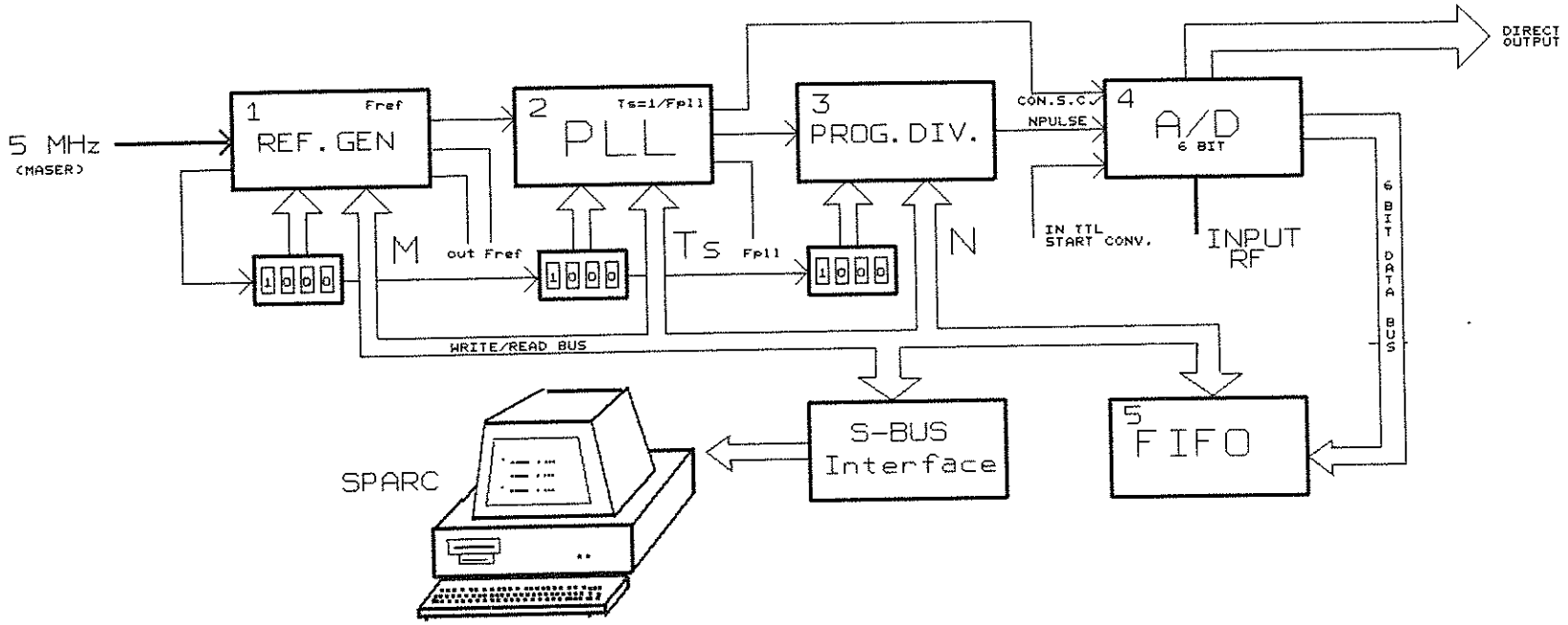
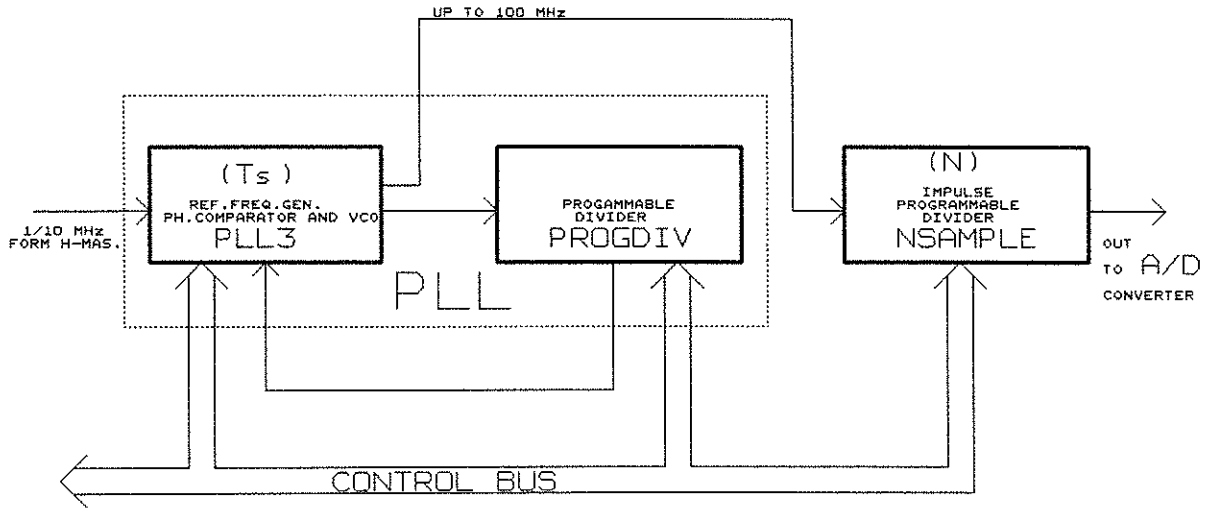


FIG.23a

-CNR- ISTITUTO DI RADIOASTRONOMIA		
Title: 70 Ms DIGITIZER:BLOCK DIAGRAM		
Size: B	Document Number: FILE:DIGO.SCH	REV
Date: December 13, 1995	Sheet of	

TRIGGER GENERATOR

PULSE BURST SECTION



RES=1/Ts*N

FIG. 23b

-CNR- ISTITUTO DI RADIOASTRONOMIA		
Title: PRO.TRIG.GENERATO:BLOCK DIAG.		
Size: B	Document Number: TITLE:BLOCK.SCH	REV
Date: September 7, 1992	Sheet of	

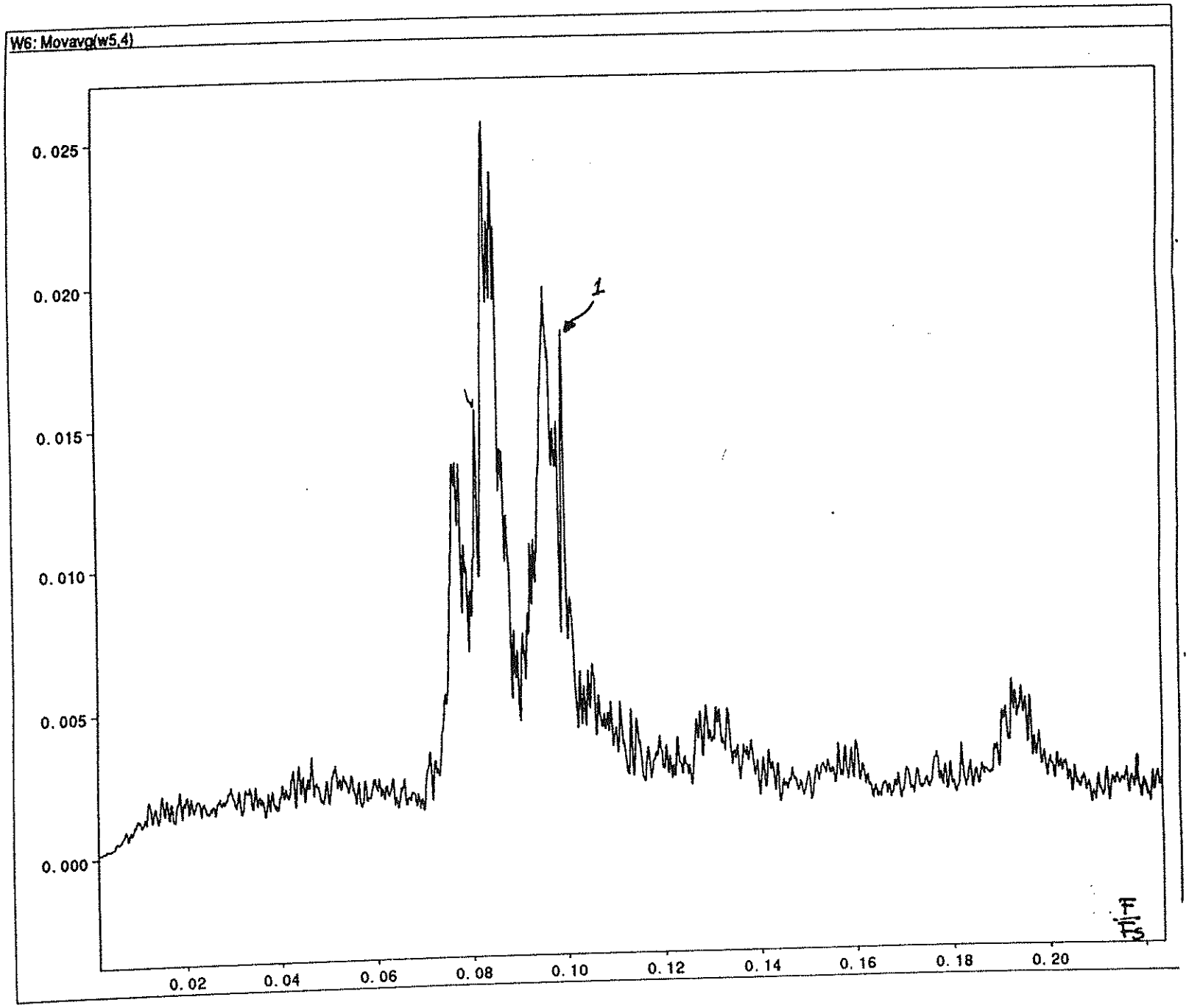

W6: Movavg(w5,4)
0.025
0.020
0.015
0.010
0.005
0.000
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
1

FIG 24-2

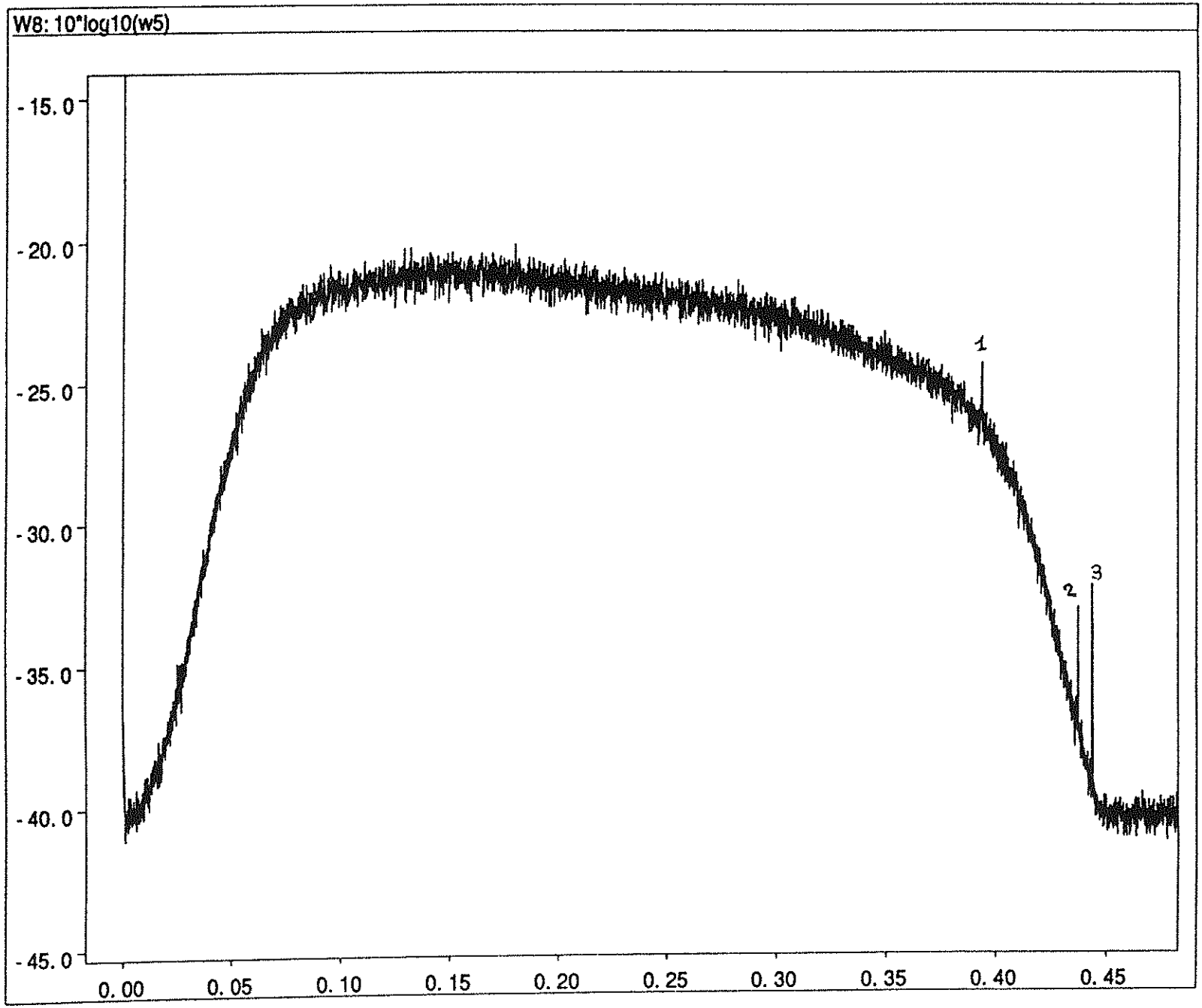

W8: 10*log10(w5)
-15.0
-20.0
-25.0
-30.0
-35.0
-40.0
-45.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
1
2
3

FIG.24b

C:\ABC\RFI.AF3

Schema a blocchi del "Controllo Interferenze"

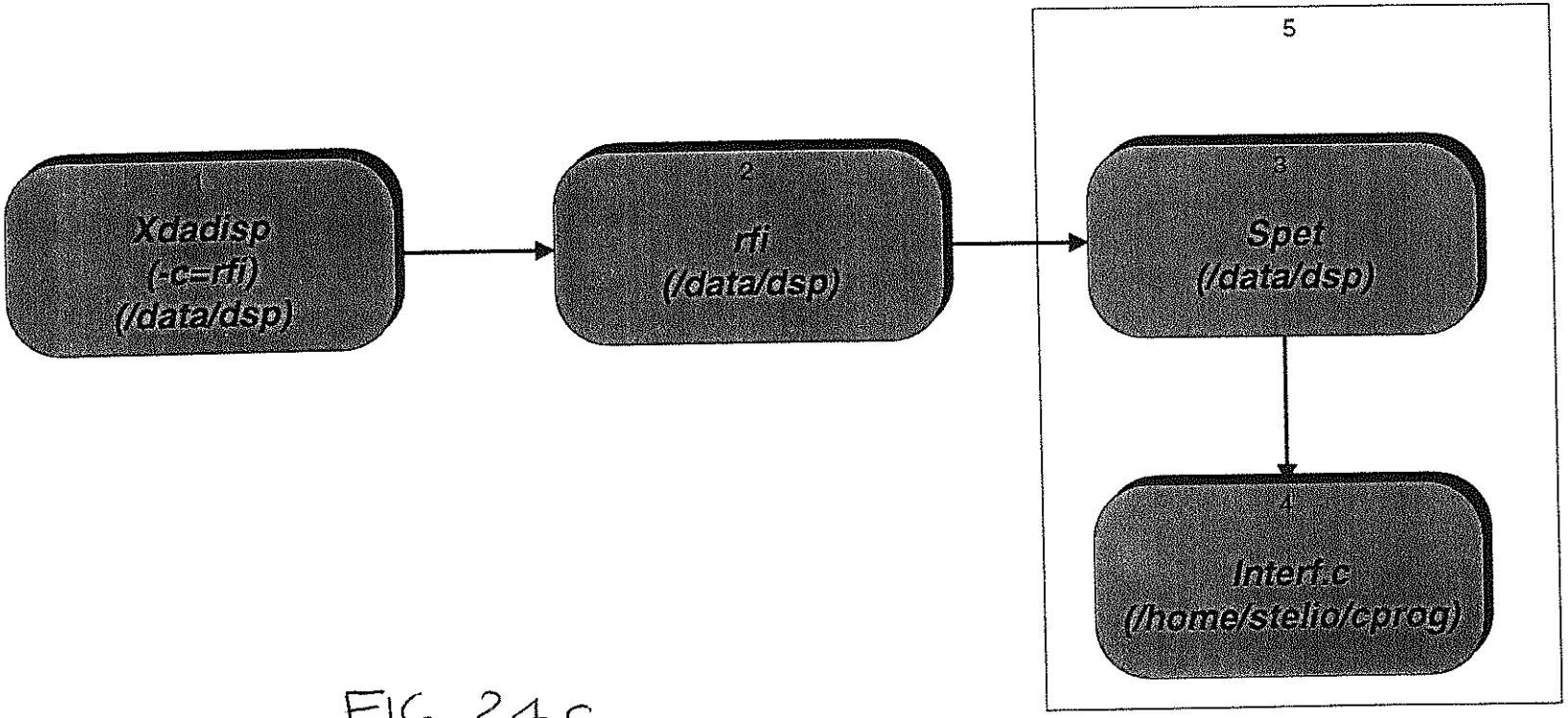


FIG. 24c

```
o interf@CR
W
E
L interf@CR
E
@BEEP(0)
RUN("/home/stelio/cprog/phase",-1) @CR
@call("spet",120000000)@CR
```

```
#include <stdio.h>
#include "/usr/nivxi/include/nivxi.h"
#include "spet.h"


 main()

{
int phase[6];
uint16 ph;
uint8 result;
uint8 i;
FILE *inpu;
inpu=fopen("/home/stelio/cprog/phase.dat","r");
/*fscanf(inpu,"%d", phase);*/

result=InitVXIlibrary();
resetvmio12();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/


enable('2');
enable('3');


for (i=0;i<=5;i++)
{
fscanf(inpu,"%d\n",phase+i);
/*printf("%d\n", phase[i]);*/

ph=64*phase[i]/360;
subcio_out(ph,cio_pshift_data,i,cio_pshift_add);

}
 result=CloseVXIlibrary();

}
```

```
@BEEP(0)
RUN("/home/stelio/cprog/setdig1",-1) @CR

@POP("p1",-1,0,"DATA ACQUISITION PHASE: ACTIVE")
RUN("/home/stelio/cprog/interf",-1) @CR
@UNPOP("p1")
!RUN("dadimp -B -C -F=ASCII -H=He -L=int /home/stelio/cprog/ad.dat  -O",-1)
@CNTL_HOME
setwf('SETYLOG(ON)')@CR
setwf('READA("/home/stelio/cprog/ad.dat")')@CR
@F6
@F6
@F6
@HIGHLIGHT_MESSAGE("-RFI CONTROL SYSTEM-")
@pause(31)
@F2
@pause(9)
@F2
@F6
```

```
/*  PROGRAMMABLE DIGITIZER CONTROL*/
/* Program Name: setdig1.c */
#include <stdio.h>
#include "nivxi.h"
#include "spet.h"
#include <math.h>
#define freq 15.0
#define fft_size 2048

void digitiz (port,value)
uint8 port;
uint8 value;

      {
        int16 ret;

                    subcio_out(value, cio_dig_data, port, cio_dig_add);
                    ret=subcio_in(cio_dig_read, port, cio_dig_add);
                    if(ret != value)
                    {
                    printf("Be careful, error in write or read!!\n");
                    printf("Check  the LOC/REM Switch!!\n");
                    }

      }


 main()

{
int response, i;
uint8 result[2048];
int16 ret;
uint8 dataw,resul;
float res;
int m;
FILE *inpu;
uint8 dmil, mil, cm, dm, im, ic, id, iu;
uint8 dataw1, dataw2, dataw3, dataw4;
float fdummy;
int ddummy;
uint32 dummy;
inpu=fopen("/home/stelio/cprog/ad.dat","w+");
resetvmio12();

resul=InitVXIlibrary();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/
enable('2');
enable('3');


initvmio12('0','b','o','n'); /*port b of IC 6 becomes out (Clock Read)*/
initvmio12('1','a','i','n'); /*port b of IC 7 becomes in (data+flags)*/
initvmio12('1','b','i','n'); /*port b of IC 7 becomes in (flags)*/
enable('0');
enable('1');


/* Reset  --_- */
dataw=0;
subcio_out (dataw,cio_dig_data,refgen_reset,cio_dig_add);
dataw=1;
subcio_out (dataw,cio_dig_data,refgen_reset,cio_dig_add);



/*=================================================================*/
/* SETTING OF THE REFERENCE GENERATOR BLOCK */

/*inp: printf("RESULUTION? (1/200 KHz) \n");
```

```
scanf("%d",&res);*/
res=0.1;
m=1000/res;


/*printf("Requested Resolution= %.3f KHz \n",res);*/


im=m/1000;
ic=(m-im*1000)/100;
id=(m-im*1000-ic*100)/10;
iu=m-im*1000-ic*100-id*10;

ddummy=im*1000+ic*100+id*10+iu;
fdummy= 1000./ddummy;
/*printf("Real Resulution= %.3f KHz \n", fdummy);*/

dataw1=iu+id*16;
dataw2=ic+im*16;

digitiz (refgen_lsb,dataw1);
digitiz (refgen_msb,dataw2);

/* Status Control   */
  ret=subcio_in (cio_dig_read, refgen_status, cio_dig_add);
/*  printf("status of Refgen. = %u \n", ret);*/

    if((ret & 0x1) == 1 )
     printf("REF. FREQ. BOARD IS IN LOCAL STATUS \n");

    if((ret & 0x2) == 0 )
      printf("5 MHz  failure \n");

/*=========================================================================*/
/*SETTING OF THE PLL FREQUENCY */

/*printf("OUTPUT PLL FREQUENCY (Freq.) ? (10.200--> 17.200 MHz (XX.xxx MHz)  \n");
printf("Out Frequency= Resolution(KHz) x Entered Freq.(MHz) (MHz)  \n");

scanf("%f", &freq);*/
fdummy=freq;
fdummy=fdummy*1000;

fdummy=fdummy/res;
ddummy=fdummy;

dm=ddummy/10000;
im=(ddummy-dm*10000)/1000;
ic=(ddummy-dm*10000-im*1000)/100;
id=(ddummy-dm*10000-im*1000-ic*100)/10;
iu=(ddummy-dm*10000-im*1000-ic*100-id*10);

fdummy=(dm*10000+im*1000+ic*100+id*10+iu)/10000.;
/*printf("Entered Frequency = %.3f \n", fdummy);*/

dataw1=iu+id*16;
dataw2=ic+im*16;
dataw3=dm;

digitiz (pll_lsb,dataw1);
digitiz (pll_msb1,dataw2);
digitiz (pll_msb2,dataw3);


/* PLL Status Control   */
   ret=subcio_in (cio_dig_read, pll_status, cio_dig_add);

/*   printf("PLL Status = %u \n", ret);*/

    if((ret & 0x1) == 0 )
   printf("PLL unlocked. Ceck Input Parameters! \n");
```

```
/* PROG. DIV. Status Control    */
 ret=subcio_in(cio_dig_read, pdiv_status, cio_dig_add);
/*printf("Prog. Div. Status = %u \n", ret);*/

     if((ret & 0x1) == 1 )
   printf("Board in Local Status! \n");

   if((ret & 0x2) == 2 )
   printf("Be Careful, Data FIFO Overflow!!\n");


resul=CloseVXIlibrary();

}
```

```
/*  RADIO INTERFERENCE CONTROL PROGRAM*/

/* Program Name: interf.c */
#include <stdio.h>
#include "nivxi.h"
#include "spet.h"
#define DUMMY 2048
#define NINTEG 20
#include <math.h>



void digitiz (port,value)
uint8 port;
uint8 value;

      {
        int16 ret;

                    subcio_out(value, cio_dig_data, port, cio_dig_add);
                    ret=subcio_in(cio_dig_read, port, cio_dig_add);
                    if(ret != value)
                    {
                    printf("Be careful, error in write or read!!\n");
                    printf("Check  the LOC/REM Switch!!\n");
                    }

      }

/*============================ FFT Computation ============================*/



void realft(data,n,isign)
float data[];
int n,isign;
{
        int i,i1,i2,i3,i4,n2p3;
        float c1=0.5,c2,h1r,h1i,h2r,h2i;
        double wr,wi,wpr,wpi,wtemp,theta;
        void four1();

        theta=3.141592653589793/(double) n;
        if (isign == 1) {
                c2 = -0.5;
                four1(data,n,1);
        } else {
                c2=0.5;
                theta = -theta;
        }
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;


        wpi=sin(theta);
        wr=1.0+wpr;
        wi=wpi;
        n2p3=2*n+3;
        for (i=2;i<=n/2;i++) {
                i4=1+(i3=n2p3-(i2=1+(i1=i+i-1)));
                h1r=c1*(data[i1]+data[i3]);
                h1i=c1*(data[i2]-data[i4]);
                h2r = -c2*(data[i2]+data[i4]);
                h2i=c2*(data[i1]-data[i3]);
                data[i1]=h1r+wr*h2r-wi*h2i;
                data[i2]=h1i+wr*h2i+wi*h2r;
                data[i3]=h1r-wr*h2r+wi*h2i;
                data[i4] = -h1i+wr*h2i+wi*h2r;
                wr=(wtemp=wr)*wpr-wi*wpi+wr;
                wi=wi*wpr+wtemp*wpi+wi;
        }
        if (isign == 1) {
                data[1] = (h1r=data[1])+data[2];
                data[2] = h1r-data[2];
        } else {
```

```
                data[1]=c1*((h1r=data[1])+data[2]);
                data[2]=c1*(h1r-data[2]);
                four1(data,n,-1);
        }
}


#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(data,nn,isign)
float data[];
int nn,isign;
{
        int n,mmax,m,j,istep,i;
        double wtemp,wr,wpr,wpi,wi,theta;
        float tempr,tempi;

        n=nn << 1;
        j=1;
        for (i=1;i<n;i+=2) {
                if (j > i) {
                        SWAP(data[j],data[i]);
                        SWAP(data[j+1],data[i+1]);
                }
                m=n >> 1;
                while (m >= 2 && j > m) {
                        j -= m;
                        m >>= 1;
                }
                j += m;
        }
        mmax=2;
        while (n > mmax) {
                istep=2*mmax;
                theta=6.28318530717959/(isign*mmax);
                wtemp=sin(0.5*theta);
                wpr = -2.0*wtemp*wtemp;
                wpi=sin(theta);
                wr=1.0;
                wi=0.0;
                for (m=1;m<mmax;m+=2) {
                        for (i=m;i<=n;i+=istep) {
                                j=i+mmax;
                                tempr=wr*data[j]-wi*data[j+1];
                                tempi=wr*data[j+1]+wi*data[j];
                                data[j]=data[i]-tempr;
                                data[j+1]=data[i+1]-tempi;
                                data[i] += tempr;
                                data[i+1] += tempi;
                        }
                        wr=(wtemp=wr)*wpr-wi*wpi+wr;
                        wi=wi*wpr+wtemp*wpi+wi;
                }
                mmax=istep;
        }
}

#undef SWAP


/*=============================================================================*/



 main()

{
int response, i;
uint8 res, resul, ddummy;
int m;
int16 ret,dummy, n,k;
uint8 dmil, mil, cm, dm, im, ic, id, iu;
uint8 dataw1, dataw2, dataw3, dataw4;
```

```
float result[DUMMY], inter[DUMMY];
double media[DUMMY];

FILE *inpu;
inpu=fopen("/home/stelio/cprog/ad.dat","w+");

res=InitVXIlibrary();
outvmio12('0','b',0);


/*SETTING OF THE PROGRAMMABLE DIVIDER */
dummy=DUMMY;
im=dummy/1000;
ic=(dummy-im*1000)/100;
id=(dummy-im*1000-ic*100)/10;
iu=(dummy-im*1000-ic*100-id*10);


/* data to write */
dataw1=iu+id*16;
dataw2=ic+im*16;

/* data writing */
digitiz (pdiv_lsb,dataw1);
digitiz (pdiv_msb1,dataw2);


for (i=0; i<DUMMY; i++)
{media[i]=0;}



/* NINTEG loop starting*/
for(n=1; n<=NINTEG;n++)
{

/* Load and Start  --_-*/
ddummy=1;
subcio_out(ddummy, cio_dig_data, pdiv_ls, cio_dig_add);
ddummy=0;
subcio_out(ddummy, cio_dig_data, pdiv_ls, cio_dig_add);
ddummy=1;
subcio_out(ddummy, cio_dig_data, pdiv_ls, cio_dig_add);


/* PROG. DIV. Status Control
 ret=subcio_in(cio_dig_read, pdiv_status, cio_dig_add);
     if((ret & 0x1) == 1 )
   printf("Board in Local Status! \n");

   if((ret & 0x2) == 2 )
   printf("Be Careful, Data FIFO Overflow!! \n");*/


/*================ =====A/D DATA READING ====================*/
for (i=0; i<DUMMY; i++)
{
outvmio12('0','b',128);
outvmio12('0','b',0);
result[i]=(invmio12 ('1','a') & 63);
}

/*======================FFT==================================*/

realft(result-1, DUMMY/2, 1);

/*===========================================================*/



/*===============Square and Average=========================*/

for (i=2;i<DUMMY; i=i+2)
{
```

```
/*square modulo and Average */
inter[i]=(result[i]*result[i]+result[i+1]*result[i+1]);
media[i]=(media[i]*(n-1)+inter[i])/n;
}
}

for (i=2; i<DUMMY; i=i+2)
{
fprintf(inpu,"%f \n",media[i]);
}



res=CloseVXIlibrary();

}
```

6. IL SISTEMA VME.

Il cuore dello spettrometro viene riportato in Fig.25 ed e` alloggiato in un crate VME. Questo e` composto da un convertitore A/D della Valley Tech. Ultra ADC da 40 MS/Sec. con 10 bit di risoluzione (Fig.25a), una scheda Ultra DSP (Fig. 25b)per il computo della FFT equipaggiata con il velocissimo DSP Sharp LH9124, dalla scheda VT-501 che fornisce il valore del modulo quadro e clock ed infine di una ulteriore scheda Ultra DSP per il calcolo on line della media prima che il risultato venga scaricato su nastro (Exabyte da 2.2 Gbyte, 8 mm). Tutto il crate VME e` controllato da una Sparc Station Sun compatibile, attraverso un link National Instr. NI-VXI (Fig.25c) e sincronizzato dal clock, generato da un DDS programmabile agganciato allo standard di stazione [3], distribuito attraverso la scheda Mag. Clock Distributor (Fig.25).

