Software di controllo e acquisizione dati di un Digital Tone Extractor su scheda Pentek

G. Comoretto[1], A. Tani[2]

1) Osservatorio Astrofisico di Arcetri INAF
2) Istituto di Radioastronomia INAF sez. di Firenze

Arcetri Technical Report N° 6/2006
Firenze, Dicembre 2006

Abstract

Il gruppo di radioastronomia dell'Osservatorio di Arcetri ha realizzato un ricevitore digitale coerente (Digital Tone Extractor, o DTE), per l'acquisizione e tracking della portante di una sonda (in particolare la Cassini) usata come probe in esperimenti di Radio Science. In questo rapporto si descrive la realizzazione del relativo software di controllo, e delle prove di stabilità effettuate sullo strumento. In particolare dai risultati si evidenzia che la scheda utilizzata per il ricevitore non programma l'oscillatore locale con una fase ripetibile, ed in tal modo la ricezione coerente non è possibile.

1 Introduzione

Un ricevitore per Radio Science è uno strumento in grado di ricevere un segnale radio coerente, generalmente costituito dalla portante del canale di telecomunicazione di una sonda spaziale.

Il ricevitore deve estrarre informazioni costituite dalla fase assoluta (riferita cioè ad uno standard di frequenza assoluto, come un maser ad idrogeno), ampiezza, e/o distribuzione spettrale del segnale ricevuto.

Attraverso tali informazioni è possibile [1]:

- determinare la dinamica della sonda, con una accuratezza pari ad una frazione della lungheza d'onda impiegata, da cui estrarre parametri significativi del campo gravitazionale di corpi incontrati dalle sonde nei loro *fly-by*: massa di asteroidi, momenti di quadrupolo e successivi di pianeti e satelliti
- caratterizzare il mezzo attraversato (plasma interplanetario o corona solare) determinando le caratteristiche di scattering, e quindi la distribuzione di dimensioni, di anelli planetari.
- misurare effetti relativistici (dilatazione temporale nel campo gravitazionale solare, onde gravitazionali)

È indispensabile che il sistema di acquisizione sia in grado di preservare con accuratezza la coerenza di fase del segnale ricevuto. Un sistema di ricezione per osservazioni interferometriche a lunghissima base (VLBI) garantisce che questa coerenza venga preservata fino al campionamento e digitalizzazione del segnale radio. Inoltre è presente nel sistema un generatore di tono con fase nota, che viene utilizzato per calibrare la parte a radiofrequenza.

Il segnale digitalizzato deve essere ulteriormente convertito di frequenza utilizzando un ricevitore ed un filtro digitali. Occorre quindi che il segnale di oscillatore locale di questo ricevitore, oltre ad essere stabile in frequenza, abbia una fase nota con precisione. Un sistema digitale permette in principio di generare questo segnale in modo predicibile, ma occorre caratterizzarne e calibrarne accuratamente il comportamento.

Il segnale ricevuto ha una frequenza rapidamente variabile, per effetto del moto relativo della Terra e della sonda, e l'oscillatore locale deve essere continuamente riprogrammato, mantenendo una fase predicibile, in modo da seguire queste variazioni. Di solito la frequenza attesa è prodotta all'interno del software di controllo mediante un modello che ha come input un vettore di parametri forniti per un determinato intervallo di tempo dal JPL.

I primi sistemi di registrazione impiegavano un circuito PLL per inseguire il segnale radio. L'uscita dell'oscillatore locale, agganciato in fase alla portante ricevuta, ne costituiva quindi una replica, e poteva venir registrato al suo posto. Questi sistemi erano detti di *registrazione closed loop*. I sistemi attuali invece registrano il segnale convertito, e filtrato su una banda molto stretta (*registrazione open loop*). In questo modo piccoli errori di inseguimento della frequenza della portante non influiscono sul segnale registrato, ed è possibile affinare in fase di riduzione dati l'accuratezza della misura della sua fase.

Come sottoprodotto dell'osservazione, lo strumento produce, in tempo reale, dei residui di frequenza, che rappresentano l'errore nella stima della frequenza ricevuta. Questi residui, opportunamente filtrati in tempo reale, costituiscono un segnale errore che può essere impiegato per correggere, manualmente o in modo automatico, il sistema di inseguimento della frequenza ricevuta. In questo caso si parla di inseguimento di frequenza *closed loop* (da non confondere con il tipo di registrazione, che è comunuque sempre *open loop*). Nel sistema oggetto di questo rapporto, per semplicità, non si è previsto questo tipo di correzione, ma le predizioni di frequenza sono basate unicamente sui modelli orbitali della sonda (inseguimento *open loop*).