I dati convertiti dalla Ultra ADC vengono passati alla scheda di calcolo Ultra DSP attraverso il connettore P2 del bus VME tramite un bus custom (Valley Tech.) ad elevata velocita` a 24 bit. Quest'ultima e` equipaggiata con un chip della Sharp tipo LH9124 (e relativi address generators LH9320) che rappresenta lo stato dell'arte dei DSP ed e` in grado di calcolare, a velocita` 6/10 volte maggiori di tutti i DSP disponibili oggi in commercio, FFT reali di varie dimensioni (sempre comunque in potenze di 2). Il chip, come già visto nel capitolo 3, ha una architettura particolarmente orientata al computo della FFT e lavora con 48 bit complessi (accumulatore) in aritmetica tipo block floating point. Questo significa che l' esponente di ogni dato all' interno dello stesso blocco (es. singola FFT) è lo stesso e che i dati tipo "integer" che provengono dal convertitore A/D sono considerati tutti "real" con esponente 1. Il chip LH9124 ha al suo interno due accumulatori da 60 bit che gli permettono di calcolare FFT di array molto lunghi senza incappare in problemi di overflow. La scheda calcola la trasformata di Fourier in maniera estremamente efficiente come riportato nella Tav.4 e può eseguire Butterfly Radix-2, Radix-4 o Radix-16 (si ricorda che il termine Radix si riferisce al numero di punti in ingresso / uscita di un uno stadio Butterfly di una FFT). Il Radix-2 ha due dati in ingresso e presenta due dati in uscita, il Radix-4 ha 4 dati in ingresso e 4 dati in uscita e, allo stesso modo, un Radix-16 ha 16 dati di input e ne presenta 16 di uscita. Per ottenere una FFT di un determinato numero di punti (potenza di 2) è opportuno scegliere la giusta combinazione di Radix. Per esempio, 256 punti possono essere maneggiati come 16x16, 16x4x4, 16x4x2x2 etc..per cui nella scelta della giusta combinanzione si deve tenere conto di due criteri principali: il rapporto segnale rumore che si ottiene dal quel Radix e la throughput rate cho lo stesso permette. Ad es. un Radix-16 è molto

efficiente da un punto di vista della velocità ma per evitare overflow, il dato deve essere normalizzato (shiftato) di più in confronto di un Radix-2. Infatti un Radix-2 richiede uno shift di 2 bit, un Radix-4 uno shift di 3 bit ed un Radix-16 di 5 bit. Ricordando che il convertitore fornisce dati codificati a 10 bit (1024 livelli), gli ultimi 5 bit verrebbero persi in un Radix-16 causando in tal modo una grossa riduzione di range dinamico. D' altra parte però, un Radix-16 è all' incirca 3.5 volte più veloce di un Radix-2. La perdita di dinamica può rappresentare un problema se si è in presenza di forti disturbi (RFI) a banda stretta in qualche regione dello spettro analizzato, cosa questa che normalmente non dovrebbe accadere nelle osservazioni radioastronomiche. La dimensione della FFT e` programmabile dall'utilizzatore attraverso il software di controllo del sistema e quest' ultimo cerca di massimizzare, dove possibile, l' uso dei Radix-16.

Lo spettro fornito e` positivo (single sided). Se e` necessario avere lobi secondari molto bassi (al disotto dei 50 dB), e` possibile effettuare una "windowing" sui dati in ingresso della Ultra DSP medianti vari tipi di finestre: Hanning, Hamming, Kaiser-Bessel o user defined. Il passaggio della windowing, da parte della scheda Ultra DSP, non porta via ulteriore tempo purche` il primo blocco di calcolo della FFT sia un radix-2 o radix-4. I dati cosi` trattati vengono memorizzati su uno dei banchi di memoria del DSP, trasformati e quindi resi disponibili sul banco di uscita dello stesso. A questo punto le stringhe di dati vengono passate alla scheda VT-501 che, oltre a distribuire il clock di sistema come visto in precedenza, ha anche il compito di calcolare il modulo quadro di ogni coppia di cui e` costituito lo spettro complesso. Le stringhe di dati reali cosi` ottenute vengono poi passate, sempre attraverso il bus custom sul connettore P2, ad un' altra scheda Ultra DSP uguale alla precedente. Questa ha il compito di mediare on line un numero di spettri programmato dall'utente nella fase di set-up del sistema, per ridurre la varianza del noise nella stima dello spettro. E` possibile mediare un numero di spettri che puo` variare da 1 a 64K. Quando viene richiesta la media di meno di 256 spettri (1-->256 a step di 1), la Ultra DSP la calcola usando una aritmetica a 24 bit fixed point e converte il risultato in floating point, in tal modo non si perde precisione nel calcolo della media. Quando viene richiesta la media di piu`di 256 spettri (256 -->64K a step di 256), la media di ogni blocco da 256 punti viene calcolata in fixed point, convertita in floating point e quindi mediata con il risultato gia` ottenuto applicando lo stesso procedimento ai precedenti blocchi. Lo spettro risultante cosi` mediato in floating point viene poi caricato nella Sparc dove viene visualizzato sullo schermo per una prima analisi qualitativa e, nello stesso tempo, memorizzato su un nastro da 8 mm tramite una unita` tipo Exabyte.

Nella Fig. 26 viene riportata la foto del sistema come si presentava, a realizzazione ultimata, nella prima versione base usata per le osservazioni degli effetti dell'impatto della cometa SL-9 con l' alta atmosfera di Giove (Luglio 94). Viene riportato il listato completo del programma sorgente scritto dalla Valley ed usato per le osservazioni dell' impatto Giove /SL-9.

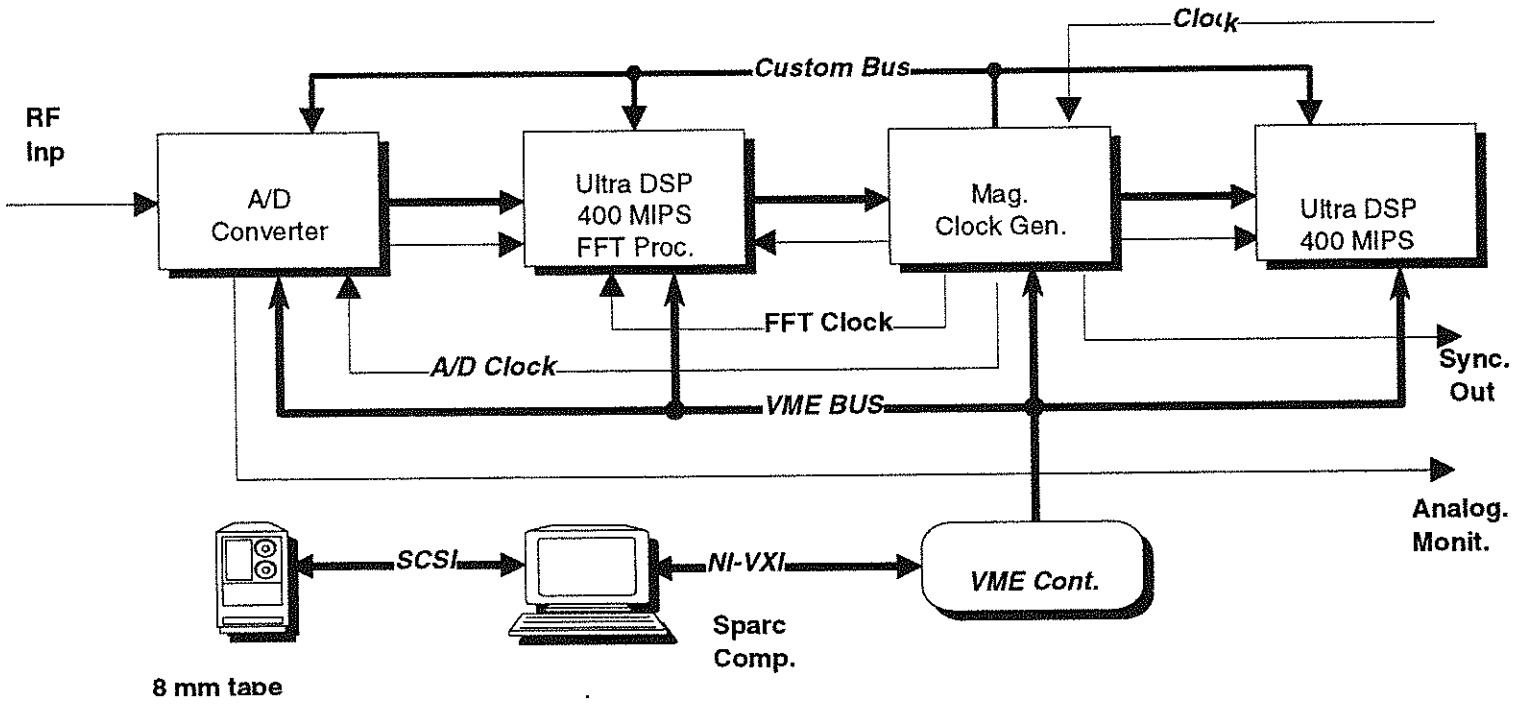

Clock
Custom Bus
RF
Inp
A/D
Converter
Ultra DSP
400 MIPS
FFT Proc.
Mag.
Clock Gen.
Ultra DSP
400 MIPS
FFT Clock
A/D Clock
VME BUS
Sync.
Out
Analog.
Monit.
SCSI
NI-VXI
VME Cont.
Sparc
Comp.
8 mm tape

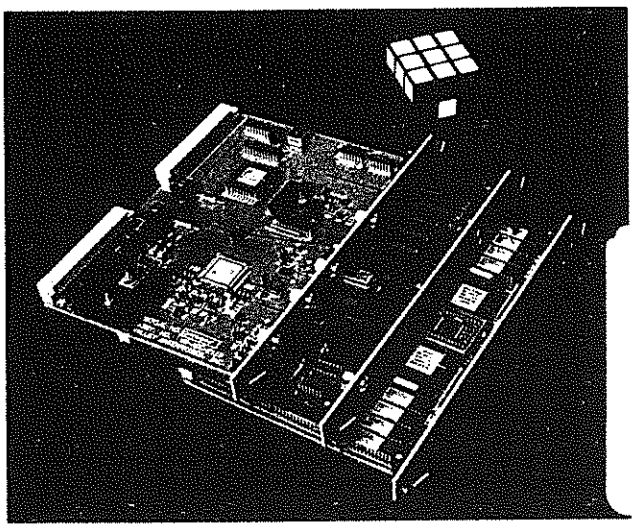
FIG. 25

DIGITIZER MODULE

VE-ADC-2110

FEATURES

- 10 bit ADC
- 40 MHz Sample Rate
- 125 MHz Input Bandwidth
- 55 dB SNR
- Harmonics: -65 dBc
- 1K Data FIFO
- DAC Output
- VME 6U x 160 mm



Our Name is Now

VALLEY TECHNOLOGIES, INC

RD #4, Rt. 309, Tamaqua, PA 18252

Ph. (717) 668-3737 Fax (717) 668-63E

DESCRIPTION

The VE-ADC-2110 is a single channel VMEbus Digitizer Module. It is compatible with the VE-32C-xxV series of DSP modules, providing a complete integrated solution for signal digitization and processing.

This digitizer combines high speed sampling of up to 40 MHz with a wide 10 bit dynamic range. Very low noise and distortion permit its use in performance critical applications.

Front panel SMA connectors are provided for signal input, digitizing clock, and DAC output to prevent corruption by backplane noise. A front panel overload indicator and a FIFO status indicator are provided.

Internal conditioning provided for sine or TTL digitizing clocks. The capability is provided to terminate or daisy-chain the digitizing clock.

Data FIFO allows for different digitizing and data clock rates. The parallel data bus output is on the P2 connector.

APPLICATIONS

- Wideband Spectral Analysis
- Video Digitization
- RF Sampling and Downconversion
- Digitizing Scope
- CT & MRI Medical Imaging
- Radar Signal Digitization

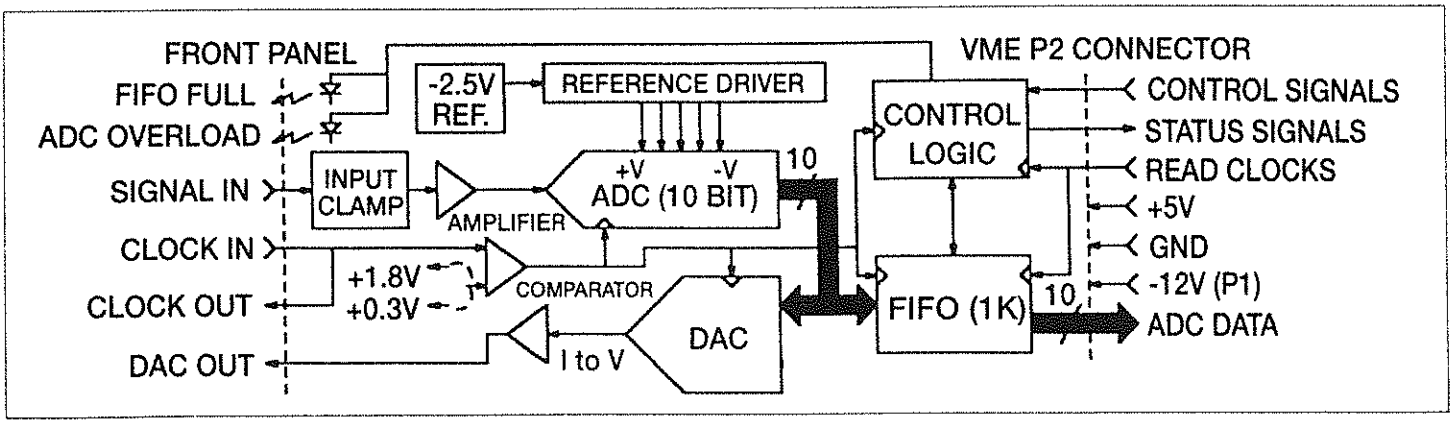


FIG 25a

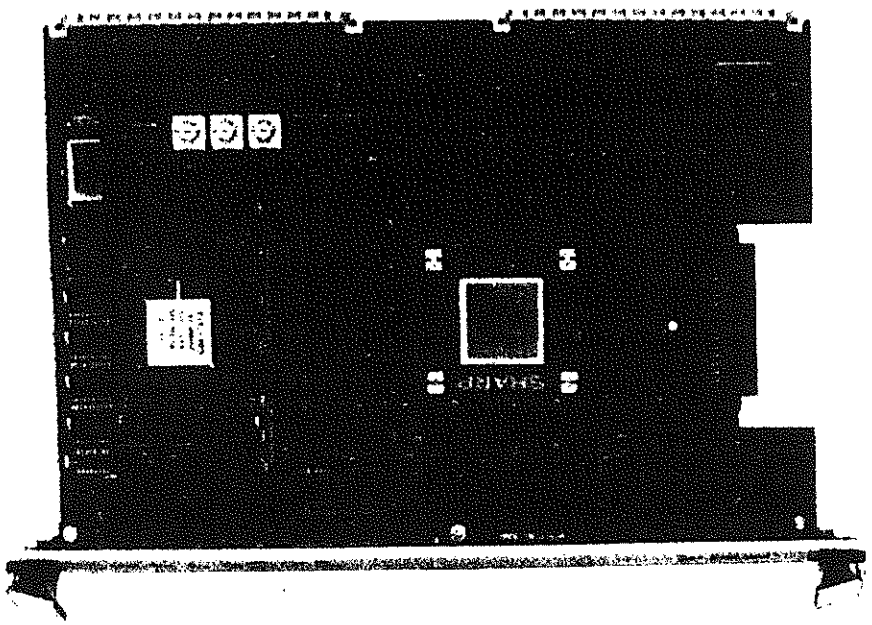
*Ultra*DSP — Ultra*Fast processor for FFT, filtering, or matrix calculatio*

Features and Benefits

- 400 MIPS performance: 1k complex FFT in 106 µsec and FIR Filters with 12.5 nsec/tap
- 24-bit block-floating point arithmetic for high precision
- Up to 128k complex or 256k real FFT in a single-slot 6U VME board
- Radix-2, Radix-4, and Radix-16 FFT butterfly operations
- I/O: 24-bit real at 40 Msamples/sec; 48-bit complex at 20 Msamples/sec
- Expandable to 9U size for 48-bit I/O at 40 Msamples/sec
- 256 kword on-board memory
- Uses the new 40 MHz Sharp FFT DSP and address generator chip set
- On-board 40 MHz general-purpose DSP for easy programming and management of hardware resources
- Choice of mezzanine modules to suit your I/O requirements
- Extensive software support for high-level programming
- Low power consumption and proven reliability

Applications

- Spectrum Analysis
- Radar Doppler Processing
- Sonar Beamforming
- Adaptive Filtering and Control
- Frequency Domain Filtering
- Time Domain Convolution
- Multichannel Correlation
- Multichannel Echo Cancellation
- 2-D Medical Imaging
- Image Compression/Decompression



Blazing Speed

The Valley Technologies *Ultra*DSP VME board brings you unprecedented performance for real-time applications such as complex or real FFTs, FIR filters and convolutions. Utilizing the Sharp LH9124 24-bit block-floating point DSP for calculations, and the AT&T DSP32C for supervision and easy programming, the *Ultra*DSP computes up to 128k complex, or 256k real FFTs at blazing speeds.

One way to appreciate its level of performance is to compare it with better known processors, such as the Texas Instruments TMS320C40 and the Intel *i*860: the *Ultra*DSP can perform a 64k FFT faster than twenty 'C40s, or eleven *i*860s!

To balance the tremendous processing power with a comparable I/O rate, the *Ultra*DSP has FIFO-buffered input and output ports which are 24 bits wide and run at speeds up to 40 Msamples/sec

You can easily configure the board for 24-bit real data I/O; 24-bit complex I/O with multiplexed real and imaginary parts; or 12-bit complex I/O with 12-bit real and 12-bit imaginary in parallel. These configurations are possible with many standard mezzanine expansion modules available from Valley Technologies.

The *Ultra*DSP outshines competitive offerings in all respects, including processing speed, memory and FFT size, and ease of programming. With some other designs you have to program the FFT processor in assembly language. Valley Technologies on the other hand, gives you an elegantly simple set of instructions that you can code in C-language.

A complete data acquisition and signal analysis system can be configured by connecting the *Ultra*ADC high-speed A/D and the *Ultra*Buf high-speed buffer memory to the *Ultra*DSP. Valley Technologies has carefully planned and designed these three products, in order to provide you with a complete solution to your most demanding signal processing problem.

Mezzanine I/O

Several I/O mezzanine boards are available to suit individual requirements. They all plug into the *Ultra*DSP mainboard, while maintaining the board's one-slot envelope. Some examples are described here and Valley Technologies will gladly address custom requirements.

FIG. 25b

Table 4

FFT Size	Complex Best	Worst	Real Best	Worst
1K	106 us	141 us	90 us	112 us
2K	244 us	305 us	141us	176 us
4K	448 us	448 us	305 us	365 us
8K	858 us	1.07 ms	561 us	561 us
16K	1.68 ms	2.10 ms	1.07 ms	2.97 ms
32K	4.14 ms	4.97 ms	2.10 ms	2.97 ms
64K	8.24 ms	8.24 ms	4.97 ms	5.80 ms
128K	16.43 ms	19.72 ms	9.89 ms	9.89 ms
256K	-	-	19.72 ms	23.01 ms

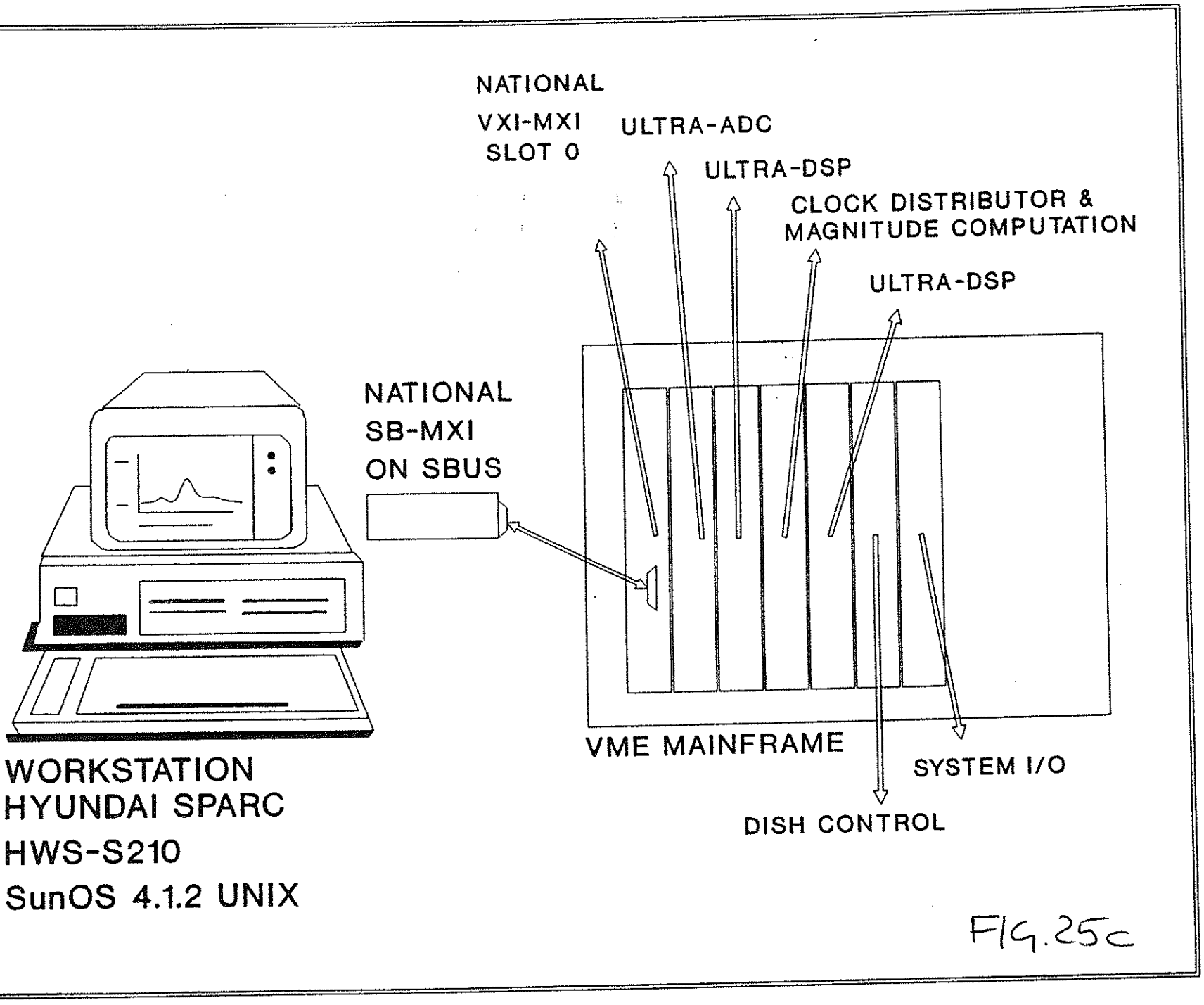

NATIONAL
VXI-MXI
SLOT 0
ULTRA-ADC
ULTRA-DSP
CLOCK DISTRIBUTOR &
MAGNITUDE COMPUTATION
ULTRA-DSP
NATIONAL
SB-MXI
ON SBUS
VME MAINFRAME
SYSTEM I/O
DISH CONTROL
WORKSTATION
HYUNDAI SPARC
HWS-S210
SunOS 4.1.2 UNIX