Un ricevitore interfacciato ai sistemi VLBI di Medicina e Noto è stato realizzato alla fine degli anni '80 ed é stato impiegato con successo fino a quanche anno fa. La grossa limitazione nella banda passante istantanea (10 Hz) lo rende tuttavia non adatto ad osservazioni con sonde in orbita intorno a pianeti, o a frequneze elevate (32 GHz), né quando sia presente un forte allargamento della riga dovuto a scattering.

Il ricevitore e il software testato in questa contesto si riferiscono a sistemi che si interfacciano allo standard VLBI ad es.delle antenne italiane di Medicina e Noto. Un ricevitore di questo tipo (Digital Tone Extractor) è stato realizzato alla fine degli anni 80 dal gruppo di Radioastronomia dell Osservatorio di Arcetri ed utilizzato con successo per tracking della sonda Ulysses[2].

Per ovviarne le limitazioni (principalmente una banda passante istantanea massima di 10 Hz), è stato proposto uno schema basato su tecnologie digitali in [3]. In questo rapporto si descrive il software di controllo di un DTE analogo a quello descritto in [3] basato su un a scheda Digital Receiver Pentek 4271.

2 Descrizione del Sistema e ambiente di Lavoro

Il ricevitore digitale sviluppato ad Arcetri è basato sullo standard VME, sia per la meccanica che per l'interfacciamento delle schede. È composto da una CPU di controllo, una scheda DSP (Pentek 4284), che ospita una daugtherboard contenente 4 ricevitori digitali, più alcune schede di servizio.

Per semplicità, la parte di interfacciamento al radiotelescopio è costituita dalla scheda Deformatter del precedente DTE, che riceve e decodifica uno stream digitale formattato dal terminale MarkIII (o compatibile). Siccome il sistema deve poter ricevere due canali di frequenza indipendenti (per misure su mezzi dispersivi e/o per correggere gli effetti dispersivi del mezzo), sono presenti due schede deformatter indipendenti.

La scheda contenente i 4 ricevitori digitali, mod. Pentek 4271 è alloggiata su una motherboard VME Pentek 4284, basata su un DSP Texas C040. Una scheda di acquisizione digitale generica permette a un processore su bus VME di leggere le informazioni di tempo decodificate dai deformatter. Il sistema colloquia, tramite un'interfaccia di rete nella CPU di controllo, con un generico computer (PC sotto sistema operativo Linux) che contiene i programmi di lavoro, gestisce l'interfaccia utente e la memorizzazione dei dati.

2.1 Hardware

La scheda Pentek 4271 contiene 4 ricevitori digitali realizzati utilizzando il chip Greychip GC1011A prodotto dalla Texas. Questo componente implementa un convertitore a banda laterale singola completamente numerico, con una frequenza massima di campionamento di 40 MHz (banda passante massima di 20 MHz), e frequenza massima di uscita di 1/32 di quella in ingresso. È composto da un oscillatore numerico (DDS) con risoluzione di 24 bit, un mixer complesso, un primo filtro (comb filter) con fattore di decimazione selezionabile tra 1/16 e 1/16384, un secondo filtro a mezza banda per la correzione della banda passante e per la reiezione dell'aliasing, e un circuito di formattazione dell uscita, che consente di scegliere tra uscita reale o complessa e rinormalizza il segnale convertito. La banda passante finale è pari ad un quarto della frequenza di campionamento in ingresso, divisa per il fattore di decimazione utilizzato nel comb filter.

Il Greychip analizza un segnale campionato con una risoluzione di fino a 12 bit, in modo sincrono con il clock di campionamento. Un segnale di sincronismo esterno permette di sincronizzare l'oscillatore digitale e le operazioni di decimazione, in modo da garantire la coerenza di fase del segnale convertito. Ogni ricevitore è dotato di una FIFO di 512 campioni, che permette di leggere i risultati in modo asincrono.

La scheda si interfaccia al DSP Texas tramite un bus proprietario della casa costruttrice (*MIX bus*).