FIG. 25c

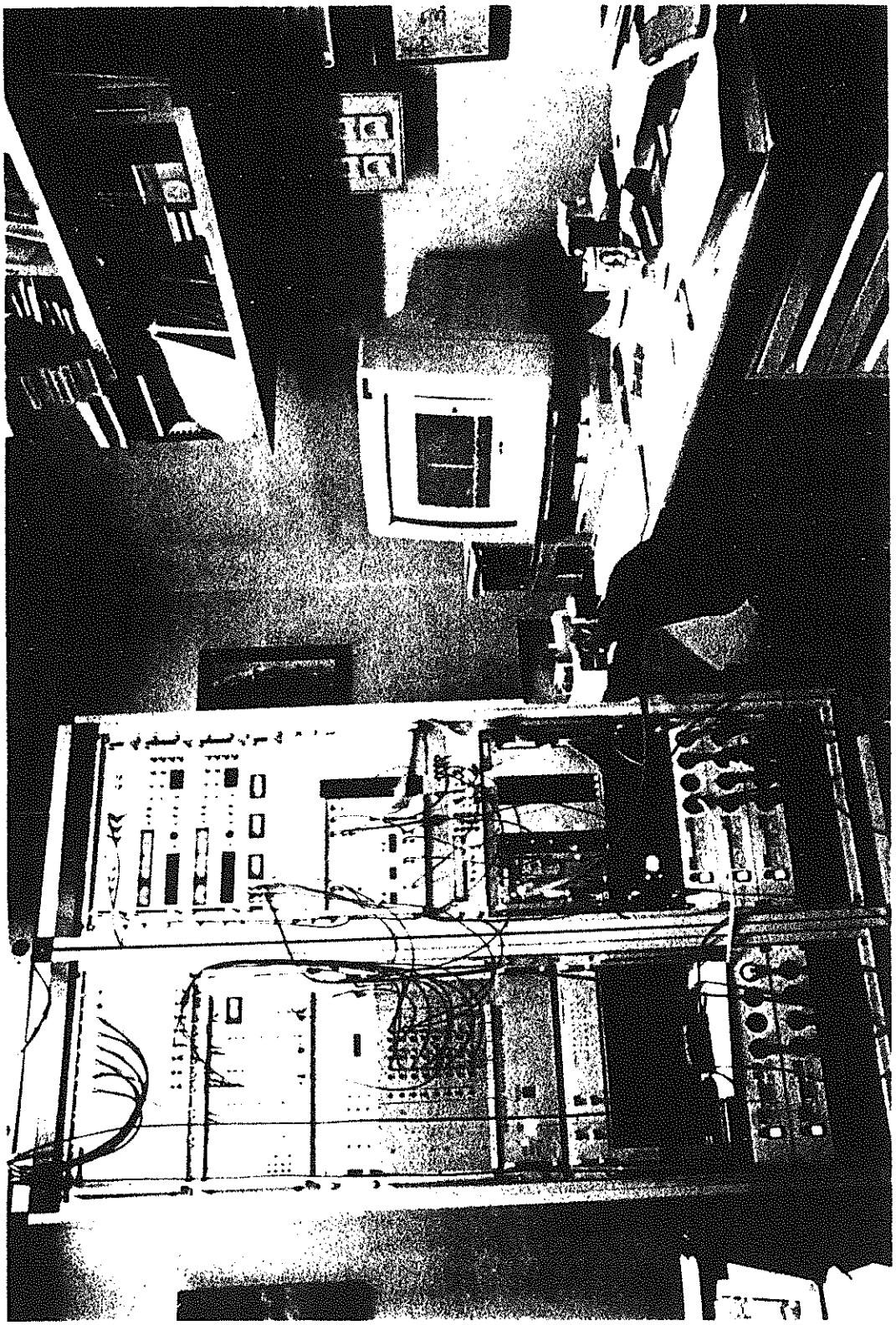


FIG. 26

```
/*
 ************************************************************************
 ************************************************************************
 **                                                                    **
 **    File Name: jcntrl.c                                             **
 **                                                                    **
 **    Description: This file contains the routines that contitute     **
 **       the main control task for the Jupiter Project                **
 **                                                                    **
 **                                                                    **
 **                                                                    **
 **                                                                    **
 **                                                                    **
 **    Modules:                                                        **
 **                                                                    **
 **    Author: d. kolesar                                              **
 **                                                                    **
 **    Language:   C                                                   **
 **                                                                    **
 **    Build Commands:                                                 **
 **                                                                    **
 **    Update Log                                                      **
 **                                                                    **
 **    Date       Initials   Description                               **
 **                                                                    **
 **    5/30/94    dmk        Creation                                  **
 **    6/28/94    dmk         Add sending restart command to avemag    **
 **                           after writing to tape                    **
 **                                                                    **
 ************************************************************************
 ************************************************************************
 */
/** include files **/

#include <stdio.h>
#include <strings.h>
#include <fcntl.h>
#include <math.h>
#include <errno.h>
#include <sys/ioctl.h>


#include <sys/time.h>
#include <unistd.h>
#include <signal.h>

#define INT24 long
#include "jupiter.h"


/** local definitions **/


/** Local Data **/

        int fftpipe;    /* File Descriptor for FFT UltraDSP */
        int avepipe_wr;    /* File Descriptor for averager UltraDSP writing*/
        int avepipe_rd;    /* File Descriptor for averager UltraDSP reading */

        long *buffer1,*buffer2; /* A buffer */
        float *tapebuf; /* Tape data Buffer */

        struct Operational_Parameters OpParam;

        struct Operational_Commands OpCmd;


        static int fftid, aveid, plotid;  /* Process Ids of the child processes */

        struct RecordHeader RecHdr ;

        int source_flag;

/** define the FFT application programs that will be selectivly
```

```
    loaded in the FFT UltraDSP for processing. The index into this character
    array = 11 - log(base2) of size*1024)
 **/

        char *ffts[9] =
        {
                "fft1k",
                "fft2k",
                "fft4k",
                "fft8k",
                "fft16k",
                "fft32k",
                "fft64k",
                "fft128k",
                "fft256k"
        };

        char *windows[9] =
        {

                "win1k.dat",
                "win2k.dat",
                "win4k.dat",
                "win8k.dat",
                "win16k.dat",
                "win32k.dat",
                "win64k.dat",
                "hwin128k.dat",
                "hwin256k.dat"
        };

/** external functions **/

        void dmparam( );
        long LONG32C( );        /* Swaps the bytes for DSP32C Long Word */
        float dsp_to_ieee( );   /* Converts DSP32C float to host format */
        int block_read( );
        int block_write( );

/** external data **/

main(argc,argv)
        int argc;
        char *argv[];
{
        int i,j,k;      /* Generic int for counters and such */

        FILE *outfile, *display; /* Output file and display pointers */

        long magpairs;  /* Number of magnitude pairs (even & odd bins) */
        long numbuf;    /* Number of 1024 mag pairs to upload */

        char Outfile[20];   /* Output File name */
        int status;         /* Generic status return variable */
        int onsource = 0;   /* On source block counter */
        int offsource = 0;  /* Off Source Counter */
        char station_time[5];  /* Station time at time of start */
        struct timeval tp;     /* Computer Time   "  ""  """*/
        struct timezone tzp;   /* Time Zone "" */



/** Initialize the VMIO12 Board(s) **/

        InitVMIO12( );


/*** Create the Named Pipes for Communication with the UltraDSPs **/


    /* At this time the pipes must be previously created using mknod */


/** Fork Out the Hostio and spectral plotting Processes **/
```

```
        if((fftid = fork( )) == 0)    /* Launch FFT Program */
        {
                execlp("hostsb","hostsb","0","init",NULL);
        }

        if((aveid = fork( )) == 0)    /* Launch Averager Program */
        {
                execlp("hostsb","hostsb","1","avemag",NULL);
        }

        if((plotid = fork( )) == 0)    /* Launch Plotting Program */
        {
                i = execlp("specplot","specplot",NULL);
                if(i == -1)
                   perror("jcntrl: ");
        }


        sleep(3);  /* Wait for hostio to start */


/** Setup the UltraFFTs **/


        if(setup(argc,argv) != 0) jexit(0);


/*******************************************************/
/**                                                   **/
/**       Read the AVE Pipe, receiving the            **/
/**       averaged spectrum and writing it out to     **/
/**       tape                                        **/
/**                                                   **/
/*******************************************************/

/** Allocate the buffers to receive the data **/

        magpairs = RecHdr.fft_size/4; /* Number of  even or odd mag samples */

        if( (buffer1 = (long *)calloc(magpairs, 4)) == NULL)
             { fprintf(stderr,"Unable to Alloc Buffer 1\n"); jexit(0);}

        if( (buffer2 = (long *)calloc(magpairs, 4)) == NULL)
             { fprintf(stderr,"Unable to Alloc Buffer 2\n"); jexit(0);}

        if( (tapebuf = (float *)calloc(magpairs*2, 4)) == NULL)
             {fprintf(stderr,"Unable to Alloc tape buffer\n"); jexit(0);}


/** Open the Tape Device or the device specified on the command line ***/

        if(argc <=1)
           outfile = fopen("/dev/rst1","wb");
        else
           outfile = fopen(argv[1],"wb");

        if(outfile == 0)
        {
             fprintf(stderr,"Error Opening output device\n");
             jexit(errno);
        }
/** Open the spectral plot pipe */

        display = fopen("SPECPIPE","wb");
           if(display == NULL)
            {
                printf("\njcntrl: Error Opening SPECPIPE\n");
                jexit(errno);
             }

/*** Build part of the Header that doesn't change ***/

        RecHdr.sync[0] = 0xa5a5001;
        RecHdr.sync[1] = 0xa5a5002;
```

```
        RecHdr.sync[2] = 0xa5a5003;
        RecHdr.sync[3] = 0xa5a5004;

/* Set the onsource counter and flag*/

        onsource = RecHdr.onsource;
        RecHdr.source_flag = ONSOURCE;


/**** Loop Forever processing the data **/

        RecHdr.blknum = 0;

        source_flag = ONSOURCE;

        ForceOnSource( );

        while(RecHdr.blknum >= 0)     /* Do Forever */


        {
/*
 * Wait for last average cycle to finish
 * This is determined by data in the avepipe
 *
 */
           if(RecHdr.blknum != 0)
           {
             i= 0;
             while( i == 0)
             {
                ioctl(avepipe_rd,FIONREAD, &i);
                if(i==0) sleep(1);
             }
           }

/** Save current value of source flag in header */

        RecHdr.source_flag = source_flag;


/** Command the antenna off source if it is time **/

        if(onsource > 0)
        {
           onsource--;
           if(onsource == 0)
           {
               SetOffSource( );
               offsource = RecHdr.offsource+1;
               source_flag = OFFSOURCE;
           }
        }


/** Command the antenna back On Source if it is time ***/

        if(offsource > 0)
        {
           offsource--;
           if(offsource == 0)
           {
               SetOnSource( );
               onsource = RecHdr.onsource;
               source_flag = ONSOURCE;
           }
        }


/** send the restart command to the averager **/

        OpCmd.cmd = LONG32C(RESTART_AVE);
        OpCmd.param[0] = LONG32C(RecHdr.blknum);
        write(avepipe_wr,&OpCmd.cmd,sizeof(OpCmd));
```

```
/** Update the header with the previous station time and system time */

        memcpy(&RecHdr.station_time[0], station_time, sizeof(station_time));
        memcpy(&RecHdr.tp, &tp, sizeof(tp));
        memcpy(&RecHdr.tzp, &tzp, sizeof(tzp));


/** Get the current the station time and the system time **/

        ut(station_time);
        gettimeofday(tp,tzp);

/** If this is thew first block, break here and go to top of loop
    and wait for the average to finish */

        if(RecHdr.blknum == 0) {RecHdr.blknum++; continue;}


/** Read the Even Magnitude Sample **/


        if((k=block_read(avepipe_rd, buffer1, magpairs*sizeof(long))) != 0 ) jexit(k)


/** Read the Odd Magnitude Samples  **/

        if((k=block_read(avepipe_rd, buffer2, magpairs*sizeof(long))) != 0 ) jexit(k)


/*** Write out the Header ***/

          if((k=block_write(fileno(outfile), &RecHdr,sizeof(RecHdr)) ) != 0 ) jexit(k)
          if((k=block_write(fileno(display), &RecHdr,sizeof(RecHdr)) ) != 0 ) jexit(k)


/** De-Interleave the samples,
 *   Convert to ieee float and
 *    write to the tape       **/

          j = 0;
          for(i=0; i<magpairs*2; i+=2)
          {
             tapebuf[i]   =  dsp_to_ieee(LONG32C(buffer1[j]));
             tapebuf[i+1] = dsp_to_ieee(LONG32C(buffer2[j]));
             j++;
          }

          if((k=block_write(fileno(outfile), tapebuf, magpairs*2*sizeof(long))) != 0 )

          if((k=block_write(fileno(display), tapebuf, magpairs*2*sizeof(long))) != 0 )


/* Increment the blocknumber */

        RecHdr.blknum++;

        } /** End of Infinite While Loop **/

        fflush(outfile);
        fclose(outfile);
        jexit(0);


}   /* End of main routine */

/*
 ***************************************************************************
 *                                                                         *
 *  Function Name: setup                                                   *
 *                                                                         *
 *  Description: This routine opens the pipes and setups up the UltraDSPs  *
 *                                                                         *
 *  Syntax: int setup(argc,argv[])                                         *
 *
```

```
 *  Parameters: the argc and argv from the main program                    *
 *                                                                         *
 *  Returns: 0 for success, -1 for failure                                 *
 *                                                                         *
 ***************************************************************************
 */

int setup(argc,argv)
        int argc;
        char *argv[];

{
        int size;       /* FFT Size */
        float avetime;  /* Average Time in seconds */
        int i;          /* Generic integer */
        float fs;

/** Open the FFT and Averager Pipes **/


        fftpipe = open("FFTPIPE",O_RDWR);
        if(fftpipe==NULL)
        {
            fprintf(stderr,"error opening FFT pipe\n");
            jexit(-1);
        }

        avepipe_wr = open("AVEPIPEWR",O_RDWR);
        if(avepipe_wr==NULL)
        {
            fprintf(stderr,"error opening AVE Write pipe\n");
            jexit(-1);
        }


        avepipe_rd = open("AVEPIPERD",O_RDWR);
        if(avepipe_rd==NULL)
        {
            fprintf(stderr,"error opening AVE Read pipe\n");
            jexit(-1);
        }



/***** Prompt user for parameters and fill in the data structures **/

        fprintf(stdout,"\nSampling Rate: ");
        fscanf(stdin,"%f",&fs);
        RecHdr.fs = (int)fs;

        fprintf(stdout,"\nFFT Size (in K): ");
        fscanf(stdin,"%d",&size);

        /* Calculate the index into the fft and window arrays */
        /* INdex = power of two above 1K, ie 1K = 0, 2K = 1 ect */

        i = (int) ((log10((float)(size*1024))/log10(2.0))-10);


/** Set the fft application program and the window file */

        strcpy(OpParam.fft_file, ffts[i]);
        strcpy(OpParam.win_file, windows[i]);
        strcpy(RecHdr.fft_file, ffts[i]);
        strcpy(RecHdr.win_file, windows[i]);


/* Set the Window Size and FFT size in Complex # Of Complex Points */

        if(size == 128 )
           OpParam.winsize = LONG32C(size*1024/4);      /* Use half a window */

        else if(size == 256)
           OpParam.winsize = LONG32C((size*1024/4) -1); /* 1 less point for 256 K ffts
```

```
        else
           OpParam.winsize = LONG32C(size*1024/2);

        OpParam.fft_size = LONG32C(size*1024); /* Real  FFT Size */
        RecHdr.winsize =  size*1024/2;
        RecHdr.fft_size = size*1024;            /* Real FFT Size */

/** Calculate the Number of fixed and floating point averages to perform **/

        fprintf(stdout,"\nNumber of Averages: ");
        fscanf(stdin,"%f",&avetime);

        if(avetime < 256.0)
        {
           RecHdr.ave_int =  (long)avetime;
           RecHdr.ave_float = 1;
        }
        else
        {
           RecHdr.ave_int = 256;
           RecHdr.ave_float = avetime/256;
        }

        OpParam.ave_float = LONG32C(RecHdr.ave_float) ;

        OpParam.ave_int = LONG32C(RecHdr.ave_int) ;

/** Ask for the number of on source and off source blocks to process */

        fprintf(stdout,"\nNumber of ON Source blocks: ");
        fscanf(stdin,"%d", &RecHdr.onsource);


        fprintf(stdout,"\nNumber of OFF Source blocks: ");
        fscanf(stdin,"%d", &RecHdr.offsource);


/** Send Parameters to UltraDSPs  **/

        OpParam.cmd = LONG32C(1);

        /* dmparam( ); */

        i = write(fftpipe,&OpParam.cmd,sizeof(OpParam));

        if( i != sizeof(OpParam))
        {
           fprintf(stderr,"\nError Writing to FFT Pipe");
           jexit(-1);
        }

        i = write(avepipe_wr,&OpParam.cmd,sizeof(OpParam));

        if( i != sizeof(OpParam))
        {
           fprintf(stderr,"\nError Writing to AVE Pipe");
           jexit(-1);
        }


        return(0);


}

void dmparam( )
{
        fprintf(stderr,"\ncmd:          %8.8x",LONG32C(OpParam.cmd));
        fprintf(stderr,"\nfs:           %d",OpParam.fs);
        fprintf(stderr,"\nwinsize:      %d",LONG32C(OpParam.winsize));
        fprintf(stderr,"\naveint:       %d",LONG32C(OpParam.ave_int));
        fprintf(stderr,"\navefloat:     %d",LONG32C(OpParam.ave_float));
```

```
        fprintf(stderr,"\nfft:         %s",OpParam.fft_file);
        fprintf(stderr,"\nWindow:      %s",OpParam.win_file);
        return;
}
/*
 ***************************************************************************
 *                                                                         *
 *  Function Name: LONG32C                                                 *
 *                                                                         *
 *  Description: This routine performs a byte swap to convert a big        *
 *    endian to little endian long word                                    *
 *                                                                         *
 *                                                                         *
 *  Syntax: long LONG32C( long ltmp)                                       *
 *                                                                         *
 *  Parameters: long ltmp - long word to byte swap                         *
 *                                                                         *
 *  Returns: swapped long word                                             *
 *                                                                         *
 ***************************************************************************
 */

 long LONG32C(ltmp)
        long ltmp;
{

        char ctmp;    /* A temporary character */

/** Create a union between a long and a 4 byte array **/

        union  { char tbytes[4]; long tlong; } tmp;

        tmp.tlong = ltmp; /* Get the input parameter */

/** Swap the bytes **/

        ctmp = tmp.tbytes[0];
        tmp.tbytes[0] = tmp.tbytes[3];
        tmp.tbytes[3] = ctmp;

        ctmp = tmp.tbytes[1];
        tmp.tbytes[1] = tmp.tbytes[2];
        tmp.tbytes[2] = ctmp;

        return(tmp.tlong);
}

int jexit(status)
        int status;
{

        if(status != 0)
          perror("jcntrl Exiting: ");

        fflush(stderr);
        kill(fftid,SIGKILL); /* Kill the hostio processes */
        kill(aveid,SIGKILL);  /* "" */
        kill(plotid,SIGKILL);  /* "" */

        _exit(status);
}

/*
 ***************************************************************************
 *                                                                         *
 *  Function Name: block_read                                              *
 *                                                                         *
 *  Description: Performs block reads on a file descriptor   lock          *
 *                                                                         *
 *  Syntax: int block_read(fd,buffer,numbytes)                             *
 *                                                                         *
 *  Parameters:                                                            *
 *              int fd;         File Descriptor                            *
 *              char *buffer;   Buffer to read into                        *
 *              int numbytes;   Total number of bytes to read              *
```

```
*                                                                              *
*  Returns: 0 or errno                                                         *
*                                                                              *
*******************************************************************************
*/


int block_read(fd,buffer,numbytes)

        int fd;         /* File Descriptor    */
        char *buffer;   /* Buffer to read into  */
        int numbytes;   /* Total number of bytes to read */


{
        int blksize = 4096;      /* Block size to read */
        register int j=0, k;
        char message[15];

        while(numbytes > 0)
        {
          if(numbytes < blksize) blksize = numbytes;

          k = read(fd,&buffer[j],blksize);
          if( k != blksize)
          {
              sprintf(message,"Block_Read %d",k);
              perror(message);
             /*  return(errno); */
          }
          j+=k;
          numbytes -=k;

        }
        return(0);
}

/*
 *******************************************************************************
 *                                                                             *
 *  Function Name: block_write                                                 *
 *                                                                             *
 *  Description: Performs block writes on a file descriptor                    *
 *                                                                             *
 *  Syntax: int block_write(fd,buffer,numbytes)                                *
 *                                                                             *
 *  Parameters:                                                                *
 *              int fd;         File Descriptor                                *
 *              char *buffer;   Buffer to write to                             *
 *              int numbytes;   Total number of bytes to write                 *
 *                                                                             *
 *  Returns: 0 or errno                                                        *
 *                                                                             *
 *******************************************************************************
 */


int block_write(fd,buffer,numbytes)

        int fd;         /* File Descriptor    */
        char *buffer;   /* Buffer to write from  */
        int numbytes;   /* Total number of bytes to write */


{
        int blksize = 4096;      /* Block size to write */
        register int j=0, k;

        while(numbytes > 0)
        {
          if(numbytes < blksize) blksize = numbytes;

          k = write(fd,&buffer[j],blksize);
          if( k != blksize)
          {
```

```
                perror("Block_write: ");
                return(errno);
            }
            j+=k;
            numbytes -=k;

        }
        return(0);
}
```

```
/*
 ************************************************************************
 ************************************************************************
 **                                                                    **
 **     File Name: rdtape.c                                            **
 **                                                                    **
 **     Description: This file contains the routines to read the       **
 **           files stored during a collection run for the CNR Jupiter **
 **           project. It will read a mission file from  any unix      **
 **           file or device and display the spectra and/or extract a  **
 **           record                                                   **
 **                                                                    **
 **     Modules:                                                       **
 **                                                                    **
 **     Author: d. kolesar                                             **
 **                                                                    **
 **     Language:   C                                                  **
 **                                                                    **
 **     Build Commands: see makefile                                   **
 **                                                                    **
 **     Update Log                                                     **
 **                                                                    **
 **     Date       Initials   Description                             **
 **                                                                    **
 **     5/30/94    dmk        Creation                                **
 **     6/30/94    dmk         Add output name option on command line **
 **                                                                    **
 **                                                                    **
 ************************************************************************
 ************************************************************************
 */
/** include files **/

#include <stdio.h>
#include <strings.h>
#include <fcntl.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <signal.h>



#define INT24 long
#include "jupiter.h"
/* #define TEST_DATA */

/** local definitions **/


        void AnalSpec( );


        struct RecordHeader RecHdr;

        struct SpecAnalResults
        {
                int f1;          /* Bin # of largest signal */
                float p1;        /* Power Level of Largest +/- 3 bins  */
                int f2;          /* Bin # of 2nd largest signal  (spur performance ) *
                float p2;        /* Power of 2nd largest signal */
                float ptot;      /* Total power within the spectrum */
                float np;        /* Noise Power */
                float snr;       /* Largest Signal to total spectral power */
        };


/** Define the command line switches */

#define NUMCMDS 8

        char *flags[NUMCMDS] = { "-f","-o","-a","-n","-p", "-e", "-h","-?"};  /* cmd

        char *help[8] =
        { "\nrdtape command switches\n",
```

```
            " -f <input filename>       defaults to /dev/rst1\n",
            " -o <output filename>      defaults to spec.dat\n",
            " -p                        graphic preview of data\n",
            " -e <blknum>               Extracts specific block\n",
            " -n <blknum>               Stop reading at specified block\n",
            " -a                        Perform monochromatic analysis\n",
            " -h or -?                  This help text\n"
          };

main(argc,argv)
        int argc;
        char *argv[];
{
        int i,j,k;
        FILE *infile;
        FILE *specfile; /* Spectral Data File */
        FILE *logfile;  /* Analysis log file */
        FILE *display;  /* Display pipe */

        long nobins=0;  /* Number of bins) */
        float *readbuf=NULL;  /* Read Buffer */
        float ftmp;
        long basesec = 0;
        float basetime = 0.0;

        char Infile[40];        /* Input File */
        char Outfile[40];       /* Output File */

        struct SpecAnalResults results;

        int min_f1,max_f1;

        float min_p1,max_p1;

        float min_snr, max_snr;

        int errflag=0;

        float binres;   /* Bin Resolution */

        int outblk;     /* Block number to dump spectrum to spec.dat" */

        int lastblk;

        int analflag;   /* Set to 1 to analyze the data */
        int preview ;  /* Preview flag, set to 1 */
        int plotid;       /* Process ID of specplot */

/** Output the Copyright stuff **/

        fprintf(stdout,"\nrdtape, Copyright Valley Technologies 1994, ver 1.1\n");

/** Setup Defaults **/

        strcpy(Infile,"/dev/rst1");
        strcpy(Outfile,"spec.dat");
        lastblk = -1;                   /* Process All BLocks */
        analflag = 0;                   /* Don't perform analysis */
        preview = 0;                    /* Don't preview */
        outblk = 0;                     /* No Putput */

/*** Parse the input line looking for switches **/


        for(i=1; i<argc; i++)
        {
          for(j=0; j<NUMCMDS; j++)   /* Parse the command */
          {
               if(strcmp(argv[i],flags[j]) == 0)break;
          }

          switch(j)
          {
                case 0:          /* -f Input file */
                   strcpy(Infile,argv[i+1]);
```

```
                i++;
                break;

            case 1:          /* -o Output file */
                strcpy(Outfile,argv[i+1]);
                i++;
                break;

            case 2:          /* -a Perform Analysis*/
                 analflag = 1;
                 logfile = fopen("anal.log","w");
                 break;

            case 3:          /* -n   Last Block to process*/
                sscanf(argv[i+1],"%d",&lastblk);
                i++;
                break;

            case 4:          /* -p  Previews the data with"specplot" */
                preview = 1;
                break;

            case 5:          /* -e  Extract specified block */
                sscanf(argv[i+1],"%d",&outblk);
                i++;
                break;

            case 6:
            case 7:
                for(i=0; i<8; i++) {fprintf(stdout,"%s",help[i]);}
                exit(0);
                break;


            default:
                fprintf(stderr,"Invalid switch\n");
                exit(0);
        }
      }

/** If preview data is enabled, Open the PIPE and fork the process */

      if(preview)
      {

         if((plotid = fork( )) == 0)    /* Launch Plotting Program */
         {
              plotid = execlp("specplot","specplot",NULL);
              if(plotid == -1)
                 perror("rdtape: ");
         }

     /** Open the spectral plot pipe */

         display = fopen("SPECPIPE","wb");
         if(display == NULL)
          {
              printf("\nrdtape: Error Opening SPECPIPE\n");
              exit(errno);
          }
      }



/** Open the Tape Device ***/

      infile = fopen(Infile,"rb");
      if(infile == NULL)
         {fprintf("Error Opening Input\n"); exit(0);}


      if (outblk != 0)
      {
```

```
                specfile = fopen(Outfile,"w");
                if(specfile == NULL)
                {
                        printf("\n Error Opening Output File\n");
                        exit(0);
                }
        }

 /*********************************************************/
 /****                                                 ***/
 /****     Main processing loop                        ***/
 /****                                                 ***/
 /****   Do until an error or the last block is read   ***/
 /****                                                 ***/
 /*********************************************************/

       errflag = 0;
       while((errflag >=0) && (RecHdr.blknum != lastblk))
       {

/** Read The Record Header **/

          j = fread(&RecHdr.sync[0],sizeof(RecHdr),1,infile);
           if(j != 1)
               printf("Error Reading Header on Tape %d\n",j);


/** Check for the Sync Words **/

           for(i=0; i<4; i++)
           {
                if(RecHdr.sync[i] != (0xa5a5001+i))
                {
                      printf(" Out of Sync %8.8x %d\n",RecHdr.sync[i],i);
                      errflag = -2;
                      break;
                }
           }

          if(preview)  /* Send to display program */
          {
                block_write(fileno(display),
                            &RecHdr.sync[0],
                            sizeof(RecHdr) );
          }



/* Set the base time of the operation */

          if(basesec == 0)
          {
              basesec = RecHdr.tp.tv_sec;
          }


/** Print out some Header If not in Preview mode ***/

        if(!preview)
        {
          ftmp = (float)(RecHdr.tp.tv_sec - basesec) + ((float)RecHdr.tp.tv_usec)*1.0e

          printf("BN: %6.6d  TM: %10.4f\n",RecHdr.blknum,ftmp);
        }

/** Get the FFT Size and determine how much data to read */

           k = RecHdr.fft_size/2; /* Half Spectrum */

/** Allocate Memory If Necessary ***/

           if(k != nobins)
           {
              if(readbuf != NULL) free(readbuf);
```

```
              nobins = k;

              readbuf = (float *)malloc(nobins*4);


              if(readbuf == NULL)
              {
                printf("Error Allocating Read Buffer \n");
                errflag = -1;
                break;
              }
            }

/** Read a Full Buffer **/

            for(i=0; i<4; i++)
            {
              j = fread((char *)&readbuf[nobins/4*i], (int)4 ,nobins/4, infile);
              if(j != nobins/4)
              {
                 printf("Error Reading Data on Tape %d\n",j);
                 errflag = -1;
                 break;
              }

              if(preview)
              {
                 block_write(fileno(display),
                             (char *)&readbuf[nobins/4*i],
                              nobins);

              }


            }

/* This is the block to output, make it so */

            if(RecHdr.blknum == outblk )
            {
                 dump_header(RecHdr);
                  for(j=0; j<nobins; j++)
                  {
                     fprintf(specfile, "%f\n",readbuf[j]);

                  }
                  fclose(specfile);
            }

#ifdef TEST_DATA

         /** Make sure the data is correct ***/

         for(j=0; j<nobins; j++)
         {
              if( (float)j != readbuf[j])
              {
                 printf("Data Error at %d :: %f\n",j, readbuf[j]);
                 break;
              }
         }
#endif


/******  Analyze the Spectrum *****/

          if(analflag)
          {
            AnalSpec(readbuf,nobins,&results);

/*** If First block set min and max ***/

            if(RecHdr.blknum == 1)
            {
```

```
            max_f1 = min_f1 = results.f1;
            max_p1 = min_p1 = results.p1;
            max_snr = min_snr = results.snr;
          }


/*** Rcord Min and Max values and write out a log of the anaysis **/

        if(results.f1 < min_f1 ) min_f1 = results.f1;
        if(results.f1 > max_f1 ) max_f1 = results.f1;

        if(results.p1 < min_p1 ) min_p1 = results.p1;
        if(results.p1 > max_p1 ) max_p1 = results.p1;

        if(results.snr < min_snr ) min_snr = results.snr;
        if(results.snr > max_snr ) max_snr = results.snr;

        fprintf(logfile,"%d  %d @ %f  %d @ %f  p: %f np: %f  snr: %f\n", RecHdr.blknum
                                        results.f1,
                                        10.0*log10(results.p1+1.0e-12),
                                        results.f2,
                                        10*log10(results.p2+1.0e-12),
                                        10*log10(results.ptot+1.0e-12),
                                        10*log10(results.np+1.0e-12),
                                        10*log10(results.snr+1.0e-12) );


        } /* End of while loop, End of processing **/

      }
/** Print out the min and max values **/

        if(analflag)
        {
          binres = (float)(RecHdr.fs)/(float)RecHdr.fft_size;
          printf(" Fs = %d, FFT = %d\n",RecHdr.fs, RecHdr.fft_size);

          printf(" Min F1 = %f\n",((float)min_f1 * binres));
          printf(" Max F1 = %f\n",((float)max_f1 * binres));
          printf(" Min P1 = %f\n", 10*log10(min_p1+1e-12) );
          printf(" Max P1 = %f\n", 10*log10(max_p1+1e-12) );

          printf(" Min SNR = %f\n",10*log10(min_snr+1e-12));
          printf(" Max SNR = %f\n",10*log10(max_snr+1e-12));

          fprintf(logfile," Min F1 = %f Mhz\n",((float)min_f1 * binres));
          fprintf(logfile," Max F1 = %f Mhz\n",((float)max_f1 * binres));
          fprintf(logfile," Min P1 = %f\n", 10*log10(min_p1+1e-12));
          fprintf(logfile," Max P1 = %f\n", 10*log10(max_p1+1e-12));

          fprintf(logfile," Min SNR = %f\n",10*log10(min_snr+1e-12));
          fprintf(logfile," Max SNR = %f\n",10*log10(max_snr+1e-12));

          fclose(logfile);

       } /* end of if analflag */


/** Close up and exit **/

        if(outblk != 0) fclose(specfile);

        fclose(infile);
        if(preview) kill(plotid,SIGKILL);  /* Terminate plotting prog */

        return;

} /* End of Main */

/*
 ************************************************************************
 *                                                                      *
 *  Function Name: AnalSpec                                             *
```

```
*                                                                              *
*  Description:  This routine will analyze a magnitude spectrum containing    *
*    a monotonic signal. It will find the following items in the spectrum      *
*                                                                              *
*           Freq of largest signal                                             *
*           Power Level of Largest +/- 3 bins                                  *
*           Frequency of 2nd largest signal  (spur performance )               *
*           Power of 2nd largest signal                                        *
*           Total power within the spectrum                                    *
*                                                                              *
*                                                                              *
*  Syntax:                                                                     *
*                                                                              *
*  Parameters:                                                                 *
*                                                                              *
*  Returns:                                                                    *
*                                                                              *
********************************************************************************
*/

void AnalSpec(buf,nobins,results)

        float buf[];     /* Buffer of spectrum */
        int nobins;      /* Number of Bins */

        struct SpecAnalResults *results;

{
#define DC_EXCLUDE 10    /* +/- Number of bins around DC
                            to exclude from processing */

#define F1_EXCLUDE 20    /* +/- Number of bins around the largest signal
                            to exclude in search processing */

        int f1_bin, f2_bin;  /* Bin Numbers for two prevelent frequencies */
        int i;

 /*** Scan Spectrum looking for largest bin
      ( Skip DC, 1st 10 bins  Also calculate total power ***/

        f1_bin = 0;
        results->ptot = 0.0;
        results->p1 = 0;

        for(i=DC_EXCLUDE; i<nobins; i++)
        {
            if(buf[i] > results->p1)
            {
               results->p1 = buf[i];
               f1_bin = i ;
            }
            results->ptot += (buf[i]*buf[i]);
        }


 /*** Now Scan looking for 2nd largest, Skip +/- 20 bins aroung f1_bin ***/

        f2_bin = 0;
        results->p2 = 0;

        for(i=DC_EXCLUDE; i<nobins; i++)
        {
                if( (i > (f1_bin-F1_EXCLUDE) ) && (i < (f1_bin+F1_EXCLUDE))) continue;

                if(buf[i] > results->p2)
                {
                   results->p2 = buf[i];
                   f2_bin = i;
                }
        }

 /*** Calculate Signal Power and spur power  using +/- 100 bins **/

        results->p1 = 0.0;
        results->p2 = 0.0;
```

```
        for(i=0; i<100; i++)
        {
                results->p1 += (buf[f1_bin-50+i]*buf[f1_bin-50+i]);
                results->p2 += (buf[f2_bin-50+i]*buf[f2_bin-50+i]);
        }

/*** Calculate SNR */

        results->np = results->ptot - results->p1;     /* Noise = total - signal */

        results->snr =  results->p1 / results->np ;    /* Signal divided by noise */

        results->f1 = f1_bin;
        results->f2 = f2_bin;

        return;
} /* End of AnalSpec */

/*
 ***************************************************************************
 *                                                                         *
 *  Function Name: block_write                                             *
 *                                                                         *
 *  Description: Performs block writes on a file descriptor                *
 *                                                                         *
 *  Syntax: int block_write(fd,buffer,numbytes)                            *
 *                                                                         *
 *  Parameters:                                                            *
 *              int fd;         File Descriptor                            *
 *              char *buffer;   Buffer to write to                         *
 *              int numbytes;   Total number of bytes to write             *
 *                                                                         *
 *  Returns: 0 or errno                                                    *
 *                                                                         *
 ***************************************************************************
 */


int block_write(fd,buffer,numbytes)

        int fd;         /* File Descriptor   */
        char *buffer;   /* Buffer to write from  */
        int numbytes;   /* Total number of bytes to write */


{
        int blksize = 4096;     /* Block size to write */
        register int j=0, k;

        while(numbytes > 0)
        {
          if(numbytes < blksize) blksize = numbytes;

          k = write(fd,&buffer[j],blksize);
          if( k != blksize)
          {
              perror("Block_write: ");
              return(errno);
          }
          j+=k;
          numbytes -=k;

        }
        return(0);
}

/*************************************************************/
/**                                                         **/
/**  dump_header                                            **/
/**                                                         **/
/**  dump the header to stdout                              **/
/**                                                         **/
/*************************************************************/
```

```
int dump_header(RecHdr)
        struct RecordHeader *RecHdr;
{

        fprintf(stderr,"Blocknum:      %d\n",RecHdr->blknum);
        fprintf(stderr,"fs:            %d\n",RecHdr->fs);
        fprintf(stderr,"Window Size:   %d\n",RecHdr->winsize);
        fprintf(stderr,"#fixed ave:    %d\n",RecHdr->ave_int);
        fprintf(stderr,"#float ave:    %d\n",RecHdr->ave_float);
        fprintf(stderr,"fft size:      %d\n",RecHdr->fft_size);
        fprintf(stderr,
           "station time  %1.1x%2.2x %2.2x:%2.2x:%2.2x\n",
                          RecHdr->station_time[0],
                          RecHdr->station_time[1],
                          RecHdr->station_time[2],
                          RecHdr->station_time[3],
                          RecHdr->station_time[4]);


        fprintf(stderr,"Source Flag:     %d\n",RecHdr->source_flag);
        fprintf(stderr,"# On Source:     %d\n",RecHdr->onsource);
        fprintf(stderr,"# Off Source:    %d\n",RecHdr->offsource);
        return(0);
}
```

```
/*
 ************************************************************************
 ************************************************************************
 **                                                                    **
 **    File Name: specplot.c                                           **
 **                                                                    **
 **    Description: This file contains the routines to generate        **
 **      a spectral monitor plot for the Jupiter project. It is        **
 **      written in Xview for a Sun Sparc Station running              **
 **      Solaris 1.1.1B. Data for display is received over             **
 **      a named pipe 'SPECPIPE"                                       **
 **                                                                    **
 **    Modules:                                                        **
 **                                                                    **
 **    Author:D. Kolesar, Valley Technologies                          **
 **                                                                    **
 **    Language: C                                                     **
 **                                                                    **
 **    Build Commands:                                                 **
 **                                                                    **
 **    Update Log                                                      **
 **                                                                    **
 **    Date      Initials   Description                                **
 **                                                                    **
 **                                                                    **
 ************************************************************************
 ************************************************************************
 */
/** include files **/

#include <stdio.h>
#include <sys/ioctl.h>
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/cms.h>
#include <xview/xv_xrect.h>
#include <xview/notify.h>
#include <math.h>

#define INT24 long
#include "jupiter.h"

/** local definitions **/

/* indices into color table renders specified colors. */

#define WHITE   0
#define RED     1
#define GREEN   2
#define BLUE    3
#define ORANGE  4
#define AQUA    5
#define PINK    6
#define BLACK   7


/** Local Variables **/


        GC gc;                  /* GC used for Xlib drawing */
        unsigned long *colors; /* the color table */

        int    numpts;                  /* Number of Points in Display */
        float  *specbuf=NULL;           /* Buffer containing the spectral data to display
        struct RecordHeader RecHdr;  /* Header record of spectral data coming from jcr
        char *recptr =(char *)&RecHdr;       /* Initialize byte pointer to record heade

        static int hdrcnt=0;            /* Current count of header bytes received */
        static int bufcnt=0;            /* Current count of specbuf bytes received */

        float Ymax,Ymin;               /* Min and max power in db to plot */

        int numpix,binspix;

        static Xv_window paintwindow;
```

```
    Canvas      canvas;
    Display     *display;
    XID         xid;


/** external functions and/or forward referencd funtions **/

        Notify_value    read_pipe( );
        void        canvas_repaint();

/** external data **/

/*
 ************************************************************************
 *                                                                      *
 *  Function Name: main                                                 *
 *                                                                      *
 *  Description:  Main Entry point for Specplot                         *
 *                                                                      *
 *  Syntax:                                                             *
 *                                                                      *
 *  Parameters:                                                         *
 *                                                                      *
 *  Returns:                                                            *
 *                                                                      *
 ************************************************************************
 */

main(argc, argv)
int     argc;
char    *argv[];
{

/*
 * initialize cms data to support colors specified above.  Assign
 * data to new cms -- use either static or dynamic cms depending
 * on -dynamic command line switch.
 */
    static char stipple_bits[] = {0xAA, 0xAA, 0x55, 0x55};

    static Xv_singlecolor cms_colors[] = {
        { 255, 255, 255 },  /* White */
        { 255, 0, 0 },      /* Red */
        { 0, 255, 0 },      /* Green */
        { 0, 0, 255 },      /* Blue */
        { 250, 130, 80 },   /* Orange */
        { 30, 230, 250 },   /* Aqua */
        { 230, 30, 250 },   /* Pink */
        { 255, 0, 255 }     /* Black */

    };

    Cms         cms;
    Frame       frame;
    XFontStruct *font;
    XGCValues   gc_val;
    Xv_cmsdata  cms_data;
    int         use_dynamic = FALSE;

        FILE *pipefp;   /* Input Pipe file pointer */

    int i;


 /*********************************************/


 /*
  * Initialize the xview server
  */
    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
    if (*++argv && !strcmp(*argv, "-dynamic"))
        use_dynamic = TRUE;

 /*
```

```
 * Define the frame
 */
    frame = xv_create(NULL,FRAME,
        FRAME_LABEL,     "Spectral Plot",
        XV_WIDTH,        540,
        XV_HEIGHT,       232,
        NULL);
 /*
  * Register the color stuff
  */
    cms = xv_create(NULL, CMS,
        CMS_SIZE,        7,
        CMS_TYPE,        use_dynamic? XV_DYNAMIC_CMS : XV_STATIC_CMS,
        CMS_COLORS,      cms_colors,
        NULL);
 /*
  * Define the canvas to draw the plot in
  */

    canvas = xv_create(frame, CANVAS,
        CANVAS_REPAINT_PROC,     canvas_repaint,
        CANVAS_X_PAINT_WINDOW,   TRUE,
        CANVAS_AUTO_SHRINK,      TRUE,
        CANVAS_AUTO_EXPAND,      TRUE,
        CANVAS_FIXED_IMAGE,      FALSE,
        CANVAS_WIDTH,            522,
        CANVAS_HEIGHT,           128,
        XV_VISUAL_CLASS,         PseudoColor,
        WIN_CMS,                 cms,
        NULL);



/*
 * Get the Display an Xview ID
 */

    display = (Display *)xv_get(frame, XV_DISPLAY);
    xid = (XID)xv_get(canvas_paint_window(canvas), XV_XID);

/* Define the Maximum and Minimum Peak in db */

        Ymax = 150.0;
        Ymin = 60;


/*
 * Load the Fonts
 */

    if (!(font = XLoadQueryFont(display, "fixed")))
    {
        puts("cannot load fixed font");
        exit(1);
    }

/*
 * Create and Initialize the Graphic Context
 */
    gc_val.font = font->fid;

    gc_val.stipple =
        XCreateBitmapFromData(display, xid, stipple_bits, 16, 2);

    gc = XCreateGC(display, xid, GCFont | GCStipple, &gc_val);

 /*
  * get the colormap from the canvas now that
  * the cms has been installed
  */

    colors = (unsigned long *)xv_get(canvas, WIN_X_COLOR_INDICES);


/*
```

```
 * Open the Input Pipe and Set up the input notify function
 */

        if( (pipefp = fopen("SPECPIPE","rb")) == NULL)
        {
                fprintf(stderr,"specplot: Error opening input pipe\n");
                exit(0);
        }

         (void) notify_set_input_func(1006, read_pipe,fileno(pipefp));

/* Start event loop */

    xv_main_loop(frame);
}
/*
 ************************************************************************
 *                                                                      *
 *  Function Name: canvas_repaint                                       *
 *                                                                      *
 *  Description: Repaint call-back proceedure. Called from xview server *
 *      when the display needs to be updated                            *
 *                                                                      *
 *                                                                      *
 ************************************************************************
 */
#define XOFFSET 20    /* Offset from height of X Axis */
#define YOFFSET 20    /* Offset from 0 for Y Axis */

void  canvas_repaint(jcanvas, jpw, jdisplay, jxid, xrects)
        Canvas          jcanvas;
        Xv_Window      jpw;
        Display         *jdisplay;
        Window          jxid;
        Xv_xrectlist    *xrects;
{
        int width,height;  /* Size of Window */
        int pix,i,j,k;
        float rtmp;
        int xpix,ypix,prevx,prevy;
        float scale;
        int offset = 10;
        int xorg,yorg; /* The origin of the plot in pixels */

        char string[40];

        float minp,maxp;

 /*** Get the  size of the canvas ***/

        width = (int)xv_get(jpw,XV_WIDTH);
        height = (int)xv_get(jpw,XV_HEIGHT);

 /** Set the width to the nearest power of 2 **/

      XClearWindow(jdisplay,jxid);

 /* Draw the Axis */

      xorg = YOFFSET;
      yorg = height - XOFFSET;

      XSetForeground(jdisplay, gc, colors[BLUE]);
      XDrawLine(jdisplay, xid, gc, xorg,yorg, width-10,yorg);
      XDrawLine(jdisplay, xid, gc, xorg,yorg, xorg, height-yorg);


      XSetForeground(jdisplay, gc, colors[RED]);  /* Color for plot */


         if(numpts == 0) return;
```

```
/* Calculate the number of points per pixel */

        numpix = width-(1.5*XOFFSET);
        binspix = numpts/numpix;

/** Calculate the data scale factor. If < 256 Fixed Point
    averages, then the data was prescaled by the difference so multiply
    the signal up.
    Otherwise divide by the number of floating point averages
 */
        if(RecHdr.ave_int < 256)
           scale = 256 - RecHdr.ave_int;
        else
           scale = 1.0/(float)RecHdr.ave_float;



/* Plot the peak within the range */

        j = 0;
        maxp = 0;
        minp = 500;

        for(pix=0; pix<numpix; pix++)
        {
          rtmp = 0.0;
          for(i=0; i<binspix; i++)
          {
             if(rtmp < specbuf[j])
               rtmp =  specbuf[j];
             j++;
          }

          rtmp = 20.0*log10((rtmp*scale) + 1.0e-12);

          if(rtmp < minp) minp = rtmp;
          if(rtmp > maxp) maxp = rtmp;

          rtmp = rtmp - Ymin;

          ypix = (height-YOFFSET)-(rtmp/(Ymax-Ymin) * (height-YOFFSET));
          xpix = pix+XOFFSET+1;
          if(pix > 0)
            XDrawLine(jdisplay, jxid, gc, prevx,prevy, xpix,ypix);
          prevx = xpix;
          prevy = ypix;

        }

/** Auto Scale for next plot, if min and max varied by too much */

        if(minp > (Ymin + 15)) Ymin = minp-10;
        if(minp < Ymin) Ymin = minp-10;

        if(maxp < (Ymax-15)) Ymax = maxp+10;
        if(maxp > Ymax) Ymax = maxp+10;

/* Put the Status Line */

     XSetForeground(jdisplay, gc, colors[BLUE]);  /* Color for Text */

        sprintf(string,"Block: %5.5d   %1x%2.2x %2.2x:%2.2x:%2.2x  ONSRC:",
                   RecHdr.blknum,
                   RecHdr.station_time[0],
                   RecHdr.station_time[1],
                   RecHdr.station_time[2],
                   RecHdr.station_time[3],
                   RecHdr.station_time[4]);

        XDrawString(display, xid, gc, width-250, height-6,string,35);

/* Indicate wether on or off source */

        if(RecHdr.source_flag == ONSOURCE)
             XSetForeground(jdisplay, gc, colors[GREEN]);  /* Color for ON */
```

```
        else
             XSetForeground(jdisplay, gc, colors[RED]);     /* Color for Off */

        XFillArc(display,xid,gc,width-40, height-14,10,10,0,360*64);

        XSetForeground(jdisplay, gc, colors[BLUE]);     /* Put a circle around */
        XDrawArc(display,xid,gc,width-40, height-14,11,11,0,360*64);


}

/*
 ************************************************************************
 *                                                                      *
 *  Function Name: read_pipe                                            *
 *                                                                      *
 *  Description: Xview notify proceedure to read from the input pipe    *
 *       NOTE: THis Proceedure is Written ONLY FOR Pipes. Using         *
 *       standard files may cause  problems. This proceedure will first read *
 *       the pipe looking for the header, Once a valid header is received *
 *       It will read the pipe untill the entire spectral data is received. *
 *       When the buffer is full, this routine will call canvas_repaint *
 *                                                                      *
 *                                                                      *
 *                                                                      *
 ************************************************************************
 */

Notify_value read_pipe(client,fd)

        Notify_client client;   /* Client Id, Not used */
        int fd;                 /* File ID */
{
        Xv_Window pw;    /* Paint Window */

        int bytes;       /* Number of bytes to read */
        int i,j;

 /*** Read the Header ***/


        while(hdrcnt < sizeof(RecHdr) )
        {
           i = ioctl(fd,FIONREAD, &bytes);    /* Get Number of bytes in pipe */
           if(bytes == 0) return NOTIFY_DONE;

           if(i== -1) return NOTIFY_DONE;

           if(bytes <= (sizeof(RecHdr) - hdrcnt))  /* Read all bytes as Header Bytes *
           {
               i = read(fd,&recptr[hdrcnt],bytes);
               hdrcnt += i;                          /* Increment Header Bytes */
           }                                        /* Read only the bytes necessary to
           else
           {
               j = sizeof(RecHdr) - hdrcnt ;
               i = read(fd,&recptr[hdrcnt],j) ;
               hdrcnt += i;
           }
           bufcnt = 0;


        }

  /*** Header has been read, check the sync flags  ***/


        for(i=1; i<=4; i++)
        {
          if(RecHdr.sync[i-1] != (0xa5a5000 + i))
          {
            printf("specplot:  Sync Error %x i\n",RecHdr.sync[i-1], i);
            exit(0);
```

```
          }
        }

/** If the Buffer has not been allocated , do it now **/

        numpts = RecHdr.fft_size/2;

        if(specbuf == NULL)
            specbuf = (float *)calloc(numpts, sizeof(float));

/*** Read the buffer, size = fft size /2 **/


        while(bufcnt < numpts)
        {
           i = ioctl(fd,FIONREAD, &bytes);       /* Get Number of bytes in pipe */
           if(bytes == 0) return NOTIFY_DONE;

           if(i== -1) return NOTIFY_DONE;

           if(bytes >= 4)                        /* Always read in multiple of 4 */
           {
             j = bytes/4 ;
             i = read(fd,&specbuf[bufcnt],j<<2);

             bufcnt += i>>2;
           }

        }

        hdrcnt = 0;    /* Set hdrcnt to zero to start reading the header */


/*** Now Call repaint routine to display the screen ***/

        pw = xv_get(canvas,CANVAS_NTH_PAINT_WINDOW,0,NULL);  /* Get pw parameter */

        canvas_repaint(canvas, pw, display, xid, NULL);     /* Repaint It */

        return NOTIFY_DONE;

}
```

```
/*
 ************************************************************************
 ************************************************************************
 **                                                                    **
 **    File Name:  avemag.c                                            **
 **                                                                    **
 **    Description:                                                    **
 **          This program is used to compute the average magnitude     **
 **          of a N point FFT received on the Type 1 Inputs            **
 **                                                                    **
 **    Modules: main - The main routine                                **
 **                                                                    **
 **    Author: D. Kolesar                                              **
 **                                                                    **
 **    Language: AT&T DSP32C                                           **
 **                                                                    **
 **    Build Commands:                                                 **
 **                                                                    **
 **    Update Log                                                      **
 **                                                                    **
 **    Date      Initials   Description                                **
 **    4/11/94   dmk         Creation                                  **
 **    4/20/94   alm         removed the CMAG function as the data     **
 **                          received to the UtraDSP-2 is already a    **
 **                          magnitude squared MSB's                   **
 ************************************************************************
 ************************************************************************
 */

/** include files **/
#include "lh9124.h"             /* DSP Definitions */
#include "lh9320.h"             /* Address Generator defintions */
#define CLANGUAGE 1             /* To enable C macros defined in ultradsp.h  */

#include "ultradsp.h"           /* Misc defs and macros */
#include "hostio.h"              /* To access stdio type functions using hostio */
#include "valleyio.h"             /* for upload and download data and file type */

#define INT24 int
#include "jupiter.h"

/* #define DEBUG 1 */

#define O_RDONLY        0
#define O_WRONLY        1
#define O_RDWR          2


/** local definitions **/

        unsigned int *streg = (unsigned int *)ST_REGISTER ;    /* Pointer to Status Reg
        unsigned int *dfreg = (unsigned int *)DF_REGISTER ;   /* Pointer to DF register
        unsigned int *fcreg = (unsigned int *)FC_REGISTER ;   /* Pointer to FC Register



/** external/Forward Referenecd functions **/

       extern void load_ag_int();
       void dmparam( );
       void fill_a( );

/** external data **/

       extern int dfdata;
       extern int fcdata;

/** Global Local Variabls */

        struct Operational_Parameters OpParam;
        struct Operational_Commands OpCmd;

        struct tagMemAddr memory_address;

        unsigned int *dfreg;                /* Pointer to Data Flow register */
```

```
void main()
{

        int i,j,m ;               /* Generic loop counters */
        register int itmp;       /* Temporary Integer */
        int rd_pipe;             /* Control Task Communication Pipe, setup and commands
        int wr_pipe;             /* Control Task Communication Pipe, averaged data */

        int vector_size;         /* Size of processing block in mag pairs */
        int num_buffers;         /* Number of fifo buffers needed to do an average */
        int float_blocks;        /* Number of floating blocks to average */
        int fixed_blocks;        /* Number of Fixed Point blocks to average */

        register int k ;                  /* A Fast Loop Counter */
        register int *Rptr, *Iptr;       /* Memory Pointers */
        register float *Rsum, *Isum;
        register int move_function;

/*
 *  Open the Pipes to the control task
 *  and Read the Operational Parameters
 */
#ifdef DEBUG
        fprintf(stderr,"Inside AVEMAG \n");
#endif

       rd_pipe = open("AVEPIPEWR",O_RDWR);  /* Jcntrl writes, this prog. reads */
       if(rd_pipe == NULL)
       {
          fprintf(stderr,"\nError Opening Read Pipe");
          exit(0);
       }


       wr_pipe = open("AVEPIPERD",O_RDWR);  /* Jcntrl reads, this prog. writes*/
       if(wr_pipe == NULL)
       {
          fprintf(stderr,"\nError Opening Write Pipe");
          exit(0);
       }


       i = read(rd_pipe,&OpParam.cmd,sizeof(OpParam));
       if(i != sizeof(OpParam))
       {
          fprintf(stderr,"\nError Reading Pipe");
          exit(0);
       }
#ifdef DEBUG
      dmparam( );
#endif


/*******************************************************/
/**                                                   **/
/**       Compute the averaging parameters            **/
/**                                                   **/
/*******************************************************/

       vector_size = OpParam.fft_size >> 2 ;      /* Number of magnitude pairs per bl

/** Calculate the number of 1K fifo buffers to read before a
    fixed point sum is performed */

       if(OpParam.fft_size >= 4096)
          num_buffers = vector_size >> 10 ; /* Fast Divide by 1024 */
       else
       {
          num_buffers = 1;              /* Process Data as though it was a 4K */
          /* vector_size = 1024; */
       }
       float_blocks = OpParam.ave_float;
```

```
        fixed_blocks = OpParam.ave_int;

/*** Reset The Output FIFOs ****/

         itmp = dfdata & 0x8fff;          /* Get sw copy of DF */
        *dfreg = itmp | 0x7000;           /* Reset Output FIFO */
        *dfreg = itmp;


/*******************************************************/
/**                                                   **/
/**       Main Processing Loop, Do Forever            **/
/**                                                   **/
/*******************************************************/

        while(1)
        {


/** Wait for command from Control Task to begin processing **/

        i = read(rd_pipe,&OpCmd.cmd,sizeof(OpCmd));

        if(OpCmd.cmd != RESTART_AVE) continue;

        for ( i = 0; i < float_blocks; i++ )   /* Floating Point Loop processing  */
        {


           for( j = 1; j <= fixed_blocks; j++)   /* Fixed Point Loop Processing */
           {

             /*******************************************************/
             /**   Move 1K Buffers from Input FIFO to Q memory     **/
             /**    adding with the partial sum in C               **/
             /*******************************************************/

             /** Reset Input FIFOs, and then setup the ags for data transfer from fifc

             itmp = dfdata & 0x8fff;              /* Get sw copy of DF */
             *dfreg = itmp | 0x6000;              /* Reset Input FIFO */
             *dfreg = itmp;

             /** Initialize Memory Addresses **/

             memory_address.Q = 0;
             memory_address.C = 0;
             memory_address.A = 0;


                load_ag_int(ADRLENGTHL,1024);        /* Set up the Length of move */

             for (k = 0; k < num_buffers; k++ )
             {
                load_ag_address(&memory_address);    /* Set next Starting Address */

#ifdef DEBUG
                fprintf(stderr,"B->Q %d\n",k);
#endif

                while( (*streg & 2) );   /** Wait for Input FIFO > Half Full, HF != 1

                if(j==1)
                   move_bufferBtoQ(0);      /* Move Data withoutAdd  */
                else
                   move_bufferBtoQ(CADD);   /* Move Buffer with Add */

                memory_address.Q += 1024;          /* Increment for next memory locatio
                memory_address.C += 1024;          /* Increment for next memory locatio
                memory_address.A += 1024;

             }
             /*******************************************************/
             /**   Move current partial to C                       **/
             /**    ( Unless it is the last vector )               **/
```

```
            /*******************************************************/

            if( j == fixed_blocks) break;  /* Go move it to A */

            memory_address.Q = 0;
            memory_address.C = 0;
            memory_address.A = 0;

            load_ag_address(&memory_address);          /* Set next Starting Address */

            load_ag_int(ADRLENGTHL,vector_size);        /* Set up the Length for move

            move_bufferQtoC(0);

#ifdef DEBUG
            fprintf(stderr,"Q->C %d\n",vector_size);
#endif
         }  /* end of Fixed Point j Loop */


         /*******************************************************/
         /**  Done with Fixed Loop, Move sum in Q to A         **/
         /**                                                   **/
         /*******************************************************/

            memory_address.Q = 0;
            memory_address.C = 0;
            memory_address.A = 0;

            load_ag_address(&memory_address);          /* Set next Starting Address */

            load_ag_int(ADRLENGTHL,vector_size);        /* Set up the Length for move
#ifdef DEBUG
               fprintf(stderr,"Q->A %d\n",vector_size);
#endif

            move_bufferQtoA(0);

         /*******************************************************/
         /**                                                   **/
         /**  Output Fixed Results to B Port                   **/
         /**                                                   **/
         /**                                                   **/
         /*******************************************************/

         /** Set Output Block Flag in df register **/

            dfdata = dfdata | 0x0100;

         /* Set the length of the move and init starting address */

            memory_address.A = 0;
            memory_address.Q = 0;
            memory_address.C = 0;
            load_ag_int(ADRLENGTHL,1024);


            for (k = 0; k < num_buffers; k++ )

            {
                load_ag_address(&memory_address);           /* Set the starting Addre

                while( !(*streg & 0x0001) ) ;     /* Wait for outfifo < HalfFull flag

#ifdef DEBUG
          fprintf(stderr,"A->B  %d\n",k);
#endif
                move_bufferAtoB(0);     /* move 1k block from A to output FIFO */

                memory_address.A += 1024;                /* Increment the A */

                dfdata = dfdata & 0xfeff;           /* Clear the Block Flag bit in th
            }

         /*******************************************************/
```

```
/**                                                          **/
/**  Perform floating point conversion and average.          **/
/**  The results of the average will be placed in the        **/
/**  upper half of RAMA.                                     **/
/**                                                          **/
/**                                                          **/
/**************************************************************/

      fcdata = fcdata | 2;  /* Set Misc Out */
      *fcreg = fcdata;

      access_RAMA( );

      Rptr =  (int *)REALRAM;
      Iptr = (int *)IMAGRAM;

      Rsum = (float *)(REALRAM + (64*4*1024) );
      Isum = (float *)(IMAGRAM + (64*4*1024) );

      k = vector_size;

  /* First Block: Just convert to DSP32C Floating Point and move */

      j = 0;
      if(i==0)
      {
        while(k-- > 0)
        {

          *(Rsum++) =  (float)(*Rptr++) ;
          *(Isum++) =  (float)(*Iptr++) ;
        }
      }

  /* Remaing Blocks: Convert to Float and sum with previous value */

      else
      {
        while(k-- > 0)
        {

          *(Rsum++) = *Rsum + (float)(*Rptr++);
          *(Isum++) = *Isum + (float)(*Iptr++);
        }
      }

      fcdata = fcdata & 0xfffd;  /* clear Misc Out */
      *fcreg = fcdata;


/**************************************************************/
/**                                                          **/
/**  If the FFT Size < 4k, Average the sub blocks            **/
/**                                                          **/
/**************************************************************/



} /** end of floating point average loop **/


/**************************************************************/
/**                                                          **/
/**  Fill the RAM With a floating point ramp for test        **/
/**                                                          **/
/**************************************************************/

#if(0)

      Rsum = (float *)(REALRAM + (64*4*1024) );
      Isum = (float *)(IMAGRAM + (64*4*1024) );

      k = vector_size;
```

```
                j = 0;
                while(k-- > 0)
                {
                   *(Rsum++) = (float)j++;
                   *(Isum++) = (float)j++;
                }
#endif

        /*******************************************************/
        /**                                                   **/
        /**  Write out the resultant averaged spectrum        **/
        /**                                                   **/
        /*******************************************************/

                Rsum = (float *)(REALRAM + (64*4*1024) );
                Isum = (float *)(IMAGRAM + (64*4*1024) );

                 dfdata = dfdata & 0xfffe; /* Turn on the LED */
                *dfreg = dfdata;


                itmp = vector_size << 2;    /* Number of bytes to write */

                k = write(wr_pipe,Rsum,itmp);

                if(k != itmp)
                {
                        fprintf(stderr,"\nError Writing Pipe");
                        exit(0);
                }

                k = write(wr_pipe,Isum,itmp);
                if(k != itmp)
                {
                        fprintf(stderr,"\nError Writing Pipe");
                        exit(0);
                }

                dfdata = dfdata | 1;     /* Turn off the LED */
                *dfreg = dfdata;

        } /* End of Infinit While Loop */


}
/*
 ***************************************************************************
 *                                                                         *
 *  Function Name: dmparam                                                 *
 *                                                                         *
 *  Description:                                                           *
 *                                                                         *
 *  Syntax:                                                                *
 *                                                                         *
 *  Parameters:                                                            *
 *                                                                         *
 *  Returns:                                                               *
 *                                                                         *
 ***************************************************************************
 */

void dmparam( )
{
        fprintf(stderr,"\nmag: cmd:         %8.8x",OpParam.cmd);
        fprintf(stderr,"\nmag: fs:          %d",OpParam.fs);
        fprintf(stderr,"\nmag: winsize:     %d",OpParam.winsize);
        fprintf(stderr,"\nmag: fftsize:     %d",OpParam.fft_size);
        fprintf(stderr,"\nmag: aveint:      %d",OpParam.ave_int);
        fprintf(stderr,"\nmag: avefloat:    %d",OpParam.ave_float);
        fprintf(stderr,"\nmag: fft:         %s",OpParam.fft_file);
        fprintf(stderr,"\nmag: Window:      %s",OpParam.win_file);
        fprintf(stderr,"\n");
        return;
}
```

```
void fill_a( )
{
        register int i;
        register int *Rptr, *Iptr;
        register int offset;
        offset = 0x00ffff;

                access_RAMA( );

                Rptr =  (int *)REALRAM;
                Iptr = (int *)IMAGRAM;

                for( i=0; i<4096; i+=2)
                {
                    *Rptr++ = 0x000000;
                    *Iptr++ = 0x000000;
                }

                Rptr =  (int *)(REALRAM + 1000);
                Iptr =  (int *)(IMAGRAM + 1000);

                *Rptr = 0x7FFF;
                *Iptr = 0x7FFF;
                return;
}
```

```
/*
 ********************************************************************************
 ********************************************************************************
 **                                                                            **
 **    File Name  : mainfft.c                                                  **
 **                                                                            **
 **  Description: General FFT program to perform a real FFT on input data.     **
 **               This programd  supports FFTs from 2K real to 256K real only  **
 **               Input Data is from a TYPE 1.                                 **
 **               Output Data is sent to TYPE 1 or written to 'forward.dat'    **
 **               The N pt Real data is processed as N/2 complex FFT followed  **
 **               by a recombination pass. The resulting spectrum is positive  **
 **               spectrum of only N/2 points.                                 **
 **               This program assumes  only 1 quadrant of the twiddles are stored**
 **                                                                            **
 **                                                                            **
 **       This file contains the source code for all FFTs from 2K real (1K complex)*
 **       to 256K real (128K complex). To build the object code for a specific **
 **       FFT size, the define FFTCxxxK must be passed to the compiler          **
 **                                                                            **
 **                                                                            **
 **                                                                            **
 **    Modules    :  main - Main routine                                       **
 **                                                                            **
 **    Author     : Anil M., Valley Technologies Inc.                          **
 **                                                                            **
 **    Language   : AT&T 'C'                                                   **
 **                                                                            **
 **    Build Commands: d3cc -c -DFFTxxxKT=n -Q -T -I<include_path> mainfft.c    **
 **                    ( where n = 1,2,4..128)                                 **
 **                                                                            **
 **                                                                            **
 **    Update Log                                                              **
 **                                                                            **
 **    Date      Initials   Description                                        **
 **                                                                            **
 **    4/6/94    alm        Creation()                                         **
 **    4/24/94   alm        added a window function                            **
 ********************************************************************************
 ********************************************************************************
 */

/** include files **/

#include "lh9124.h"             /* DSP Definitions */
#include "lh9320.h"             /* Address Generator defintions */
#define CLANGUAGE 1             /* To enable C macros defined in ultradsp.h  */

#include "ultradsp.h"           /* Misc defs and macros */
#include "hostio.h"              /* To access stdio type functions using hostio */
#include "valleyio.h"                   /* for upload and download data and file type

#include <io.asm>               /* 32C Low level functions */
#include <libap.h>              /* Trig Functions */


#define OUTB  1           /* Set to Nonzero to output results continously to B port */
#define DEBUG 1           /* Set to non-zero to output debug messges */

/*** Define constants macros depending on which FFt is to be compiled in */


#ifdef _FFT1K
#define K    512                /* size of real FFT */
#define LOOP 1                  /* number of times this program need to get 1k blocks
#define ODDPASSES 1             /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT2K