Nella configurazione hardware utilizzata, il segnale di sincronismo viene connesso all'impulso di 1 picco al secondo ricavato dallo stream di dati nel deformatter. L'impulso marca, con un'accuratezza in ultima analisi legata allo standard atomico utilizzato, il campione acquisito allo scoccare di ogni secondo. I quattro ricevitori digitali impiegano questo segnale per sincronizzare ogni cambiamento di frequenza dell'oscillatore locale, e per azzerare la fase di questi all'inizio di ogni secondo. La frequenza dell'oscillatore locale può quindi essere specificata in modo asincrono in un qualsiasi istante del secondo precedente a quello in cui deve essere applicata, e il suo aggiornamento avviene automaticamente.

2.2 Software

Il software nel suo complesso consta di tre componenti, eseguiti rispettivamente nel DSP (software di basso livello), nella CPU VME (*Server*), e nel computer utente (*Client*).

La parte di basso livello gestisce direttamente l'hardware mediante il DSP della scheda 4284 Pentek. Tramite *MIX bus* imposta i registri della scheda 4271 per programmare i 4 ricevitori digitali Greychip, ne legge i risultati e li trasmette al *Server*.

Il secondo modulo funge da interfaccia tra i comandi di alto livello forniti dal *Client* e lo strato di basso livello, genera le frequenze per gli oscillatori locali, temporizza l'osservazione ed esegua una prima formattazione dei dati raccolti. Il *Server* è concepito per essere utilizzato su un SBC (Singol Board Computer) dotata di sistema operativo Real Time commerciale VxWorks.

Il *Client* comunica con l'utente, che specifica in forma interattiva o attraverso dei file di parametri i parametri fondamentali dell'osservazione, come le predizioni di frequenza, il tempo di integrazione, la frequenza dell'oscillatore locale dei ricevitori radio usati

Il programma *Client* ha il compito di registrare una banda di frequenza in modo cieco intorno alla frequenza prevista, senza seguire eventuali errori nelle predizioni di frequenza. Il programma dovrebbe visualizzare gli offset tra la frequenza del segnale osservato e le predizioni, in modo tale che quando la frequenza si avvicina al bordo della banda utile l'operatore possa correggere manualmente il modello di predizione. [4]. Nella prima versione qui documentata, che costituisce essenzialmente un testbench per i rimanenti componenti, questa funzione non è presente, e anche la trattazione degli errori è ridotta al minimo. Tuttavia l'utente può inserire offset di frequenza in tempo reale durante l'acquisizione. Il *Client* funziona su wokstation SUN o pc con sistema operativo Linux.

Questi moduli sono stati provati utilizzando come segnale un'onda quadra di fase nota, ed esaminando la fase delle armoniche di cui è composta. In questo modo è possibile ricavare i ritardi introdotti dai vari elementi della catena ricevente. Infine dall'analisi dei *residui di fase* ovvero della differenza tra fase ricevuta e fase programmata si valuta se il ricevitore è effettivamente in grado di ricevere coerentemente il segnale della sonda.

Il software è stato modellizzato secondo i metodi della programmazione ad oggetti (OOP). Il software è stato decomposto in *oggetti*, che comunicano tra di loro per mezzo di messaggi. Gli oggetti appartengono a classi, tipi astratti caratterizzati da un comportamento ben definito. Il funzionamento di ognuno dei componenti elencati sopra è descritto tramite *casi d'uso*, situazioni tipiche di eventi che mettono in luce le relazioni dei vari oggetti.

Non essendo disponibile un compilatore di un linguaggio ad oggetti (C++), l'implementazione del programma è stata fatta utilizzando il linguaggio C standard, con alcune convenzioni per permettere di distinguere gli oggetti. Ad esempio i metodi di una classe hanno come prefisso il nome della classe stessa, e come primo argomento un puntatore all'oggetto a cui si riferiscono.

3 Il Server

3.1 Introduzione

Il *Server* riceve comandi e parametri dall' interfaccia utente *Client* in forma interattiva e/o attraverso dei file. I parametri specificano una osservazione, mentre i comandi permettono di iniziare e fermare l'osservazione, o compiere altre azioni sul ricevitore.

Il programma è concepito per essere utilizzato su una scheda Motorola MV2100, che è un SBC (Singol Board Computer) basato su processore PPC e dotata di sistema operativo Real Time VxWorks.

Una osservazione è descritta dai seguenti parametri, per ciascuno dei 4 canali in ingresso:

- un modello di predizione di frequenza, specificato con un vettore di 6 parametri. I parametri sono calcolati a partire dalle predizioni di frequenza fornite dai programmi di modelizzazione orbitale della sonda. In base a questo modello, il *Server* calcola in ogni istante la frequenza di cielo attesa per il segnale ricevuto.