#define  K     1024
#define  LOOP  1
#define ODDPASSES 1             /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT4K
```

```
#define  K       2048
#define  LOOP    2
#define ODDPASSES 0                /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT8K
#define     K      4096
#define     LOOP   4
#define ODDPASSES 1                /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT16K
#define  K       8192
#define  LOOP    8
#define ODDPASSES 0                /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT32K
#define  K       16384
#define  LOOP    16
#define ODDPASSES 0                /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT64K
#define  K       32768
#define  LOOP    32
#define ODDPASSES 1                /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT128K
#define  K       65536
#define  LOOP    64
#define ODDPASSES 0                /* ODD PASS FFTs End In RAM Q */
#endif

#ifdef _FFT256K
#define  K       131072
#define  LOOP    128
#define ODDPASSES 1                /* ODD PASS FFTs End In RAM Q */
#endif


/** local definitions **/
        static struct tagMemAddr AddrGen;       /* Address generator Structure */

        unsigned int *streg = (unsigned int *)ST_REGISTER ;    /* Pointer to Status Reg
        unsigned int *dfreg = (unsigned int *)DF_REGISTER ;   /* Pointer to DF register
        unsigned int *fcreg = (unsigned int *)FC_REGISTER ;   /* Pointer to FC Register

        int scale;     /* Scale factor to use during passes */
        unsigned int dsfo;      /* Data scale factor output */
        unsigned int blkcnt;    /* FFT Block Counter */

        unsigned int bexp;       /* Block Exponent */
        unsigned int nexp;       /* Normalization Exponent */

/** external functions **/

/** external data **/

    extern int dfdata;
    extern int fcdata;


#if(!OUTB)
        long *realptr;
        long *imagptr;
        FILE *fp;
        float realtmp1,realtmp2;
#endif
/*
 ***************************************************************************
 *  Function Name: main                                                    *
 *  Description:  main routine                                             *
 *  Syntax: int main( )
```

```
 *  Parameters: none                                                        *
 *  Returns: none                                                          *
 ***************************************************************************
 */

void main( )
{

          register int i,j;
          int data_in_q;       /* Flag indicating, after the recomb. pass, if the
                                  resultant data is in RAM Q */

/*
 ***********************************************************************
 *                                                                     *
 *   Set Coefficient Complement Mode to use the signals from           *
 *   the AG and to use only quarter twiddles. (For 256k real FFT we    *
 *   need only (64k+1) coeff's, thus reducing in twiddle memory size)  *
 *                                                                     *
 ***********************************************************************
 */
    i = dfdata;                    /* get the software copy of Data Flow register */
    dfdata = i | 0x0006;           /* set CCI & CCR bits */
    set_ag_byte(AGMODE,0x081808); /* set the AG mode register to use only 1/4 twiddles

/** Set Apparent Size of the Twiddle Memory **/
    load_ag_int(AGMEMSIZEL,(256*1024)) ;       /* length for a 256k Real FFT */

/** Set the Increment register to 1 in all three AG's  **/
    load_ag_int(ADRINCL, 0x1); /* init the address increment reg */

/** Set the Decrement register to 1 in all three AG's  **/
    load_ag_int(ADRDECL, 0x1);   /* init the address increment reg */

/** Enable A/D Fifos on ULTRA-ADC board **/

        fcdata = fcdata | 2;
        *fcreg = fcdata;

/*** Main FFT Processing Loop ***/

/** Reset the Output FIFOS **/

         i = dfdata & 0x8fff;           /* Get sw copy of DF */
        *dfreg = i | 0x7000;            /* Reset Output FIFO */
        *dfreg = i;

while(blkcnt >= 0 )      /* Process Forever  */
{
/** Set Starting Address **/

        AddrGen.Q = 0;             /* Initial Address of data*/
        AddrGen.A = 0;             /* Initial Address of data*/


#ifdef _FFT256K

        AddrGen.C = 65536;         /* Initial Address of window coefficients*/
#else
        AddrGen.C = 65537;         /* Initial Address of window coefficients*/

#endif



        load_ag_int(NUMSAMPL,1024);                /* Set 1k  Moves */
        load_ag_int(ADRLENGTHL,1024);      /* length of buffer to move */

/** Reset the input FIFOS **/

        i = dfdata & 0x8fff;            /* Get sw copy of DF */
        *dfreg = i | 0x6000;                /* Reset Input FIFO */
        *dfreg = i;

/** Initialize scale factor and max scale factor */
```

```
        scale = 0;
        dsfo = 0;

/*************************************************************************
 *                                                                       *
 *  INPUT SECTION                                                        *
 *                                                                       *
 *    Get 1k-blocks from input FIFO(TYPE1), Move to RAM A                *
 *    Window the data while moving it from the TYPE1                     *
 *                                                                       *
 *    The window is stored in the upper 64K-1 of the C memory.           *
 *    (because the 1/4 twiddles takes the lower 64k+1       ),           *
 *                                                                       *
 *    For Window sizes = 128K or 256K, the program needs to              *
 *    fudging the window, by starting the address for the window         *
 *    at  the last twiddle factor. The result of this operation          *
 *    is that the first and last two real data points will have been     *
 *    Multiplied by 0 -j1, the 270 deg twiddle. To compensate for this   *
 *    miss multiply, the first and last data points will be zeroed       *
 *    prior to perfroming the FFT. This should cause no  significant loss*
 *    of perfromance                                                     *
 *                                                                       *
 *************************************************************************/

        for(i=0; i<LOOP; i++)
        {
             load_ag_address(&AddrGen);       /** Set the address for next VMUL pass**/

             while( *streg & 2) ;             /* wait for infifo half full flag */

/*** Multiply by the Window Function ***/

#ifdef _FFT128K
             if(i<32)                  /* Increment through the window Function*/
#else
             if(i<64)                  /* Increment through the window Function*/
#endif
             {
                AddrGen.C += 1024;
                AddrGen.A += 1024;
                move_bufferBtoA(VMUL | 2);   /* move 1k block into A mem., set MISC o
             }
             else          /* Decrement through the window function */
             {

                AddrGen.Q = AddrGen.Q | 0x80000;          /* Set A19 bit for B Bort

                load_ag_address(&AddrGen);                /* and load the starting a

                AddrGen.C -= 1024;                         /* decrement the C mem ad
                AddrGen.A += 1024;

                set_ag_byte(LATENCY00, 0x178505);           /* VMUL pass using RBWA
                set_ag3(AG_PROG_MEM00, AGINC, MODDEC, AGINC);  /* set the reg */
                set_fc( VMUL | 2);                          /* and this function */
                set_df( RBWA|AGQ|DSPA );
                wait_till_done();

                AddrGen.Q = AddrGen.Q & 0xf7ffff;        /** Restore the A19th bit  *

             }


 /** Monitor the Input for the largest Data Scale Factor (DSFO)
     for subsequent scaling during the FFT Passes **/

             dsfo = (*streg  & 0xe000) ;       /* get DSFO's and select DSISEL bit */
             if(dsfo > scale )  scale = dsfo ;

  }              /* End of block moves */


 #ifdef _FFT256K
          access_RAMA( );
```

```
        *(int *)REALRAM = 0;
        *(int *)IMAGRAM = 0;
        *(int *)(REALRAM+((128*1024 -1)*4)) = 0;
        *(int *)(IMAGRAM+((128*1024 -1)*4)) = 0;
#endif

  /*****************************************************************
   *                                                               *
   *   FFT pass: Performs the K point complex FFT, on real data    *
   *                                                               *
   *****************************************************************/

       load_ag_int(NUMSAMPL,K); /* set the number of samples N to size of complex FFT

#ifdef _FFT1K
          fft512b(0,scale);     /* Perform 512 point complex FFT. Results end in RAMQ *
#endif

#ifdef _FFT2K
          fft1k(0,scale);       /* Perform 1K complex FFT. Results end in RAMQ */
#endif

#ifdef _FFT4K
          fft2k(0,scale);       /* Perform 2K complex FFT. Results end in RAMA */
#endif

#ifdef _FFT8K
          fft4k(0,scale);       /* Perform 4K complex FFT. Results end in RAMQ */
#endif

#ifdef _FFT16K
          fft8k(0,scale);       /* Perform 8K complex FFT. Results end in RAMA */
#endif

#ifdef _FFT32K
          fft16k(0,scale);      /* Perform 16K complex FFT. Results end in RAMA */
#endif

#ifdef _FFT64K
          fft32k(0,scale);      /* Perform 32K complex FFT. Results end in RAMQ */
#endif

#ifdef _FFT128K
          fft64k(0,scale);      /* Perform 64K complex FFT. Results end in RAMA */
#endif

#ifdef _FFT256K
          fft128k(0,scale);     /* Perform 128K complex FFT. Results end in RAMQ */
#endif

 /*****************************************************************
  *                                                               *
  *    Recombination PASS:       From A to Q, with auto-scaling   *
  *                              Extracts the real points.        *
  *                                                               *
  *****************************************************************/

 #if(ODDPASSES)       /*  FFT Results is in Q, Perform Recomb Q-> A   */

        set_ag_byte(LATENCY00, 0x178585);              /* MOVE pass using RQWA */
        set_ag3(AG_PROG_MEM00, BFCTUS, BFCTT, BFCTL);  /* set the reg */
        set_fca(BFCT | 2);
```

```
        set_df( RQWA|AGQ|DSPA);
        wait_till_done();
        data_in_q = 0;          /* Clear Data In Q Flag */

#else           /*   FFT Results in A, Perform Recomb A -> Q     */

        set_ag_byte(LATENCY00, 0x848518);           /* MOVE pass using RAWQ */
        set_ag3(AG_PROG_MEM00, BFCTL, BFCTT, BFCTUS);  /* set the reg */
        set_fca(BFCT | 2);
        set_df( RAWQ|AGA|DSPA);
        wait_till_done();
        data_in_q = 1;          /* Set Data In Q Flag */

#endif

        bexp =  ((*streg >> 2)  & 0x003f);      /* Get Block Exponent */

        if(blkcnt == 0)    /* Reset the normalization constant */
        {
          nexp = bexp + 2;
        }

/*****************************************************************************
 *                                                                           *
 *    SCALE PASS:                                                            *
 *                                                                           *
 *       Scaling is performed by generating a normalization                  *
 *       exponent 'nexp', and normalizing each block to this value           *
 *       The normalization exponent is calculated by taking the              *
 *       block exponent of the first FFT during a average group              *
 *       and adding 2. Each succesive block will then be divided             *
 *       by an amount = 2**(nexp - bexp). If (bexp > nexp), an overflow      *
 *       condition has occured and the output data will be be zeroed.        *
 *                                                                           *
 *       Recalculation of a new normalization constant will be performed under *
 *       control of the Averager Board when it assertes MISCIN. Scale values *
 *       up to a value of 7 may be performed during a single pass by utilizing *
 *       the DSFI bits in the FC register. A Scale value of 0 to 7 will be   *
 *       performed during the OUTPUT Pass. Any addtional scale passes        *
 *       will be performed prior to output.                                  *
 *                                                                           *
 *                                                                           *
 *****************************************************************************/


 /** Calculate the scale factor **/

        scale = nexp - bexp ;

        while((scale > 7) && (scale > 0) )
        {
           AddrGen.A = 0;
           AddrGen.Q = 0;
           AddrGen.C = 0;
           load_ag_int(ADRLENGTHL,K );
           load_ag_address(&AddrGen);        /* Set the starting Address */

          /* Move the data,
             Set the move function to
                                a MOVD function code
                                with MISCOUT  set
                                User Defined scale factor of 7
           */

           if(data_in_q)
           {
             move_bufferQtoA( (7<<13) | MOVD | 2 | 1 ) ;
             data_in_q = 0;
             scale = scale - 7;
           }
           else
           {
             move_bufferAtoQ( (7<<13) | MOVD | 2 | 1 ) ;
             data_in_q = 1 ;
```

```
            scale = scale - 7 ;
          }

        }

/*****************************************************************
 *                                                               *
 * OUTPUT PASS:                                                  *
 *           Send 1k-blocks to output FIFO                       *
 *                                                               *
 *****************************************************************/

#if(OUTB)

/** Set Output Block Flag in df register & turn on the LED **/

        dfdata = (dfdata | 0x0100) & 0xfffe;

/** Check for neg. scale factor indicating an overflow, and zero the data
    by setting the df register to zero the input data */


        if( scale < 0  )
        {
            dfdata = (dfdata & 0x8ffe) | 0x4000;

            fprintf(stderr,"FFT Overflow\n");
            scale = 0;
        }

/* Set the length of the move and init starting address */

        AddrGen.A = 0;
        AddrGen.Q = 0;
        AddrGen.C = 0;
        load_ag_int(ADRLENGTHL,1024);

        scale  = ( scale << 13) | MOVD | 2 | 1 ;   /* Set Move Function */
        for(i=0;i<LOOP;i++)
        {
            load_ag_address(&AddrGen);        /* Set the starting Address */

            while( !(*streg & 0x0001) ) ;     /* Wait for outfifo < HalfFull flag */

            if(data_in_q)
            {
              move_bufferQtoB(scale);         /* move 1k block from Q to output FIFO */
            }
            else
            {
              move_bufferAtoB(scale);         /* move 1k block from A to output FIFO */
            }

            AddrGen.A += 1024;                /* Increment the A */
            AddrGen.Q += 1024;                /*    and Q memory for next pass */

            dfdata = dfdata & 0xfeff;         /* Clear the Block Flag bit in the df reg
        }

        dfdata = (dfdata & 0x8fff) | 0x0001;    /* Clear the DZI function in case it wa
                                                /* & turn off the LED */

#endif

  /*****************************************************************
   *       Saving results of FFT to file 'forward.dat'             *
   *       #ifdefine OUTB = 0 (OPTIONAL,GENERALLY USED FOR TESTING) *
   *****************************************************************/

#if(!OUTB)

        if(blkcnt==10)
        {

         if(data_in_q)
```

```
        {
          AddrGen.A = 0;
          AddrGen.Q = 0;
          AddrGen.C = 0;
          load_ag_int(ADRLENGTHL,K );
          load_ag_address(&AddrGen);          /* Set the starting Address */

          move_bufferQtoA(MOVD | 2);
        }
        access_RAMA( );
        fp = fopen("forward.dat","w");
        realptr = (long *)REALRAM;
        imagptr = (long *)IMAGRAM;
        for(i=0; i<K; i++)
        {
           realtmp1 = (float) *realptr++;
           realtmp2 = (float) *imagptr++;
           /* realtmp1 = realtmp1*realtmp1 + realtmp2*realtmp2; */
           fprintf(fp,"%d %f %f\n",i,realtmp1,realtmp2 );
        }
       exit(0);
       }
#endif

        blkcnt++;    /* Increment Block Count */

   }   /* End of Infinite While loop */

    exit(0);
}              /***** END OF MAIN */
```

```
/*
 ************************************************************************************
 ************************************************************************************
 **                                                                                **
 **    File Name  : italy.c                                                        **
 **                                                                                **
 **  Description: General FFT program to perform a real FFT on input data          **
 **               received from input FIFO. Results will be sent to output FIFO    **
 **               The N pt Real data is processed as N/2 complex FFT followed      **
 **               by a recombination pass. The resulting spectrum is positive      **
 **               spectrum of only N/2 points.                                     **
 **               This program downloads the 64k+1 twiddles from 'qtwds256k.dat'   **
 **               This general program uses only quarter twiddles for FFT processing.*
 **               The resulting spectrum is placed in 'forward.dat' file(OPTIONAL)**
 **               This example program does FFT's from 2k Real to 256k Real only   **
 **                                                                                **
 **    Modules    :  main - Main routine                                           **
 **                                                                                **
 **    Author     : Anil M., Valley Technologies Inc.                              **
 **                                                                                **
 **    Language   : AT&T 'C'                                                       **
 **                                                                                **
 **    Build Commands: d3cc -c -Q -T -I<include_path> italy.c                      **
 **                                                                                **
 **    Update Log                                                                  **
 **                                                                                **
 **    Date      Initials   Description                                            **
 **                                                                                **
 **    4/6/94    alm        Creation()                                             **
 **       4/24/94  alm          added a window function                           **
 ************************************************************************************
 ************************************************************************************
 */

/** include files **/
#include "lh9124.h"             /* DSP Definitions */
#include "lh9320.h"             /* Address Generator defintions */
#define CLANGUAGE 1             /* To enable C macros defined in ultradsp.h  */

#include "ultradsp.h"           /* Misc defs and macros */
#include "hostio.h"              /* To access stdio type functions using hostio */
#include "valleyio.h"             /* for upload and download data and file type */

#define INT24 int
#include "jupiter.h"


#define TWDSIZE (64*1024 + 1)        /* Size of twiddle factors */
        char twiddles[] = "qtwds256k.dat";


#define O_RDONLY        0
#define O_WRONLY        1
#define O_RDWR          2


/** local definitions **/

        static struct tagMemAddr AddrGen;       /* Address generator Structure */

        unsigned int *streg = (unsigned int *)ST_REGISTER ;    /* Pointer to Status Reg
        unsigned int *dfreg = (unsigned int *)DF_REGISTER ;   /* Pointer to DF register
        unsigned int *fcreg = (unsigned int *)FC_REGISTER ;   /* Pointer to FC Register

   struct Operational_Parameters OpParam;

        int *Rptr, *Iptr; /* Pointer to data */


/** external functions **/

        void dmparam( );

/** external data **/
    extern int dfdata;
```

```
    extern int fcdata;

/*
 ************************************************************************
 *                                                                      *
 *  Function Name: main                                                 *
 *  Description:  main routine                                          *
 *  Syntax: int main( )                                                 *
 *  Parameters: none                                                    *
 *  Returns: none                                                       *
 ************************************************************************
 */

void main( )
{
   int K;                          /* FFT Size in bytes*/
   register int i,j;
   int pipedes;

 .

/** Initialize the ADDRESS Generators **/

        AddrGen.A = 0;                  /* set address generator A Base for moving the da
        AddrGen.Q = 0;                  /* set address generator Q Base for moving the da
        AddrGen.C = 0;                  /* set address generator C Base for moving the tw
        load_ag_address(&AddrGen);      /* Set Starting Address in AGs */



/*
 *  Open the Pipe to the control task
 *  and Read the Operational Parameters
 */

       pipedes = open("FFTPIPE",O_RDWR);
       if(pipedes == NULL)
       {
          fprintf(stderr,"\nError Opening Pipe");
          exit(0);
       }
       i = read(pipedes,&OpParam.cmd,sizeof(OpParam));
       if(i != sizeof(OpParam))
       {
          fprintf(stderr,"\nError Reading Pipe");
          exit(0);
        }
#ifdef DEBUG
     dmparam( );
#endif

/*
   ********************************************************************
   *                                                                  *
   *  Read in the twiddle factors from the host and load into         *
   *            coefficient 'C' (lower)memory                         *
   *                                                                  *
   ********************************************************************
   */
        access_RAMA();              /* Access  A memory on UltraDSP board */
        fprintf(stderr, "loading %s...\n",twiddles);
        download_a_file(twiddles, REALRAM, IMAGRAM, TWDSIZE*4,IO_BIN);

     /** The Twiddles Need to be divided by two since the hardware bit
      CSFI is diabled for the jupiter project **/

     /* Divide the Twiddle Coefficients by 2 to prevent overflow */

#if(0)
        Rptr = (int *)REALRAM;
        Iptr = (int *)IMAGRAM;

        for(i=0; i<TWDSIZE; i++)
        {
           *Rptr = (*Rptr)  + 0;
           *Iptr = (*Iptr)  + 0;
```

```
            *Iptr = *Iptr | 0x800000;
            Rptr++;
            Iptr++;
          }
          upload_a_file("uptwd.dat", REALRAM, IMAGRAM, TWDSIZE*4,IO_ASCII);

#endif

          load_ag_int(ADRLENGTHL, TWDSIZE);  /* load the AG length register for move fur
          move_bufferAtoC(0);              /* move data to C Ram bank */

    /*****************************************************************
     *                                                               *
     *  Read in the window function from the host and load into      *
     *            coefficient 'C' (lower)memory                      *
     *                                                               *
     *****************************************************************/

      access_RAMA();            /* Access  A memory on UltraDSP board */
      fprintf(stderr, "loading %s...\n",OpParam.win_file);

      K = OpParam.winsize;

      download_a_file(OpParam.win_file, REALRAM, IMAGRAM, K*4,IO_BIN);

#if(0)

      /* Divide the Window Coefficients by 2 to prevent overflow */

      Rptr = (int *)REALRAM;
      Iptr = (int *)IMAGRAM;

      for(i=0; i<K; i++)
      {
            *Rptr = (*Rptr)/2;
            *Iptr = (*Iptr)/2;
            Rptr++;
            Iptr++;
       }

#endif
      AddrGen.A = 0;                /* set the  */
      AddrGen.Q = 0;                /*     AG Starting Address  */
      AddrGen.C = 65537;            /*                for moving the window */
      load_ag_address(&AddrGen);

      load_ag_int(ADRLENGTHL, K);  /* load the AG length register for move func. */

      move_bufferAtoC(0);          /* move data to C Ram bank */


      printf("Load %s\n",OpParam.fft_file);
      load_program(OpParam.fft_file);

      exit(0);

}

void dmparam( )
{
          fprintf(stderr,"\ncmd:         %8.8x",OpParam.cmd);
          fprintf(stderr,"\nfs:          %d",OpParam.fs);
          fprintf(stderr,"\nwinsize:     %d",OpParam.winsize);
          fprintf(stderr,"\naveint:      %d",OpParam.ave_int);
          fprintf(stderr,"\navefloat:    %d",OpParam.ave_float);
          fprintf(stderr,"\nfft:         %s",OpParam.fft_file);
          fprintf(stderr,"\nWindow:      %s",OpParam.win_file);
          fprintf(stderr,"\n");
          return;
}
```

```
#
# file: makefile
#
# Desription: Make file for building mainfft test programs
#
# usage: 'make mainfft' for mainfft program
#
# author: Anil M., VTI 5/18/94
#
#  Update Log
#_____________________________________________________________________
#
IDIR=$(VALLEY)/include                  # .h include path
KDIR=$(VALLEY)/lib                     # libud32C path
STARTUDS = $(VALLEY)/lib/startuds.s

none:
        @echo You must specify a target to build ...
        @echo
        @echo ....... init       UltraDSP FFT init    program
        @echo ....... fft1k      UltraDSP FFT FFT1K   program
        @echo ....... fft2k      UltraDSP FFT FFT2K   program
        @echo ....... fft4k      UltraDSP FFT FFT4K   program
        @echo ....... fft8k      UltraDSP FFT FFT8K   program
        @echo ....... fft16k     UltraDSP FFT FFT16K  program
        @echo ....... fft32k     UltraDSP FFT FFT32K  program
        @echo ....... fft64k     UltraDSP FFT FFT64K  program
        @echo ....... fft128k    UltraDSP FFT FFT128K program
        @echo ....... fft256k    UltraDSP FFT FFT256K program
        @echo ....... avemag     UltraDSP AVERAGER Average Magnitude program
        @echo ....... jcntrl     Sun Sparc main control program
        @echo ....... rdtape     Sun Sparc Tape Reader Test Program
        @echo
        @echo ....... all        Build Everything
        @echo ....... clean      clean up the directory for a fresh build

all: init fft1k fft2k fft4k fft8k fft16k fft32k fft64k fft128k fft256k jcntrl rdtape a

clean:
        rm *.o
#
# Install the executables
#
install:
        @install jcntrl   $(VALLEY)/bin
        @install rdtape   $(VALLEY)/bin
        @install specplot $(VALLEY)/bin
#
# Build the Control Program
#
jcntrl:  jcntrl.c ieee.c vmio12.c
        cc -o jcntrl jcntrl.c ieee.c vmio12.c -I/usr/nivxi/include  -L/usr/nivxi -lm
#
# Build the Read Tape Program
#
rdtape:  rdtape.c
        cc -o rdtape rdtape.c -lm

#
# Build the Averager Program
#
avemag: avemag.o startuds.o
        d31d -L$(KDIR) -o $@ avemag.o jupiter.bld > $@.map

#
# Build the Initialize program for the FFT Application
#
init:   init.o startuds.o
        d31d -L$(KDIR) -o $@ init.o jupiter.bld > $@.map
#
# Build the ploting program
#
specplot: specplot.c
        cc specplot.c -lxview -lolgx -lX11 -lm  -I/usr/openwin/include -o specplot
```

```
#
#*************************************************************************
#  This section builds an executable for each of the FFT sizes
#  Each FFT object module is dirived from the same source file 'mainfft.c'
#  by passing the variable 'WHICHFFT' set to FFT size to the compiler
#

fft1k: fft1k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft2k: fft2k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft4k: fft4k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft8k: fft8k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft16k: fft16k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft32k: fft32k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft64k: fft64k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft128k: fft128k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map


fft256k: fft256k.o startuds.o
        d31d -L$(KDIR) -o $@ $@.o jupiter.bld > $@.map
#
#*************************************************************************
# Define the make for each of the FFT object files
# Pass the compiler a define specifing the FFT size
#
#
fft1k.o: mainfft.c
        d3cc -c -D_FFT1K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft2k.o: mainfft.c
        d3cc -c -D_FFT2K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft4k.o: mainfft.c
        d3cc -c -D_FFT4K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft8k.o: mainfft.c
        d3cc -c -D_FFT8K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft16k.o: mainfft.c
        d3cc -c -D_FFT16K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft32k.o: mainfft.c
        d3cc -c -D_FFT32K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft64k.o: mainfft.c
        d3cc -c -D_FFT64K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft128k.o: mainfft.c
        d3cc -c -D_FFT128K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c

fft256k.o: mainfft.c
        d3cc -c -D_FFT256K -Q -T -I$(IDIR) -o $@ -Wa,-l$@ mainfft.c
#
#*************************************************************************
```

```
# Define the make proceedure for startuds
#

startuds.o:     $(STARTUDS)
        d3as -Q -I$(IDIR) -lstartuds $(STARTUDS)

#
# Define the default compile commands
# for the DSP32C C and assembly programs
#
%.o:    %.s
        d3cc -c  -Q -T -I$(IDIR) -o $@ -Wa,-l$(*F) $<
#
%.o:    %.c
        d3cc -c  -Q -T -I$(IDIR) -o $@ -Wa,-l$(*F) $<
#
.SUFFIXES:

.SUFFIXES: .s .c .o

.l.c:
        @echo $<
.s.o:
        d3cc -c -DWHICHFFT='$(FFTTYPE)' -Q -T -I$(IDIR) -o $@ -Wa,-l$(*F) $<

.c.o:
        d3cc -c -DWHICHFFT='$(FFTTYPE)' -Q -T -I$(IDIR) -o $@ -Wa,-l$(*F) $<
```

```
/*
 ************************************************************************
 ************************************************************************
 **                                                                    **
 **    File Name: jupiter.h                                            **
 **                                                                    **
 **    Description: Include file for the jupiter project. This file    **
 **       contains data structures and constants for inclusion in      **
 **       both the UltraDSP applications and the Host Application      **
 **                                                                    **
 **    Author: D. Kolesar                                              **
 **                                                                    **
 **    Language: C                                                     **
 **                                                                    **
 **                                                                    **
 **    Update Log                                                      **
 **                                                                    **
 **    Date      Initials   Description                                **
 **                                                                    **
 **                                                                    **
 ************************************************************************
 ************************************************************************
 */

/* Define the structures for communication between the controlling
   host program and the UltraDSP Application

 INT24 - THis macro must be defined before #include jupiter.h
        On the SPARC, INT24 is defined as a "long "(32 bit) on the
        UltraDSP it is defined as an "int" (24 bits). The AT&T Compiler
        stores 24 bit ints, as 32 bits with the upper 8 bits as sign
        extend. By using this convention, these variables may be passed
        between the programs with no loss of accurcy.
        If the value is not defined it is assumed to be a long;
 */
#ifndef INT24
#define INT24 long
#endif

 /** Define the operational parameter data structure. This packet will be
  ** sent to both UltraDSP applicaions upon mission startup in order
  ** to define their operating parameters
  **/


   struct Operational_Parameters
   {
          INT24 cmd;           /* Set to 1 for this command  */
          INT24 fs;            /* Sampling Rate in Khz  */
          INT24 winsize;       /* Size of Window ( in complex words)    */
          INT24 ave_int;       /* Number of Integer averages to perform */
          INT24 ave_float;     /* Number of Floating Point Averages to perform */
         INT24 fft_size;       /* Size of the Real FFT  */
         INT24 scale_off;      /* Scale factor offset   */
         INT24 scale;          /* Fixed Scale factor    */
          char fft_file[12];   /* File name of FFT application routine to run  */
          char win_file[12];   /* File name of Window File to use  */
   };

/** Define the structure for Operational_Commands, These commands
    are used to control or change the operational mode of the
    UltraDSP. This structure is loosly defined as 8 long word (32 byte)
    structure and each unique implementation should maintain the size but
    change the meaning of the words
 **/

  struct Operational_Commands
 {
        INT24 cmd;       /* Command word, unique for each command */
        INT24 param[7]; /* Place Holder for up to 7 parameters */
 };

/** Define some values for the commands that are implemented */

#define SETUP 1              /* Initial parameter setup command */
```

```
#define RESTART_AVE 0x202    /* Restart Averagoing function */


/* Define the Header Record used for host data.
   THIS STRUCTURE IS NOT USED BY THE ULTRADSP APPLICATIONS
   That execute on the UltraDSP boards  */

#ifdef _sys_time_h

         struct RecordHeader     /* Record Header */
        {
           long sync[4];           /* 128 bit sync word     */
           long blknum;            /* Current Block Number */
           long fs;                /* Sampling Rate in Mhz */
           long winsize;           /* Size of Window ( in complex words)     */
           long ave_int;           /* Number of Integer averages to perform */
           long ave_float;         /* Number of Floating Point Averages to perform */
           long fft_size;          /* Size of the Real FFT */
           long scale_off;         /* Scale factor offset (not used) */
           long scale;             /* Fixed Scale factor  (not used) */
           char fft_file[12];      /* File name of FFT application routine to run */
           char win_file[12];      /* File name of Window File to use */
           struct timeval tp;      /* Computer Time */
           struct timezone tzp;
           char station_time[5]; /* Station Time */
           char spareut[3];        /* Pad up utime to long word */
           long source_flag;       /* On or Off source flag, 0 = off */
           long onsource;          /* Number of blocks on source */
           long offsource;         /* Number of blocks off source */
           long spare[5];         /* Pad up to 128 bytes */

        } ;

/** Define the on source and off source values for .source_flag */

#define OFFSOURCE 0
#define ONSOURCE 1

#endif
```

```
/**** Test of Pipe and fork routine for IPC **/

#include <stdio.h>
#include <errno.h>
#include <fcntl.h>

#define RD 0
#define WR 1

int pfd[2];     /* Pipe ids */

main( )
{
        int i,j,k;
        float *buf;     /* A buffer */
        int child_id;


/*** OPen up a pipe */

        /* pipe(pfd); */
/* Now fork the processes */

        child_id = fork( );


/** Child Processing */

        if(child_id == 0)
        {

                pfd[RD] = open("RDPIPE",O_RDONLY);
                buf = (float *)calloc(256*1024, sizeof( float));
                if(buf == NULL) { perror(" Child alloc: ");exit(0);}

                for(k=0; k<60; k++)
                {
                  i=0;
                  while(i<256*1024)
                  {
                    j = read(pfd[RD],&buf[i],4096);
                    if(j != 4096)
                    {
                        fprintf(stderr,"Child: Error reading pipe %d %d \n",pfd[RD],j
                        exit(0);
                     }
                     if(buf[i] != i)
                     {
                        fprintf(stderr,"child: Sync Error \n");
                     }
                     i += 1024;
                   }
                }  /* end of for(k loop */

                fprintf(stderr,"Child Exiting \n");
                cfree(buf);
                exit(0);
        }


 /** Parent Processing **/

        if(child_id != 0)
        {
                pfd[WR] = open("RDPIPE",O_WRONLY);
                buf = (float *)calloc(256*1024, sizeof(float));
                if(buf == NULL) { perror(" Parent alloc: ");exit(0);}

                for(k=0; k<60; k++)
                {
                  i=0;
                  while(i<256*1024)
                  {
                     buf[i] = i;
```

```
                    j = write(pfd[WR],&buf[i],4096);
                    if(j != 4096)
                    {
                       fprintf(stderr,"parent: Error writing pipe %d %d \n",pfd[WR],
                       exit(0);
                    }
                    i+=1024;
                  }
               } /* End of for(k  loop */

               fprintf(stderr,"parent Exiting \n");
               cfree(buf);
               exit(0);
         }
}
```

```
#include <stdio.h>
#include "/usr/nivxi/include/nivxi.h"
#include "spet.h"


 main()

{
int phase[6];
uint16 ph;
uint8 result;
uint8 i;
FILE *inpu;
inpu=fopen("/home/stelio/cprog/phase.dat","r");
/*fscanf(inpu,"%d", phase);*/

result=InitVXIlibrary();
resetvmio12();

initvmio12('2','b','o','n'); /*port b of IC 8 becomes out (address)*/
initvmio12('2','a','o','n'); /*port b of IC 8 becomes out (data)*/
initvmio12('3','a','i','n'); /*port a of IC 9 becomes in  (data)*/


enable('2');
enable('3');


for (i=0;i<=5;i++)
{
fscanf(inpu,"%d\n",phase+i);
/*printf("%d\n", phase[i]);*/

ph=64*phase[i]/360;
subcio_out(ph,cio_pshift_data,i,cio_pshift_add);

}
 result=CloseVXIlibrary();

}
```

7) - PRIME VALUTAZIONI DELLO STRUMENTO

Appena terminato l'assemblaggio dello spettrometro si e`collegato il sistema alla parabola VLBI tramite un cavo coassiale sotteraneo connesso ad un video converter, a larghezza di banda programmabile, del Mark III. Si sono effettuate osservazioni di righe molecolari di cui si riportano qui di seguito una serie di plottati. Tutte le osservazioni fatte non sono state calibrate, per cui si devono considerare effettuate solo per valutazioni a livello qualitativo. Infatti uno dei problemi principali incontrati nel post-processing dei dati relativi alle osservazioni dell' impatto di Giove con la SL-9, e` proprio stato quello della stima del flusso della riga di emissione osservata. In Fig. 27 viene riportato il plottato della osservazione della riga dell'ammoniaca di L183. Come si puo` notare, qui si sono usati 16K canali e sono gia` visibili dettagli (1 ora di osservazione on+off) che con gli autocorrelatori standard avrebbero richiesto osservazioni multiple di segmenti del pattern singoli per volta. Come gia` menzionato varie volte in precedenza, la costruzione dello spettrometro e` stata finalizzata alle osservazioni delle conseguenze dell' impatto dei frammenti della cometa SL-9 con l' alta atmosfera di Giove. In quella occasione il sistema ha permesso la rivelazione di una debole emissione in riga dell'acqua a 22.235 GHz nella zona dell'impatto del frammento E con Giove stesso. Il post processing dei dati e` stato effettuato quasi totalmente a "mano" nel senso che non erano stati scritti, per mancanza di tempo, software di supporto alla elaborazione dei dati. Uno dei principali problemi e` stato quello di dovere conoscere l' esatta velocita` relativa tra il blob preso in considerazione e l'antenna di Medicina. Questo perche` a 22 GHz la parabola di Medicina ha un beam di circa 2 primi d' arco che, a circa 800 milioni di chilometri, diventa maggiore delle dimensioni angolari di Giove per cui lo scenario in cui si e` operato e` rappresentato schematicamente in Fig. 28. I blobs in transito, all' interno di un determinato intervallo di tempo, erano visibili contemporaneamente per cui l'unica maniera di distinguerli era quella di compensare, un blob per volta, lo shift Doppler dovuto alla posizione occupata istante per istante. Questo, come gia` accennato, richiedeva la conoscenza esatta della velocita` relativa blob considerato / antenna VLBI. La compensazione e`stata resa possibile usando le informazioni sulla velocita` blob / centro della terra minuto per minuto, date da un package software "pescato" via internet dall'Osservatorio di Meudon (Francia) scritto da Pierre Colom la cui uscita appariva nel formato riportato in Tav. 5 La prima colonna e` riferita alla data corrente, la seconda all' ora UT (hh mm), la sesta alla velocita` mentre l' ultima riportava il frammento relativo. In Fig. 29a si puo` vedere come appare la riga dell' acqua relativa

alla zona di impatto del frammento E, ottenuta compensando il Doppler con la macro "Delay N" (N=numero di canali) di DADiSP™. In Fig.29b invece come appare la stessa riga ottenuta da elaborazioni fatte da Pierre Colom a Meudon sui dati forniti da Medicina via Internet, usando un software dedicato alla riduzione dati per spettroscopia (CLASS).

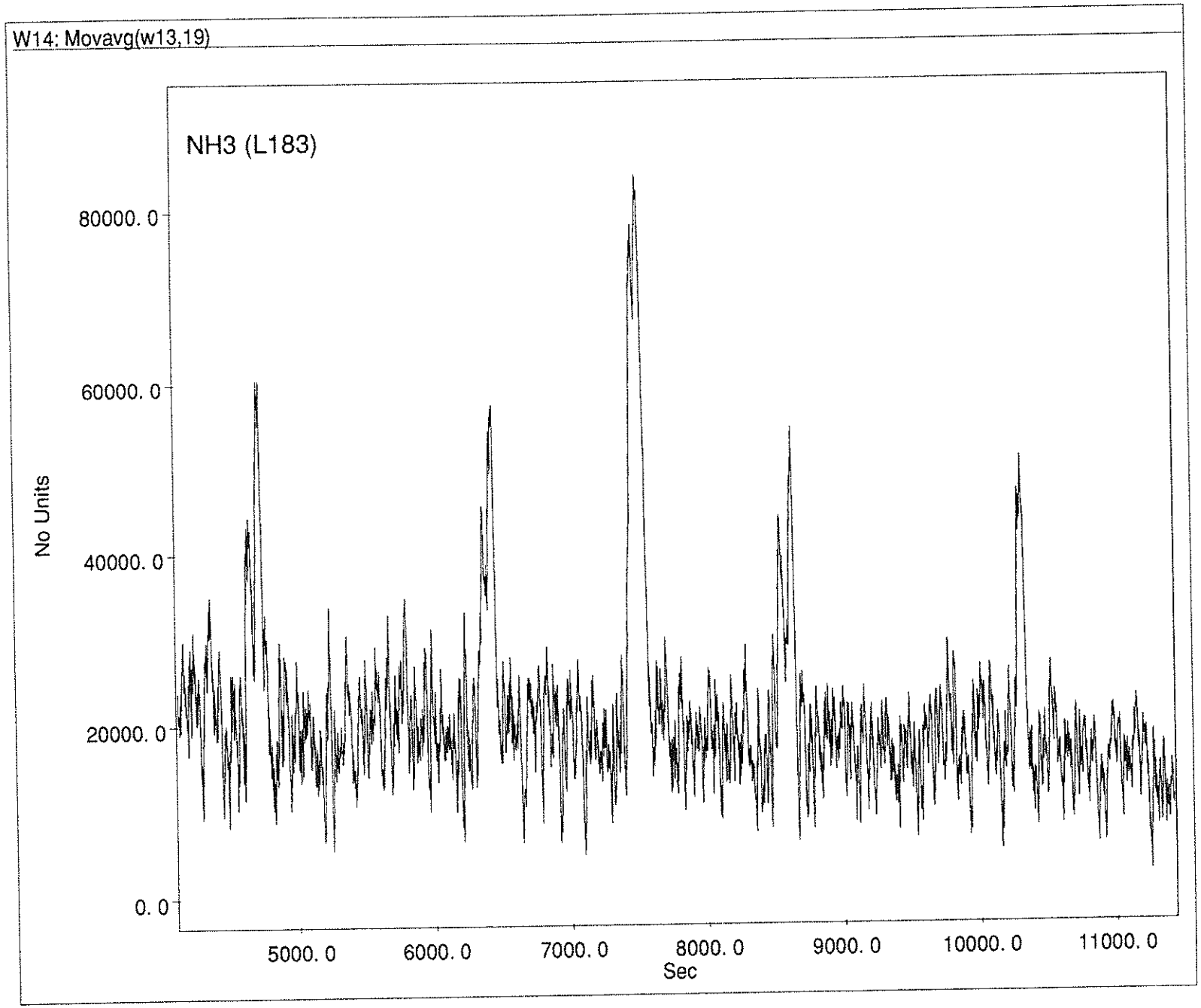

W14: Movavg(w13,19)
NH3 (L183)
80000. 0
60000. 0
40000. 0
20000. 0
0. 0
No Units
5000. 0
6000. 0
7000. 0
8000. 0
9000. 0
10000. 0
11000. 0
Sec

FIG. 27

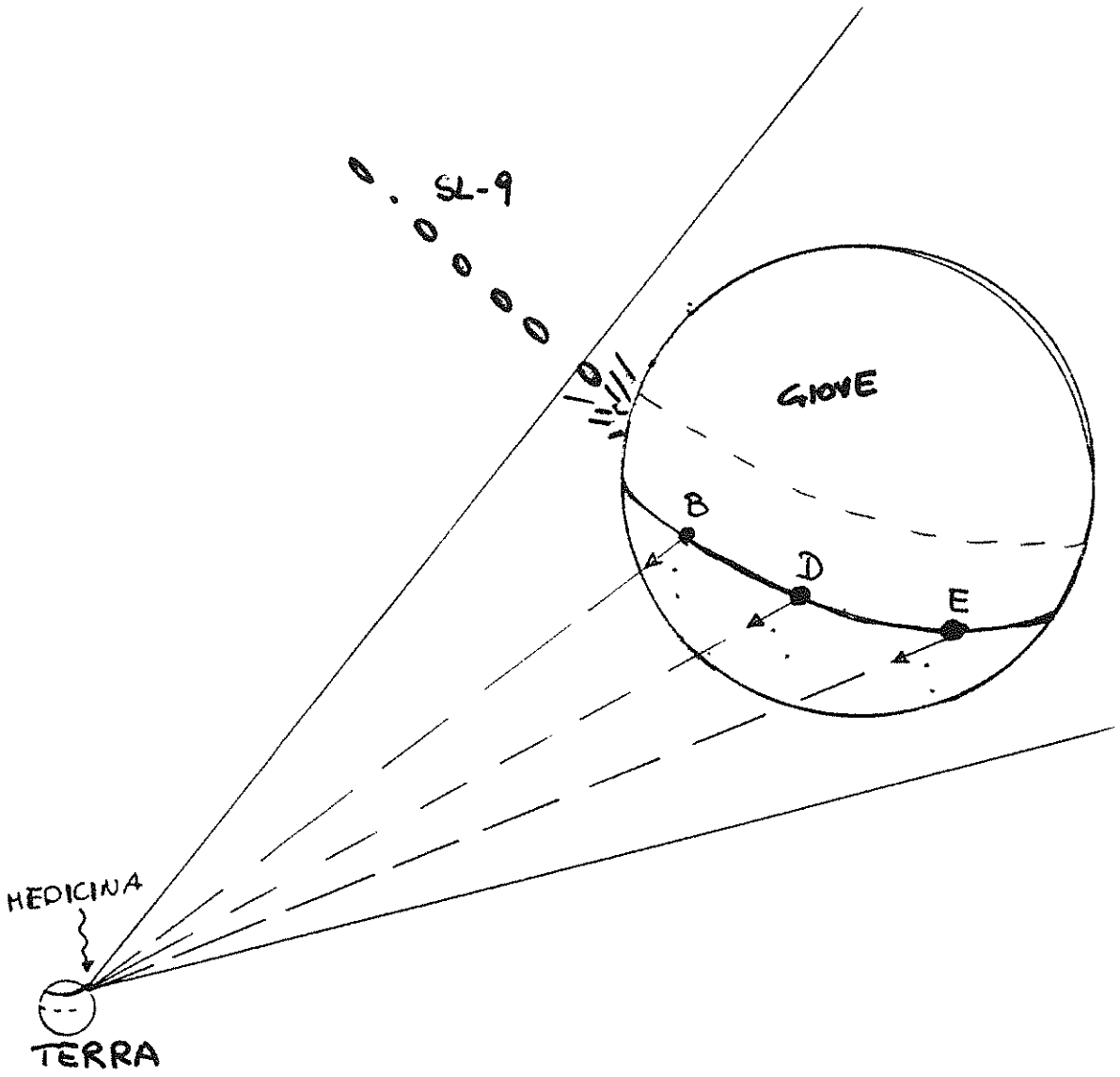


FIG. 28

19/ 7	18: 0	-9.1	-9.3	13.0	29.8	20=B
19/ 7	18: 0	-14.9	-7.5	7.8	33.8	18=D
19/ 7	18: 0	6.6	-15.6	7.1	18.7	17=E
19/ 7	18: 0	4.1	-14.5	10.6	20.5	16=F
19/ 7	18: 0	-15.8	-7.3	6.1	34.4	15=G
19/ 7	18: 0	-2.5	-11.9	13.8	25.2	14=H
19/ 7	18: 1	-9.2	-9.3	12.9	29.9	20=B
19/ 7	18: 1	-15.0	-7.5	7.6	33.9	18=D
19/ 7	18: 1	6.5	-15.6	7.2	18.7	17=E
19/ 7	18: 1	4.0	-14.5	10.7	20.6	16=F
19/ 7	18: 1	-15.8	-7.3	5.9	34.4	15=G
19/ 7	18: 1	-2.7	-11.8	13.8	25.3	14=H
19/ 7	18: 2	8.1	-16.5	0.1	17.5	21=A
19/ 7	18: 2	-9.3	-9.2	12.9	30.0	20=B
19/ 7	18: 2	-15.0	-7.5	7.5	33.9	18=D
19/ 7	18: 2	6.5	-15.6	7.4	18.8	17=E
19/ 7	18: 2	3.9	-14.4	10.8	20.6	16=F
19/ 7	18: 2	-15.9	-7.3	5.8	34.5	15=G
19/ 7	18: 2	-2.8	-11.8	13.8	25.3	14=H
19/ 7	18: 3	8.1	-16.5	0.3	17.5	21=A
19/ 7	18: 3	-9.5	-9.2	12.8	30.1	20=B
19/ 7	18: 3	-15.1	-7.4	7.4	33.9	18=D
19/ 7	18: 3	6.4	-15.5	7.5	18.8	17=E
19/ 7	18: 3	3.8	-14.4	10.8	20.7	16=F
19/ 7	18: 3	-15.9	-7.3	5.7	34.5	15=G
19/ 7	18: 3	-2.9	-11.7	13.9	25.4	14=H
19/ 7	18: 4	8.1	-16.5	0.4	17.5	21=A
19/ 7	18: 4	-9.6	-9.1	12.8	30.2	20=B
19/ 7	18: 4	-15.2	-7.4	7.3	34.0	18=D
19/ 7	18: 4	6.3	-15.5	7.6	18.9	17=E
19/ 7	18: 4	3.7	-14.3	10.9	20.8	16=F
19/ 7	18: 4	-16.0	-7.3	5.5	34.5	15=G
19/ 7	18: 4	-3.0	-11.7	13.9	25.5	14=H
19/ 7	18: 5	8.1	-16.5	0.6	17.5	21=A
19/ 7	18: 5	-9.7	-9.1	12.7	30.2	20=B
19/ 7	18: 5	-15.2	-7.4	7.2	34.0	18=D
19/ 7	18: 5	6.3	-15.5	7.7	18.9	17=E
19/ 7	18: 5	3.6	-14.3	11.0	20.8	16=F
19/ 7	18: 5	-16.0	-7.2	5.4	34.6	15=G
19/ 7	18: 5	-3.2	-11.6	13.9	25.6	14=H
19/ 7	18: 6	8.1	-16.5	0.7	17.5	21=A
19/ 7	18: 6	-9.8	-9.1	12.6	30.3	20=B
19/ 7	18: 6	-15.3	-7.4	7.0	34.1	18=D
19/ 7	18: 6	6.2	-15.4	7.8	19.0	17=E
19/ 7	18: 6	3.5	-14.2	11.1	20.9	16=F
19/ 7	18: 6	-16.1	-7.2	5.3	34.6	15=G
19/ 7	18: 6	-3.3	-11.6	13.9	25.7	14=H
19/ 7	18: 7	8.1	-16.5	0.8	17.5	21=A
19/ 7	18: 7	-9.9	-9.0	12.6	30.4	20=B
19/ 7	18: 7	-15.4	-7.4	6.9	34.1	18=D
19/ 7	18: 7	6.1	-15.4	8.0	19.0	17=E
19/ 7	18: 7	3.4	-14.2	11.2	21.0	16=F
19/ 7	18: 7	-16.1	-7.2	5.1	34.6	15=G
19/ 7	18: 7	-3.4	-11.5	13.9	25.8	14=H
19/ 7	18: 8	8.1	-16.5	1.0	17.5	21=A
19/ 7	18: 8	-10.1	-9.0	12.5	30.5	20=B
19/ 7	18: 8	-15.4	-7.4	6.8	34.2	18=D
19/ 7	18: 8	6.1	-15.4	8.1	19.1	17=E
19/ 7	18: 8	3.3	-14.2	11.3	21.1	16=F
19/ 7	18: 8	-16.2	-7.2	5.0	34.7	15=G
19/ 7	18: 8	-3.6	-11.5	13.9	25.9	14=H
19/ 7	18: 9	8.1	-16.5	1.1	17.5	21=A
19/ 7	18: 9	-10.2	-8.9	12.4	30.6	20=B
19/ 7	18: 9	-15.5	-7.3	6.7	34.2	18=D

TAV5

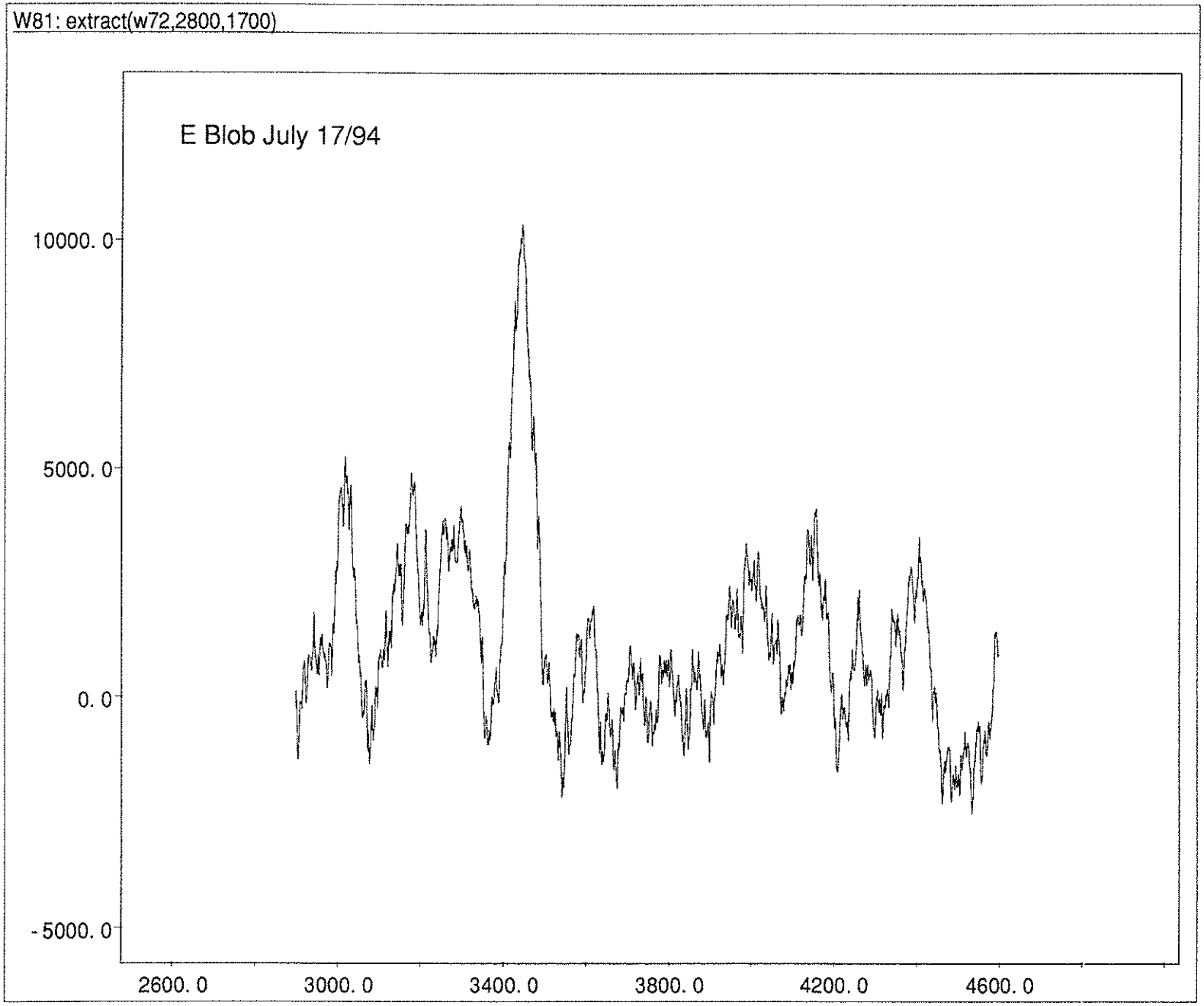

W81: extract(w72,2800,1700)
E Blob July 17/94
10000. 0
5000. 0
0. 0
-5000. 0
2600. 0
3000. 0
3400. 0
3800. 0
4200. 0
4600. 0

Fig. 29a

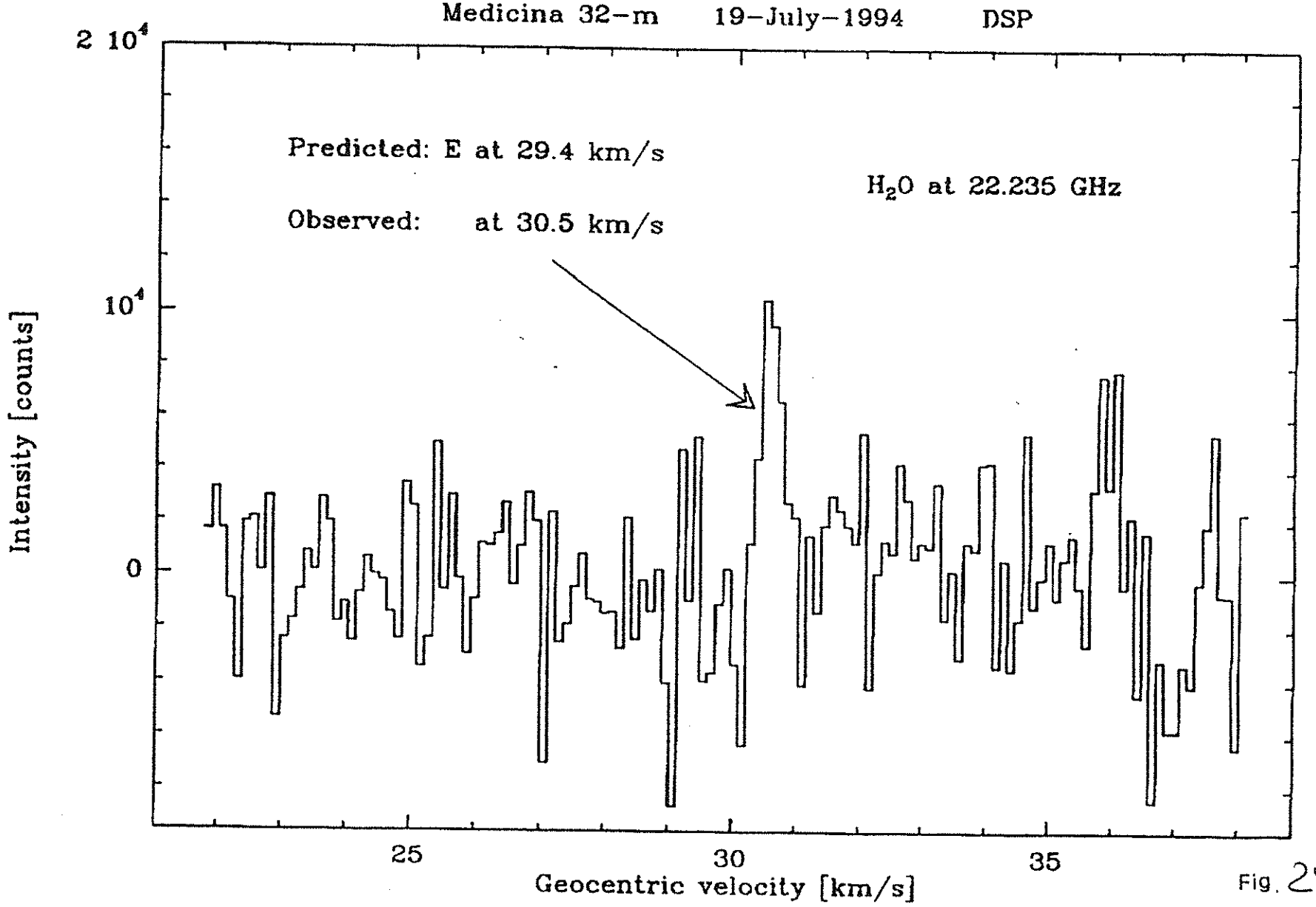

Medicina 32-m 19-July-1994 DSP
2 10⁴
10⁴
0
Intensity [counts]
Predicted: E at 29.4 km/s
Observed: at 30.5 km/s
H2O at 22.235 GHz
25
30
35
Geocentric velocity [km/s]

Fig. 29 b

8) - UPGRADE DELLO STRUMENTO.

Una delle limitazioni maggiori dello strumento nella sua versione base appena presentata e` rappresentata dalla bassa efficienza (time duty-cycle). In particolare, l' efficienza del 45% a 6 MHz riportata nella tabella riassuntiva delle caratteristiche principali dello spettrometro, e` dovuta principalmente a:

1) Throughput rate limitata nel link Spark-VME (National Ins. NI-VXI).
2) Mancanza di un doppio buffer nella scheda Ultra-DSP di calcolo della FFT
   e nella scheda Ultra-DSP che calcola la media.
3) Mancanza di un doppio buffer tra la scheda che esegue la media e la Sparc station.

Per questo motivo si e` deciso di effettuare un upgrade dello spettrometro per rimuovere le limitazioni citate. Viene riportate di seguito la tabella contenente le caratteristiche principali che lo spettrometro dovrebbe presentare nella nuova configurazione.

- System Controller:	40 MHz Sparc Comp.
- Input Bandwidth:	0.5 / 20 MHz
- Typical Efficiency:	100% @ 12 MHz
	50% @ 20 MHz
- Dual Channel Mode	100% @ 6 MHz
	25% @ 20 MHz
- Number of channels:	512,..131072 (power of 2)
- Average	1/256 @ step 1
	256/64K @ step 256
- Output Format:	32 bit Float. Point
- Windows:	Hann., Hamm.,
	Kaiser-Bessel, etc..
- DSPs used:	2 x Sharp LH9124
- Mass Storage device:	DAT ~2.2 Gb 4 mm tape

Il sistema e` predisposto per potere operare in “single” o “dual channels” mode. In modo canale singolo il sistema e` in grado di di processare dati in real time su una banda di 12 MHz (cioe`senza perdere dati) ed in modo bursted fino a 20 MHz. Anche in questo caso, con appropriati algoritmi, lo

spettrometro fornisce lo spettro positivo programmabile da 512 fino ad un massimo di 131072 canali. E` in grado di fornire spettri singoli o il risultato della media (per ridurre la varianza del noise dove richiesto) di N spettri con N che puo` variare da 2 a 65536 (da 1 a 256 spettri a step di 1 e da 256 a 65536 a step di 256). In modo dual channel, per polarimetria o altro, il sistema e` in grado di accettare agli ingressi bande di 6 MHz ed elaborare i dati in real time (100%). Lo schema a blocchi della parte di elaborazione del nuovo sistema viene riportato in Fig. 30 (gli altri blocchi rimangono invariati). Questo e` formato da due convertitori A/D, due moduli Ultra-DSP-1128 FFT processor, una scheda VT-524 per il calcolo del modulo quadro e la media degli spettri ed infine, una single board Sparc Force CPU3CE collegata ad una unita`nastro tipo DAT da 4 mm. Il sistema di calcolo sopradetto e` collegato, via un link Ethernet TCP/IP, alla Workstation AXIL da 85 MHz che funge da consolle per operatore. Analizziamo ora lo stesso schema a blocchi in maniera piu` dettagliata.

I blocchi di ingresso (1) e (2) sono costituiti da due schede Ultra-ADC Valley Tech. da 10 bit e 40 Msample/ s. per la conversione analogico/digitale del segnale o dei segnali di ingresso. La rappresentazione del segnale nel dominio delle frequenze (FFT) e` calcolata dai due moduli Ultra-DSP-1128. Ciascuna di queste e` composta da due schede ed occupano, di conseguenza, due slot VME: una occupata dalla Ultra-DSP card e l' altra dalla nuova scheda 1128 che ha a bordo due buffer di memoria in ingresso e due buffer per l'uscita (porta QR / QI) da 256KWords rispettivamente ed un DSP (AT&t 32C) in grado di operare la "finestrizzazione" dei dati nel dominio del tempo prima, cioè, che questi siano forniti alla scheda per il calcolo della FFT. Questo permette di usare un passo Radix-16 già nel primo stadio della FFT, velocizzando così il calcolo. Si ricorda, infatti, che se si vuole effettuare il windowing dei dati senza perdere tempo occorre usare nel primo passo un Radix-2 o 4 in quanto essendo i coefficienti di questi unitari, si possono sostituire con i coefficienti della window. Essendo però obbligati ad inserire un Radix-2 o 4 al primo passo, occorrono in genere più passi di conseguenza il tempo globale di calcolo diventa più lungo. Nel modo dual channel ogni modulo composto Ultra-DSP-1128 calcola FFT (24 bit block floating point) fino a 256Kpunti in real time fino a 6 MHz. Nel modo canale singolo, i due moduli Ultra-DSP-1128 sono multiplexati in tempo e permettono il calcolo di FFT di 256Kpunti in real time su bande di ingresso di 12 MHz. Gli spettri forniti, dopo il computo del modulo quadro, possono essere mediati insieme o separatamente a bordo della scheda VT-524.

I coefficienti della finestra ed i dati relativi alla lunghezza della FFT vengono caricati sui moduli Ultra-DSP-1128 attraverso il bus VME rendendo il sistema estremamente flessibile e programmabile on-fly per il

compito richiesto. L' aggiunta del modulo 1128 alla scheda Ultra-DSP è stata fatta per avere a disposizione un doppio buffer nella parte di ingresso/uscita dei dati nel chip Sharp LH9124 (porta Q). Questa soluzione permette al chip LH9124, a bordo di ogni Ultra-DSP, di calcolare la FFT sui dati correnti mentre scarica il risultato precedente sul buffer di uscita e nello stesso tempo aquisisce i nuovi dati su quello di ingresso. Questo permette di bypassare uno dei problemi che limitava in maniera più determinante la throughput rate della configurazione base di partenza.

La scheda VT524 esegue il modulo quadro (16 bit) di ogni coppia complessa che compone lo spettro in uscita dalla Ultra-DSP-1128 (24+24 bit block floating point) e la media degli spettri ottenuta usando 32 bit in virgola fissa. Questo porta a 65536 (64K) il limite superiore del numero degli spettri che si possono mediare senza incorrere in problemi di overflow. Il risultato finale della media viene trasferito al controllore del VME (FORCE 3CE embedded Sparc board) via lo stesso bus VME. La scheda VT524 ha a bordo due buffer di memoria che gli permettono di acquisire dati (spettri) dai moduli Ultra-DSP-1128 mentre la precedente media viene scaricata sul bus VME per essere passata alla Sparc Force-3CE. Viene così rimossa un' altra causa di rallentamento, presente nel processing dei dati della precedente architettura, che contribuiva al forte degrado del time duty cycle. Il controller del sistema, come già accennato in precedenza, è costituito da una Sparc station su singola scheda che, oltre a gestire tutto il sistema, sovraintende alle operazioni di handling del bus VME. Questa Sparc è un calcolatore su singola scheda della FORCE computer tedesca mod. 3CE con sistema operativo SOLARIS 1.1B. Le sue funzioni sono principalmente quelle di effettuare il setup ed il controllo dei moduli che operano il processing dei dati ed il trasferimento degli spettri mediati dalla scheda VT524 all' unità nastro DAT da 4 mm. L' unità nastro è collegata alla Sparc tramite una interfaccia tipo SCSI2 e presenta una throughput rate (alla testina) di circa 800/900 KByte/Sec. Lavorando con il massimo numero di canali (131072) e considerando che le medie vengono effettuate con aritmetica a 32 bit in virgola fissa, ogni spettro è di 524288 byte per cui se non si vuole perdere dati occorre mediare a bordo circa 50/55 spettri prima di scaricarli su nastro (10/12 mSec. è il tempo necessario per ogni spettro). In queste condizioni operative la risoluzione temporale è di circa 0.5 secondi. Lavorando invece con 1024 canali, gli spettri mediati sono di 4096 byte per cui occorrono all' unità nastro circa 4 mSec. per la memorizzazione in modo burst. Anche in questo caso occorre mediare circa 50/55 spettri prima di scaricare su nastro il risultato; la risoluzione temporale risulta essere, ovviamente, di 4 mSec.

Il controllo ed il monitoraggio dell' intero sistema è effettuato remotamente tramite una Sparc AXIL da 85 MHz che funge da consolle della FORCE 3CE.

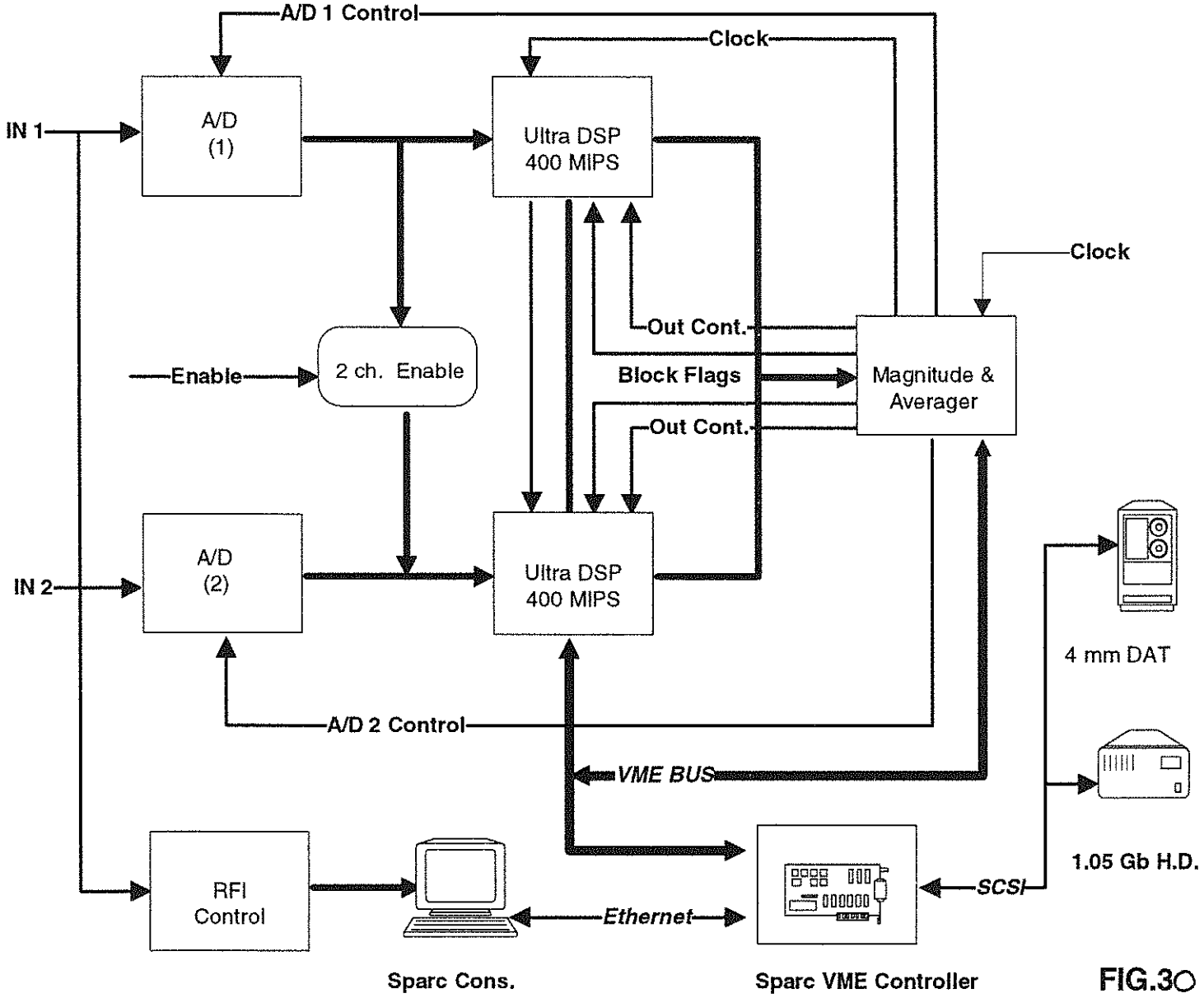

A/D 1 Control
Clock
IN 1
A/D
(1)
Ultra DSP
400 MIPS
Clock
Out Cont.
Enable
2 ch. Enable
Block Flags
Magnitude &
Averager
Out Cont.
IN 2
A/D
(2)
Ultra DSP
400 MIPS
4 mm DAT
A/D 2 Control
VME BUS
1.05 Gb H.D.
RFI
Control
SCSI
Ethernet
Sparc Cons.
Sparc VME Controller

FIG.3O

9) - FUTURI UPGRADE

L'upgrade in corso, come appena visto, portera` il sistema ad una efficienza massima (100%) con banda di 12 MHz ma il numero dei canali rimarra` sempre 131072 come nella versione base dello spettrometro. Un futuro upgrade potrebbe essere quello di aumentare il numero di canali fino a 8/16 milioni per potere rendere idoneo il sistema ad operare all'interno del programma SETI. Vi sono vari tipi di approccio al problema di cui, in questo capitolo, si fara` una rapida panoramica. Per aumentare il numero di canali, due sono le strade percorribili in maniera più immediata:

A) -Uso di FFT processor in grado di effettuare FFT di array molto lunghi (dell' ordine di 16/32 milioni di punti nel nostro caso)

B) -Uso di banchi di filtri digitali di tipo polifase.

Vediamo ora di analizzare brevemente ciscuno dei due tipi di approccio cercando di valutare in ambedue i casi quali sono i vantaggi e quali gli svantaggi.

A) Per questo caso, occorre avere un FFT processor che sia in grado di gestire matrici di grandi dimensioni per potere effettuare FFT di array con molti milioni di punti. In pratica per potere fare una FFT di 32 milioni di punti occorre un sistema come riportato in Fig. 31. Il blocco (1) e` costituito dall'attuale sistema Valley Tech. in grado di effettuare FFT di array di 131072 punti in 10 mSec. Se si acquisiscono, ad esempio, 128 spettri (stringhe ax1 ax2 ax3 ....axn) si puo` formare nel blocco (2) una matrice composta da 131072 colonne e 64 righe (128 Kx128=16 Mpunti). Ogni punto di questa matrice e` ottenuto moltiplicando ogni punto dello spettro ax1 ax2 ax3...axn per un apposito coefficiente di fase contenuto nella memoria identificata dal blocco (3) (si ricorda che questi sono spettri acquisiti in tempi diversi per cui un ritardo nel dominio del tempo e` uno shift di fase in quello delle frequenze). A questo punto l' algoritmo richiederebbe di ricalcolare la FFT colonna per colonna come riportato nella stessa Fig. 31 al blocco(2). Un calcolo di questo tipo assorbirebbe una grossa quantita` di tempo nell' indirizzamento dei dati colonna per colonna e quindi distanti 131072

punti l' uno dall' altro. Per rendere piu` veloce questa fase, conviene muovere i dati nella memoria formando la matrice trasposta come nel blocco (3) della Fig.31. A questo punto del processo la colonna 1 della matrice1 risulta essere la riga 1 della matrice 2 (trasposta) dove i dati sono uno di seguito all' altro e di conseguenza facilmente e velocemente indirizzabili (ovviamente questo vale per tutte le rimanenti colonne/righe delle matrici 1/2). Il computo della seconda FFT risulta essere ora molto efficiente. Alla fine si "attaccano" tutti i segmenti di spettro ottenuti per ricavare la FFT dei 16 milioni di punti. Un schema a blocchi di un tale sistema potrebbe essere quello riportato in Fig. 32. I blocchi (1), (2) e (3) espletano gli stessi compiti gia` visti in precedenza mentre la gestione della matrice viene affidata alle schede CES 64040 del blocco (4). Uno dei problemi piu` complessi da affrontare in questo sistema e` la grande quantita` di memoria richiesta dai corrispondenti blocchi (2),(3) e (4) della Fig. 31 e piu`precisamente:

- memoria per i coefficienti di fase (16 Mpunti in floating point ----> 48 Mbyte)
- Memoria per la matrice 1 (131072 x 4 x 128 K ---->64 Mbyte)
- la matrice 2 (131072 x 4 x 128 K ---->64 Mbyte)

La matrice 2 potrebbe rioccupare lo stesso spazio occupato dalla matrice 1 riducendo cosi` a 112 Mbyte la memoria richiesta. Le schede del blocco (4) devono anche essere in grado di muovere grosse quantita` di dati in tempi estremamente brevi. Questo implica che le schede montino a bordo processori e memorie estremamente veloci. Le schede della CES fanno uso dei nuovi chip power PC 604 da 100 MHz. Un' alternativa all'uso di schede molto costose come le CES, potrebbe essere rappresentata dall'uso di Alfa Station in single board per VME.

Uno spettrometro che si basa su un FFT processor di questo tipo soffre pero` di problemi di dinamica introdotti da eventuali presenze di forti radio interferenze nella banda analizzata. Come si puo` vedere nella Fig.33 una ipotetica interferenza non troppo forte ma visibile a 29 MHz (caso a) viene compressa quasi totalmente con il noise in presenza di una forte interferenza a 31 MHz (caso b). Questo perchè l' FFT processor scala fortemente il risultato per potere fare stare in scala anche la riga più forte.

Il numero di canali in questo caso è limitato, oltre che da problemi di dinamica, dalle dimensioni dei buffer richiesti per i termini di fase e dati.

C:\ABC\LONGFFT.AF3

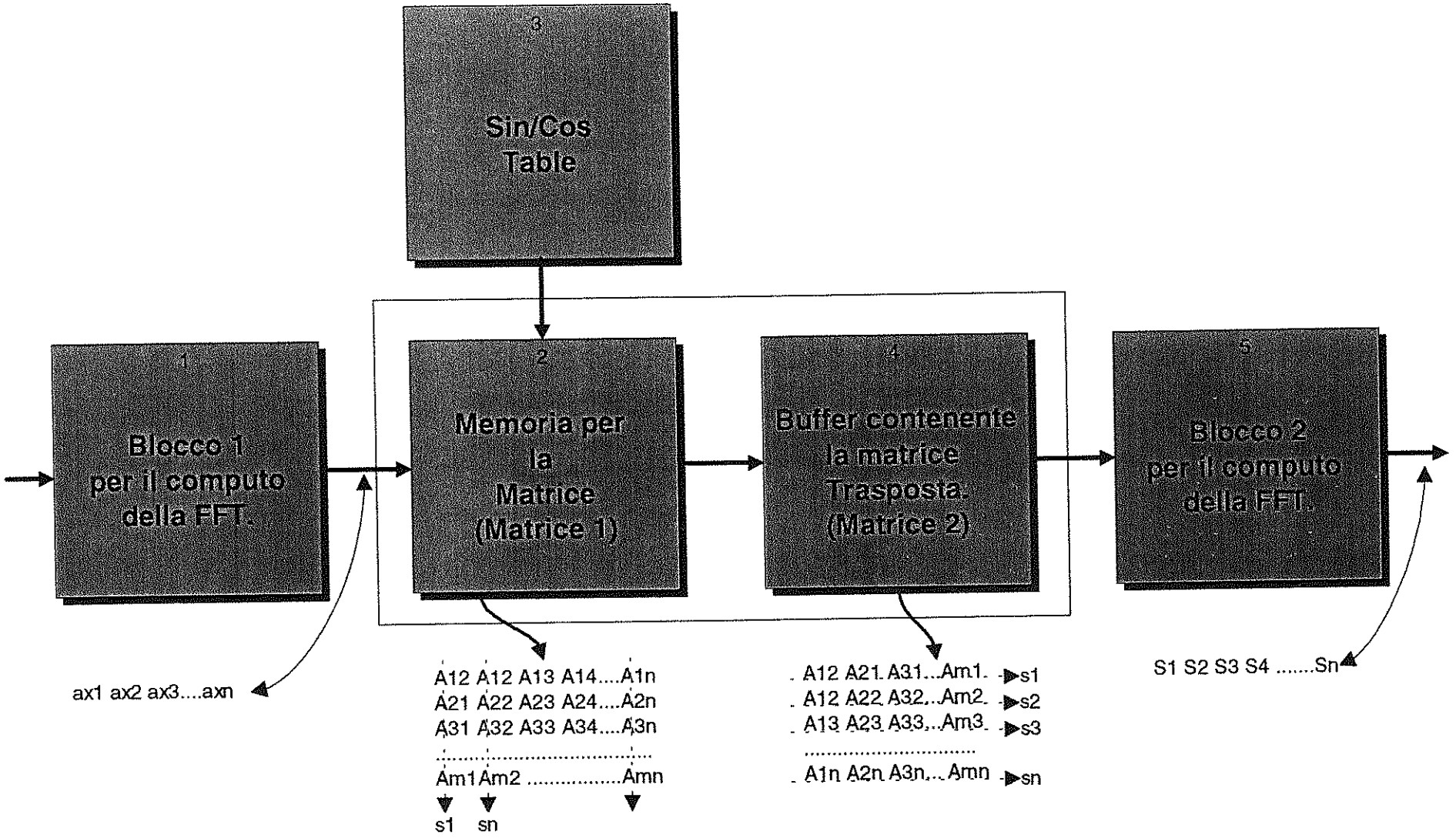


FIG. 31

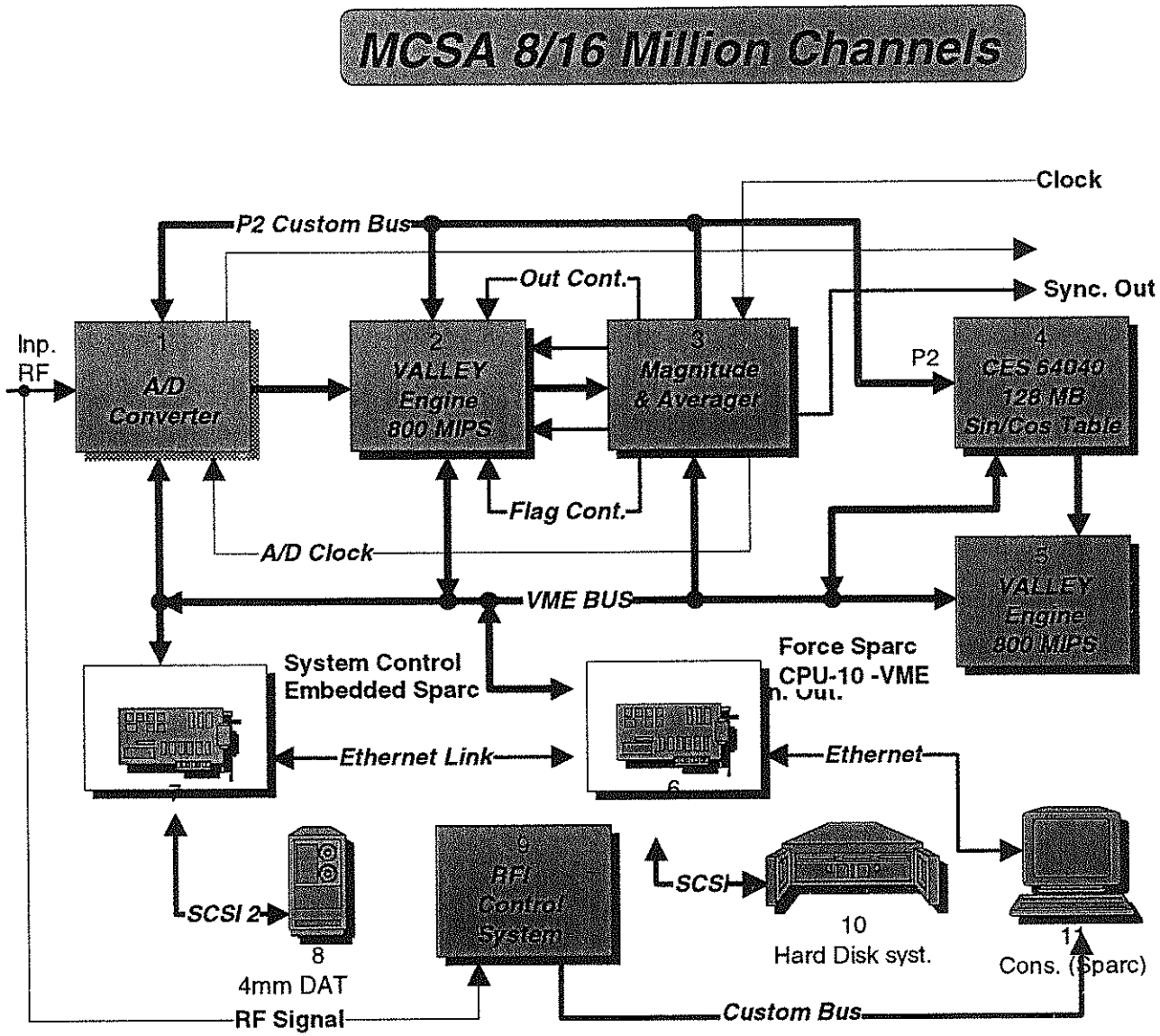
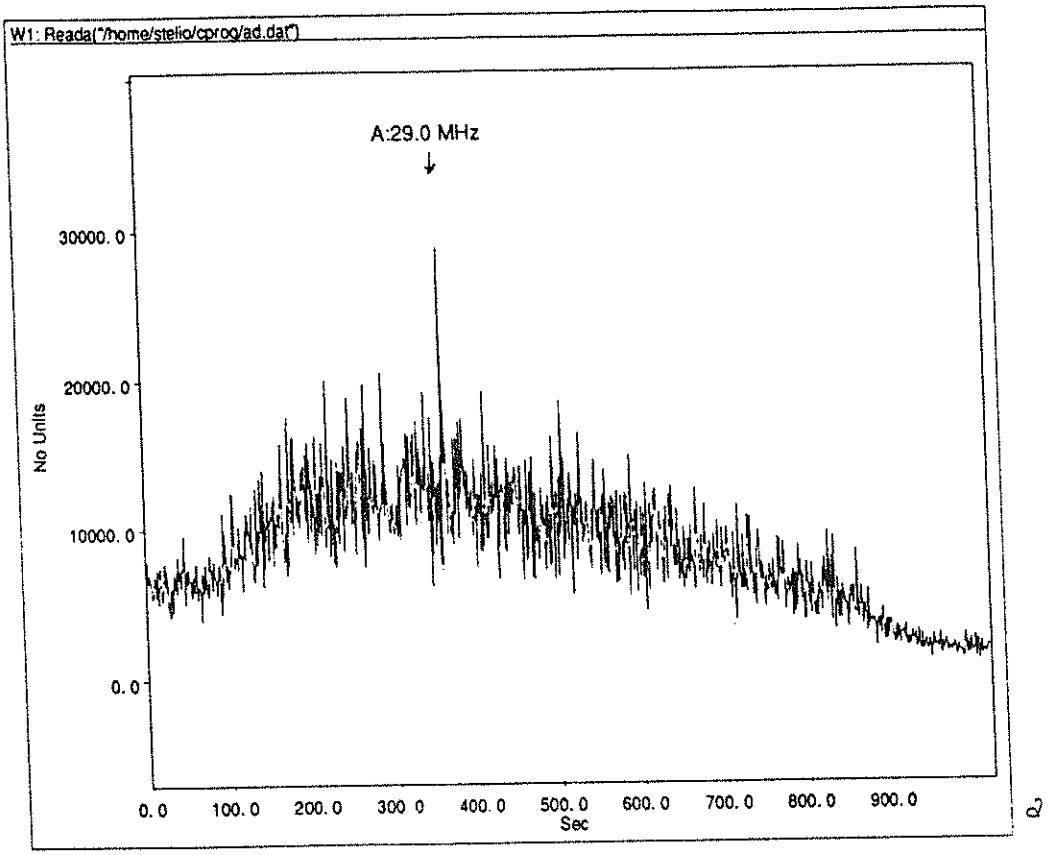
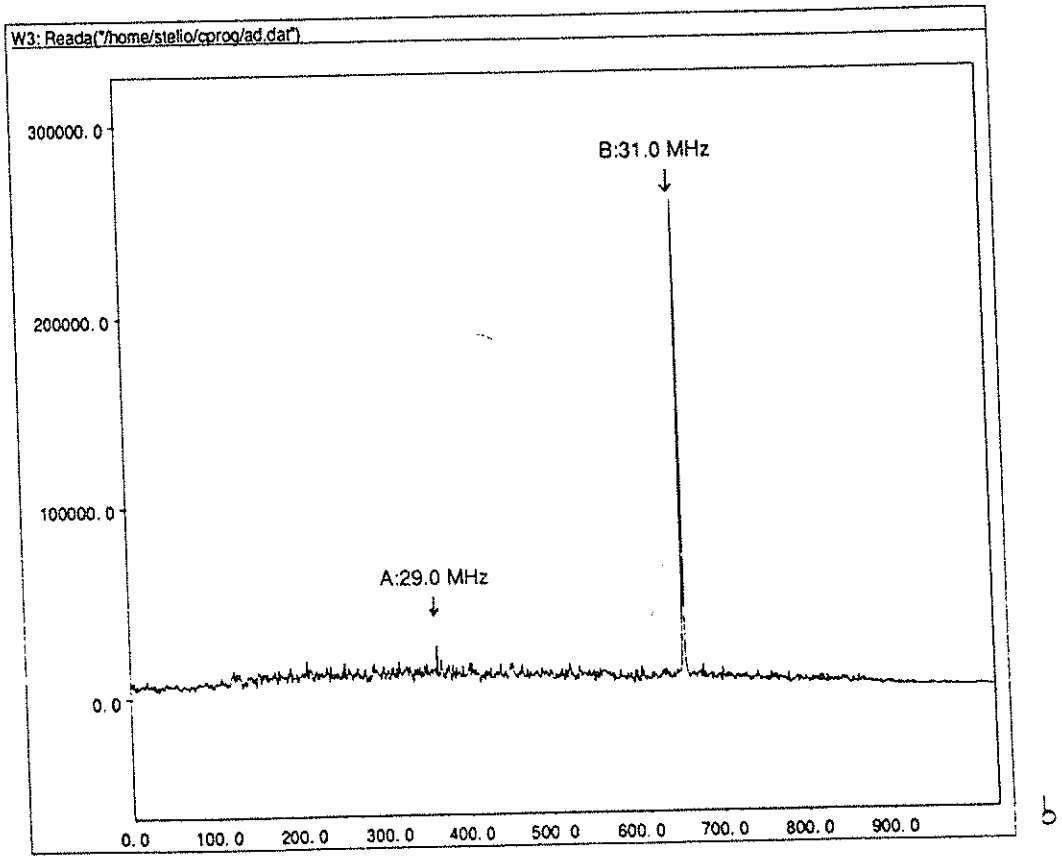

MCSA 8/16 Million Channels
Clock
P2 Custom Bus
Out Cont.
Sync. Out
Inp. RF
1 A/D Converter
2 VALLEY Engine 800 MIPS
3 Magnitude & Averager
P2
4 CES 64040 128 MB Sin/Cos Table
Flag Cont.
A/D Clock
VME BUS
5 VALLEY Engine 800 MIPS
System Control Embedded Sparc
Force Sparc CPU-10 -VME
Ethernet Link
Ethernet
7
6
SCSI 2
8
4mm DAT
9 RF Control System
SCSI
10
Hard Disk syst.
11
Cons. (Sparc)
RF Signal
Custom Bus

FIG. 32


W1: Reada("/home/stelio/cprog/ad.dat")
A:29.0 MHz
30000. 0
20000. 0
10000. 0
0. 0
No Units
0. 0
100. 0
200. 0
300 0
400. 0
500. 0
600. 0
700. 0
800. 0
900. 0
Sec

a


W3: Reada("/home/stelio/cprog/ad.dat")
300000. 0
200000. 0
100000. 0
0. 0
B:31.0 MHz
A:29.0 MHz
0. 0
100. 0
200. 0
300. 0
400. 0
500 0
600. 0
700. 0
800. 0
900. 0

b

FIG. 33 a-b

B) In questo caso si prevede l' uso di banchi di filtri digitali. A questo punto è necessario accennare, anche se brevemente, ai filtri digitali. Con un segnale monocromatico quasi tutti i filtri lavorano correttamente, mentre se il segnale è composto da una miriade di frequenze diverse dobbiamo, invece, iniziare a prendere in cosiderazione la risposta di fase del filtro. Se diverse frequenze componenti il segnale impiegano tempi diversi per attraversare il filtro, l' uscita dello stesso sarà distorta. L' unica maniera di evitare distorsioni è quindi quella di usare filtri lineari in cui qualsiasi frequenza componente il segnale impiega lo stesso tempo ad attraversarli. I FIR filter digitali sono per definizione filtri lineari ed in pratica effettuano la convoluzione tra i dati che arrivano e la rispota del filtro ad un impulso:

$$A(n) = \sum_{k=0}^{n-1} c(k)d(n-k) \qquad \text{n=0,1,2,3,.....2n-2}$$

dove A(n) è la risposta ad un determinato istante, c(k) sono i coefficienti e d(n) i dati in ingresso. Questa formula assomiglia molto a quella della correlazione:

$$A(n) = \frac{1}{N} \sum_{k=0}^{N-n-1} c(k)d(k) \qquad \text{n=-n+1 .... -1, 0, 1, 2 .....n}$$

Infatti, quando si parla di correlazione in senso non matematico, spesso si intende "convoluzione". Sempre in senso "non matematico", un FIR filter è una "forma modificata" della cross-correlazione tra un segnale noto ed il segnale di ingresso. La implementazione hardware di un FIR è abbastanza intuitiva e consiste nella moltiplicazione dei dati in ingresso con i relativi coefficienti, nella somma dei risultati e da uno shift dei dati come da Fig. 34 a,b. Ovviamente e` possibile implementare lo stesso processo via software. E`, quindi, chiaro che un banco di n filtri digitali sarebbe costituito da n filtri (Fig.35) come appena visto, e quindi non praticamente realizzabile da un certo numero di canali in poi (16/32) per l'elevato numero di dispositivi necessari (versione hardware) o per la richiesta di elevate capacita` e velocita` di calcolo (versione software). A questo punto vengono in aiuto le conseguenze relative all' alterazione della sampling rate di un segnale e più precisamente, la decimazione dei campionamenti.

C:\ABC\FIR.AF3

n-taps FIR Filters

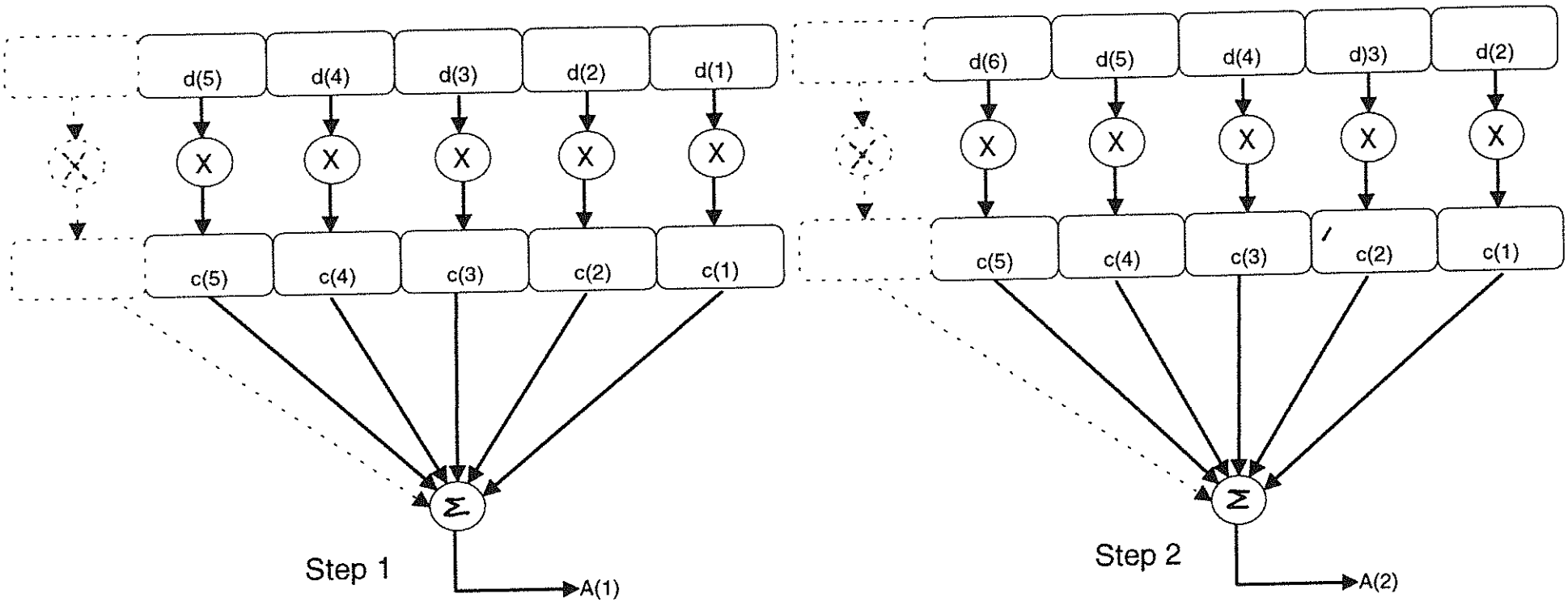


FIG. 34

C:\ABC\BANK.AF3

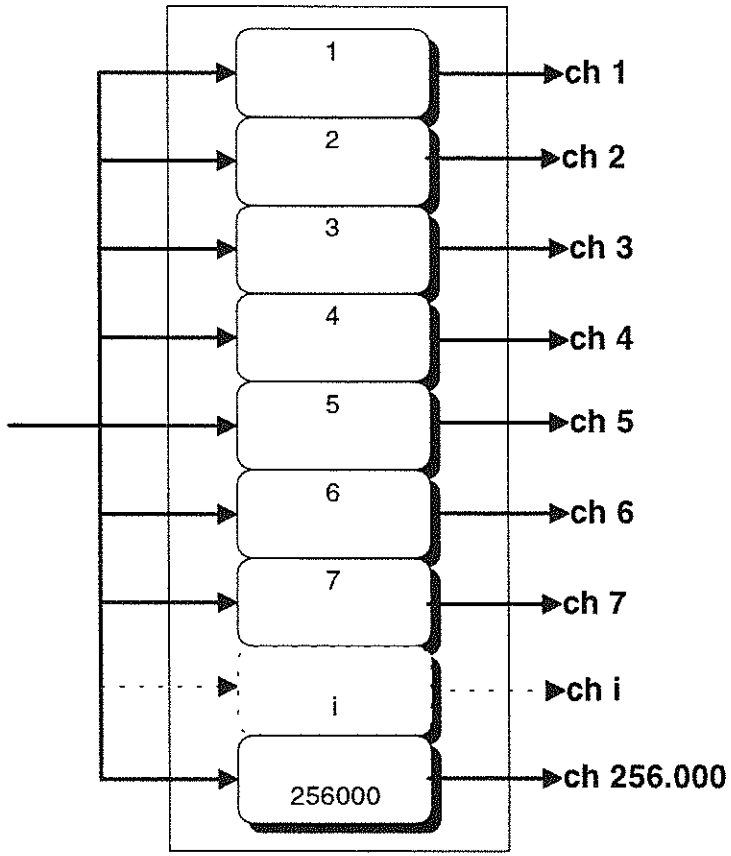


FIG. 35

FIR Filter Bank

Senza entrare nel merito della "decimazione" (sono stati scritti interi volumi su questo argomento), possiamo dire che combinando hardware e software dedicato e` possibile arrivare alla architetture dei filtri polifasi e banchi di filtri polifasi/DFT, in maniera relativamente semplice. Un filtro polifase in pratica è da considerarsi una struttura in cui un commutatore ideale, in sincronismo con il sampling time, distribuisce" nello stesso modo alle varie celle di filtro i campionamenti in arrivo dal convertitore A/D (Fig.36) (dalla stessa figura si può vedere come il fattore di decimazione dell' esempio sia 3). Sfruttando poi i filtri polifase è possibile costruire banchi di filtri formati da set di M filtri passa banda di eguale larghezza. La risposta del m-esimo filtro e` una versione in banda base dello stesso ma shiftato in frequenza di:

$$mf_s / M$$

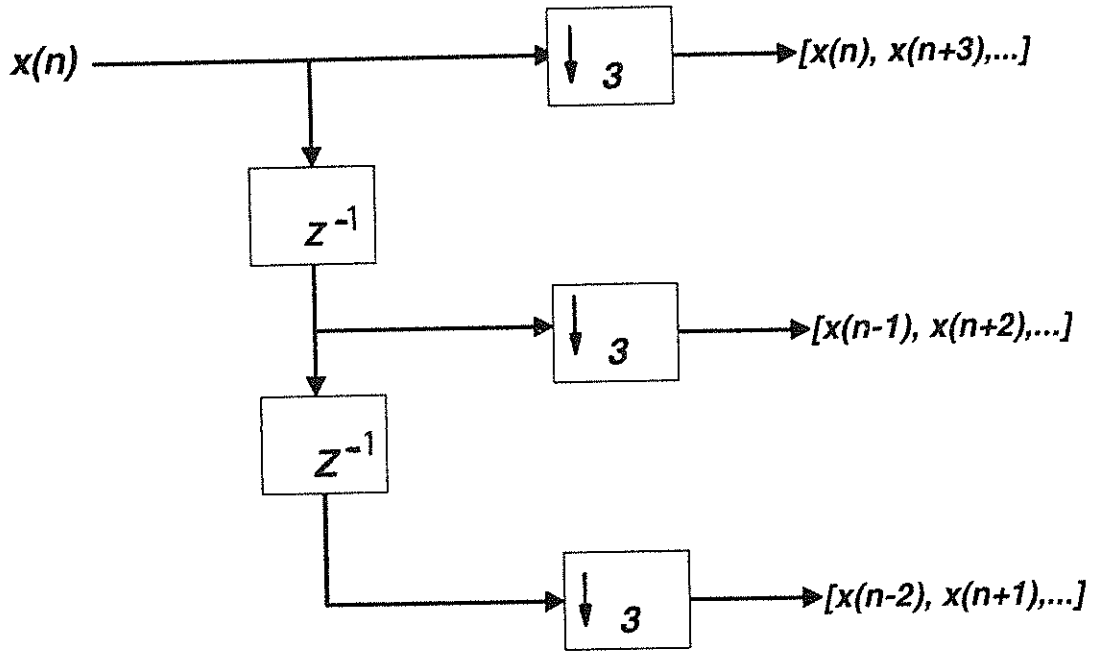
Nella Fig.36 possiamo sostituire a z con $ze^{J2\pi m/M}$ per cui la funzione di trasferimento $H_m(z)$ del filtro si puo`esprimere come $H_m(z) = H_0(ze^{-j2\pi m/M})$ dove $H_0$ e` la funzione di trasferimento del filtro in banda base. Applicando le trasformazioni classiche dei filtri polifase alla funzione di trsferimento H_0 otteniamo:

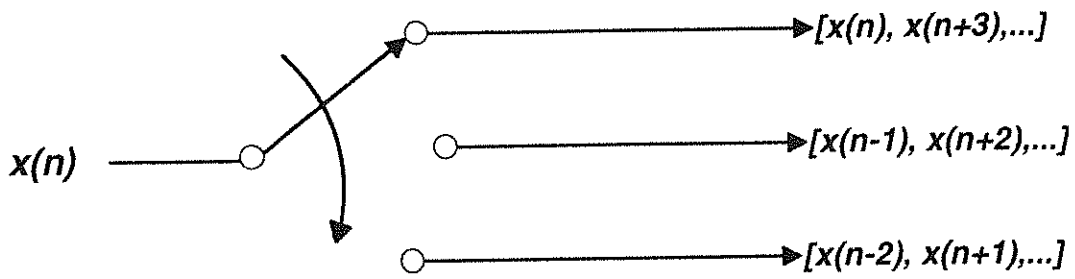
$$H_m(z) = \sum_{k=0}^{M-1} z^{-k} \exp(-j2\pi mk / M) P_k(z^M)$$

dove P_k rappresenta ciascuno dei nuovi filtri decimati (questi risultano essere indipendenti da m). Ponendo $W = \exp(-j2\pi / M)$, possiamo scrivere per l' intero banco di filtri la relazione

$$\begin{bmatrix} H_0(z) \\ H_1(z) \\ . \\ H_{M-1}(z) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & .... & W^0 \\ W^0 & W^1 & .... & W^{M-1} \\ . & . & . & \\ W^0 & W^{M-1} & ... & W^{(M-1)^2} \end{bmatrix} \begin{bmatrix} P_0(z^M) \\ z^{-1}P_1(z^M) \\ . \\ z^{-(M-1)}P_{M-1}(z^M) \end{bmatrix}$$

Come gia` visto in altre sedi, la matrice in W non rappresenta altro che la matrice della DFT mentre i segnali in ingresso della DFT sono le uscite dei filtri polifase (P_k). Trovando la configurazione dei


x(n)
3
[x(n), x(n+3),...]
z -1
3
[x(n-1), x(n+2),...]
z -1
3
[x(n-2), x(n+1),...]


x(n)
[x(n), x(n+3),...]
[x(n-1), x(n+2),...]
[x(n-2), x(n+1),...]

filtri polifasici in banda base, il calcolo in piu` richiesto per sintetizzare l' intero banco di filtri e` solamente quello relativo alla DFT. La struttura dell'intero banco, quindi, e` quella riportata in Fig.36. I grandi analizzatori di spettro digitale costruiti dalla NASA per il progetto SETI (fino a 32 milioni di canali), si basano tutti su questo tipo di architettura (Fig. 37a,b).

Visto che lo spettrometro usa per il calcolo della FFT il chip set della Sharp LH9124/LH9320 e vista la enorme velocita` di questi ultimi, sara` opportuno, in appendice, effettuare alcune considerazioni sul come approntare un banco di filtri.

Ritornando all'upgrade dello spettrometro una configurazione possibile a questo punto potrebbe essere quella riportata in Fig.38. Il gruppo A/D (Ultra ADC Valley Tech.) e` lo stesso delle versioni precedenti. Segue il blocco (2) composto da un banco di filtri polifasi da 128/256 canali. Gli spettri in uscita da questo blocco vengono ordinati in una matrice all'interno del blocco (3) costituito da due pagine di memoria in modo tale che mentre una e` in fase di riempimento, sull' altra si esegue la trasposta della precedente matrice di dati. A questo punto il blocco (4) esegue di nuovo la FFT sulle righe della matrice trasposta mentre il (5) esegue il modulo quadro su ogni canale e la media di un certo numero di spettri. Il blocco (8) esegue il controllo dello spettro nel senso di controllare se e in quale porzione dello spettro esistono righe "sospette" da sottoporre ad ulteriori indagini. Segue una scheda Alfa PC per la ricomposizione dello spettro ed eventuali successive elaborazioni.

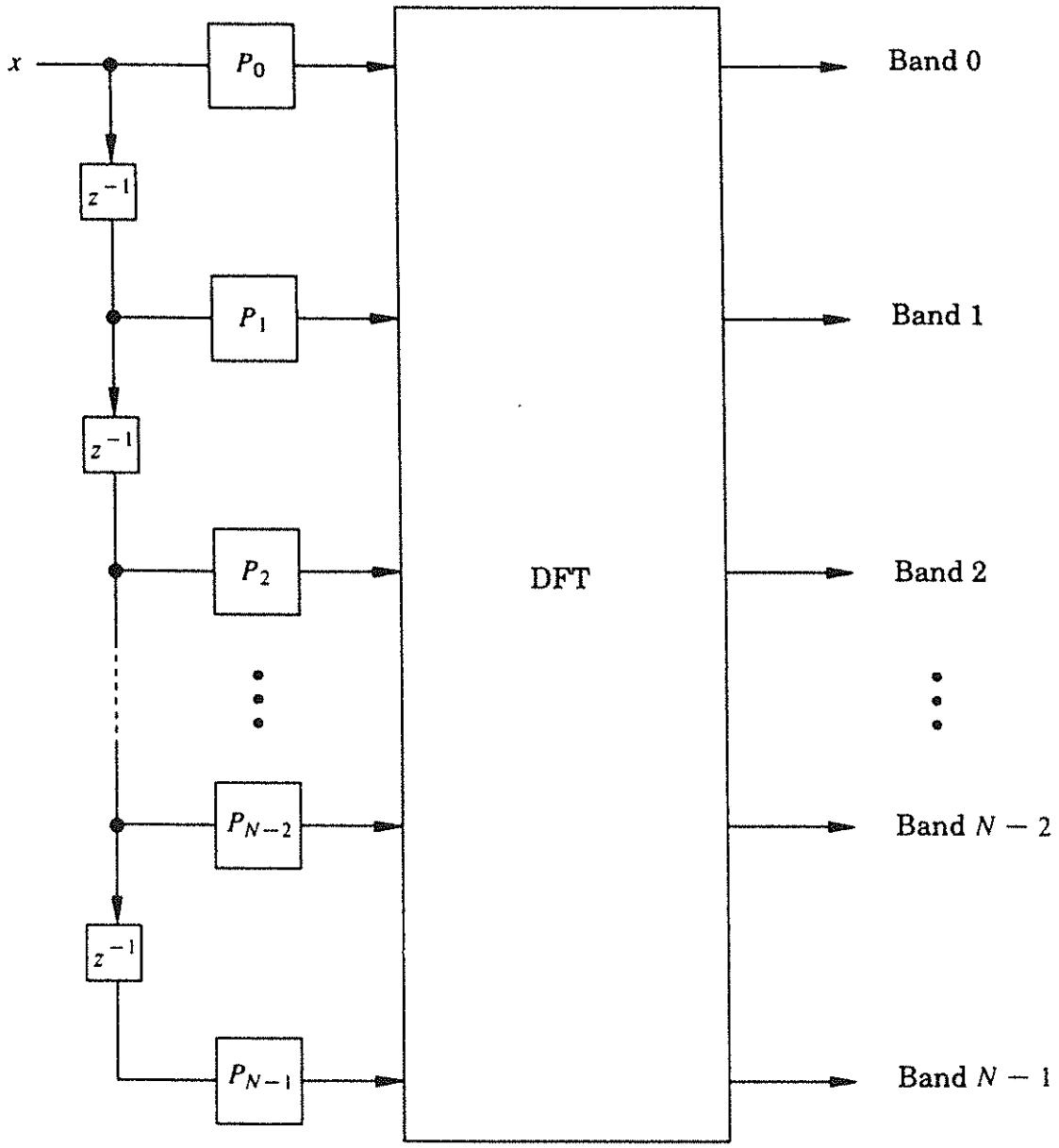


FIG. 36

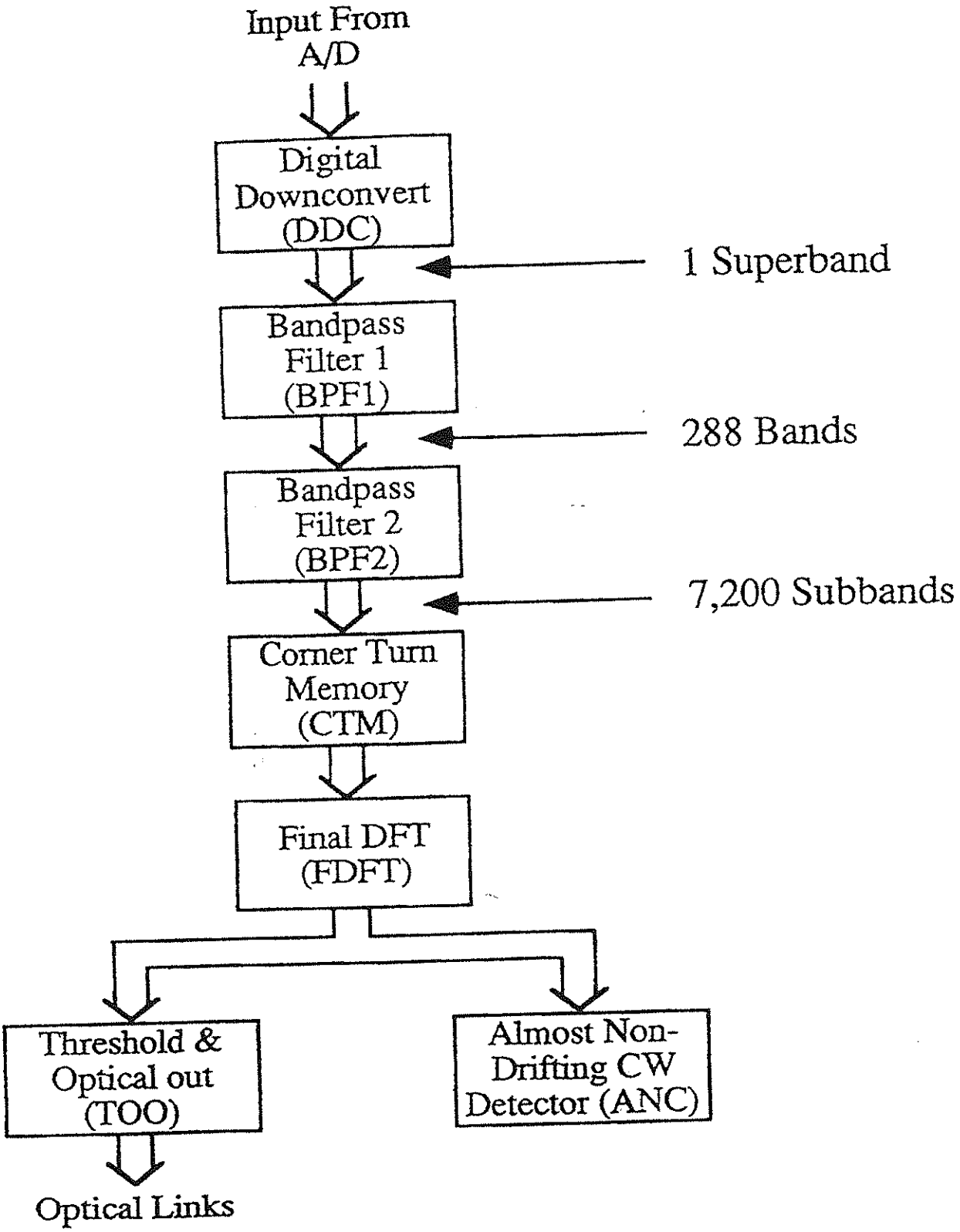

Input From A/D
Digital Downconvert (DDC)
1 Superband
Bandpass Filter 1 (BPF1)
288 Bands
Bandpass Filter 2 (BPF2)
7,200 Subbands
Corner Turn Memory (CTM)
Final DFT (FDFT)
Threshold & Optical out (TOO)
Almost Non-Drifting CW Detector (ANC)
Optical Links

FIG. 37a

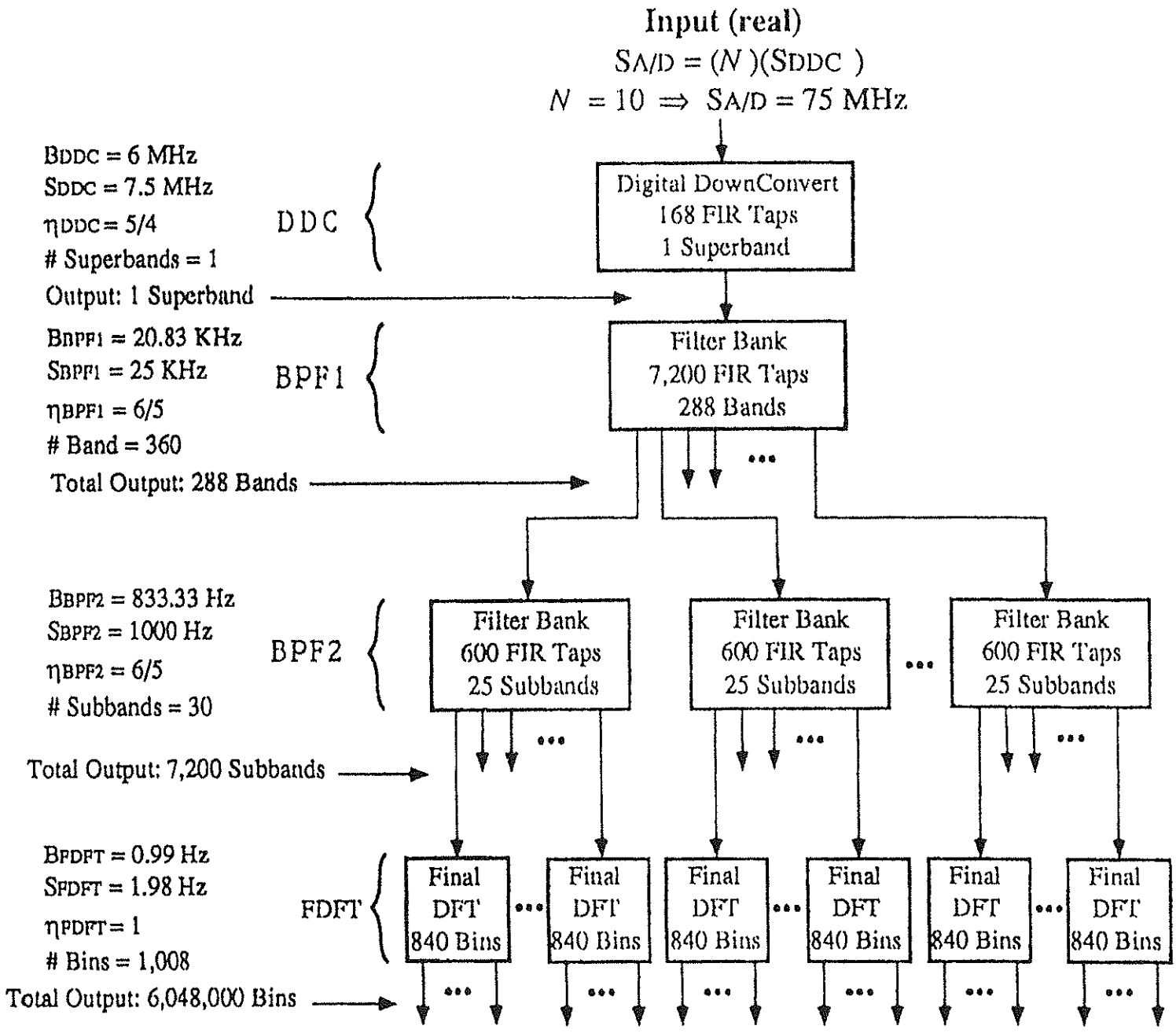

Input (real)
SA/D = (N)(SDDC)
N = 10 ⇒ SA/D = 75 MHz
BDDC = 6 MHz
SDDC = 7.5 MHz
ηDDC = 5/4
Superbands = 1
Output: 1 Superband
DDC
Digital DownConvert
168 FIR Taps
1 Superband
BBPF1 = 20.83 KHz
SBPF1 = 25 KHz
ηBPF1 = 6/5
Band = 360
Total Output: 288 Bands
BPF1
Filter Bank
7,200 FIR Taps
288 Bands
BBPF2 = 833.33 Hz
SBPF2 = 1000 Hz
ηBPF2 = 6/5
Subbands = 30
BPF2
Filter Bank
600 FIR Taps
25 Subbands
Filter Bank
600 FIR Taps
25 Subbands
Filter Bank
600 FIR Taps
25 Subbands
Total Output: 7,200 Subbands
BFDFT = 0.99 Hz
SFDFT = 1.98 Hz
ηFDFT = 1
Bins = 1,008
Total Output: 6,048,000 Bins
FDFT
Final DFT 840 Bins
Final DFT 840 Bins
Final DFT 840 Bins
Final DFT 840 Bins
Final DFT 840 Bins
Final DFT 840 Bins

Fig. 37b

C:\ABC\MEGACH.AF3

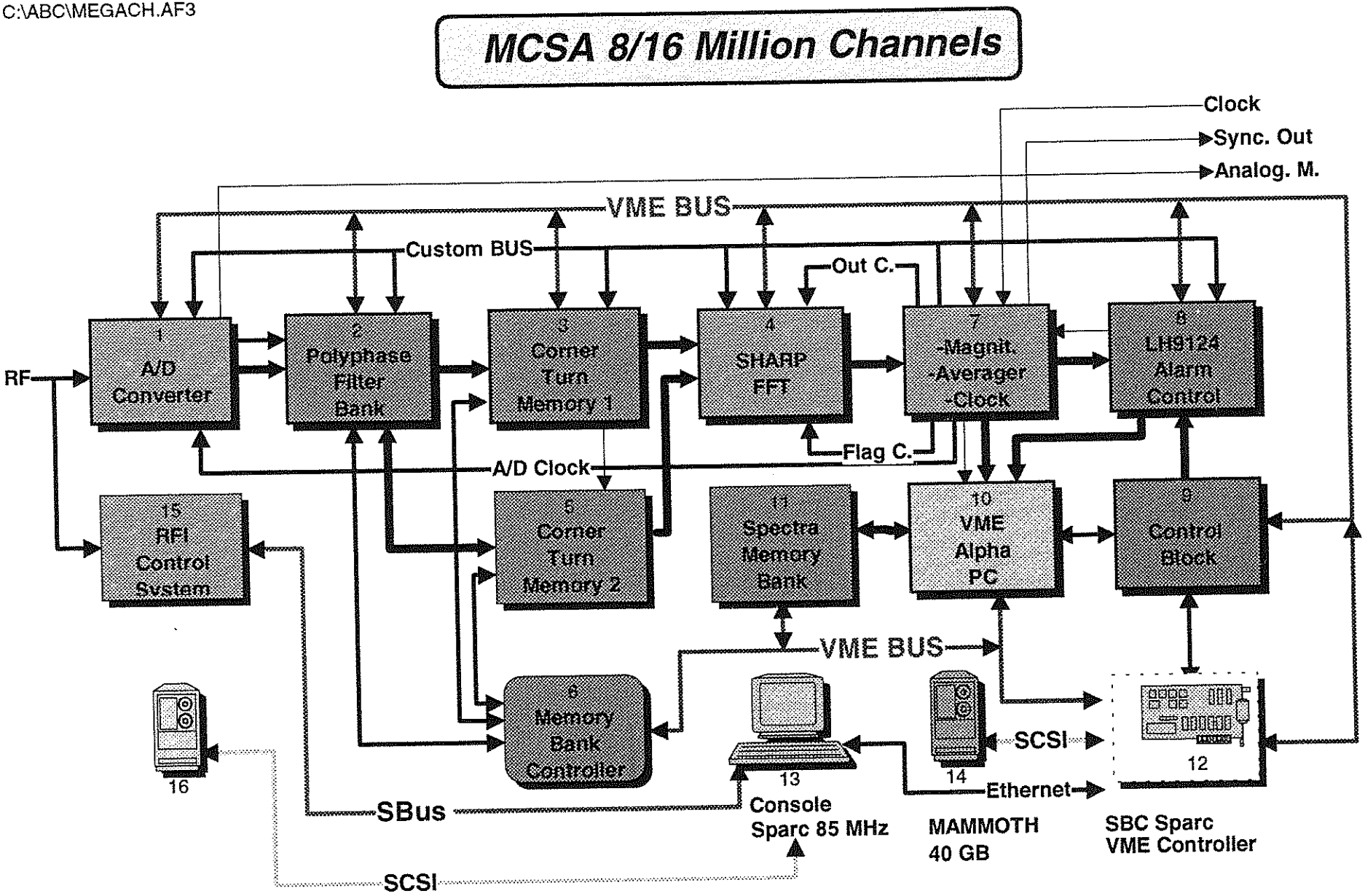


FIG. 38

10 - REFERENCE

[1] "Generatore digitale di segnali programmabile (DDS)"
IRA 184/94
C. Bortolotti, S. Montebugnoli

[2] "Interfaccia di I/O del sistema Pulsar di Medicina
IRA 153/91
A. Maccaferri, A. Cattani

[3] "Sistema di digitizzazione programmabile da 50 Ms/s max, a 6 bit"
IRA 186/94
A. Cattani, S. Montebugnoli

[4] SHARP Data Sheet (1993).

[5] "Sistema di interfacciamento parallelo in ambiente VME"
IRA 185/94
A. Cattani, A. Maccaferri, S. Montebugnoli

[6] "Introduzione al chip set LH9124/LH9320. Proposta di applicazione ad una scheda Multi DSP per analisi On-Line di segnali radioastronomici"
IRA 197/95
S. Montebugnoli, C. Bortolotti, S. Buttaccio, A. Cattani, N. D'Amico, G. Grueff, A. Maccaferri, G. Maccaferri, A. Orfei, M. Roma, M. Tugnoli, G. Tuccari, S. Tassinari, P. Vendruscolo

[7] "Phase Shifter a stato solido realizzati con tecnica strip-line per la formazione del tracking beam E/W nel radiotelelscopio Croce del Nord"
IRA 203/95
M. Roma, M. Tugnoli, C. Bortolotti, S. Montebugnoli.