- Tempo di integrazione elementare, in μsec. Esso deve essere maggiore di 16 μsec, corrispondente alla minima decimazione che lo strumento supporta. Il sampling rate, e la banda registrata dallo strumento, sono pari all'inverso di questo tempo. Il tempo deve dividere esattamente un secondo.

- Frequenza complessiva degli oscillatori locali, in Hz. Questo parametro serve per determinare la frequenza attesa all'ingresso del ricevitore digitale, nota la freqnza di cielo (determinata dal modello).

- Guadagno della catena di filtri del ricevitore. Va determinato sperimentalmente, in modo da evitare saturazione dei campioni convertiti, ma nello stesso tempo conservare un buon range dinamico. È espresso come una potenza di 2, corrispondente ad un numero di bit che vengono conservati nel risultato.

- Abilitazione di ciascuno dei 4 canali disponibili. I parametri realtivi ai canali disabilitati sono ignorati.

Il *Server* rappresenta uno strato software intermedio tra l'interfaccia utente e il programma di piu' basso livello che gestisce direttamente il ricevitore attraverso il DSP. Il programma da un lato colloquia con il *Client* tramite una socket TCP/IP e dall'altro si interfaccia, attraverso il bus VME, tramite la memoria condivisa Global SDRAM della scheda 4284 al programma di basso livello nel DSP. L'accesso a questa memoria è sincronizzato da un semaforo.

Il programma riceve dal DSP dati e messaggi di errore, e li ritrasmette al *Client*, formattati in modo opportuno.

Una funzione ausiliaria, ma fondamentale, del *Server* consiste nel caricare nella memoria del DSP il relativo programma, all'accensione dello strumento.

Il software è realizzato mediante thread secondo i principi della programmazione concorrente. Sono presenti tre thread, che comunicano tra di loro attraverso code di messaggi:

- Principale: gestisce la comunicazione con la socket, e coordina i rimanenti task. Al suo interno la subroutine **Leggi_Comandi** riceve ed interpreta i comandi dal *CLient*.

- **Leggi_Buffer**: controlla la presenza di dati nel buffer di comunicazione con il DSP del ricevitore digitale.

- **Track**: temporizza l'osservazione ed invia i comandi di programmazione di frequenza al DSP.

Il software è stato sviluppato servendosi della tecnologia *Object Oriented*. Ogni thread è modellizzato come una classe, e i messaggi spediti alle relative code di ingresso corrispondono a messaggi asincroni, che attivano metodi a classe.

A partire dai casi d'uso e dai diagrammi dinamici di sequenza, si sono ricavate le Classi che poi sono state implementate come strutture in C. Nei seguenti capitoli si descrivono i casi d'uso principali del *Server*

3.2 Caso d'uso: Inizializzazione

Durante la fase di inizializzazione, supponendo caricato il programma di basso livello che gestisce il DSP, il *Server*:

- crea la socket lato *Server*, effettua il binding con l'indirizzo del *Server* e si pone in attesa di connessioni TCP dal lato client sulla porta specificata tramite le chiamate *listen* e *accept*.

- Inizializza le code di messaggi in ingresso per i thread **Track** e **Leggi_Buffer**.

- Avvia il thread principale **Track** che programma il ricevitore, scrivendo i comandi opportuni nella locazione di memoria condivisa della scheda 4284.

- Avvia il thread **Leggi_Buffer** che acquisisce i dati dalla scheda 4271 leggendo dal bus VME i dati che il programma di basso livello ha scritto nella memoria condivisa.

- Avvia la subroutine **Leggi_Comandi** che rappresenta l'interprete per i comandi che il *Client* invia dalla socket.

Al termine dell'inizializzazione, il thread principale resta in attesa di comandi nella subroutine **Leggi_Comandi**. La socket di comunicazione è aperta in lettura dal therad principale, e in scrittura dal thread **Leggi_Buffer**.

I messaggi che il *Server* può ricevere dal *Client* sono raggruppati nelle seguenti categorie:

- **ALL_PROG**, che specifica tutti i parametri necessari per programmare ciascun canale (vedi capitolo 5.3).

- **PART_PROG** che permette impostare solamente un nuovo valore di frequenza o guadagno per ciascun canale.

  I parametri di questi comandi vengono trasmessi direttamente al DSP, dopo una opportuna conversione di formato. Ad eccezione della frequenza e del guadagno questi paramentri vengono memorizzati nell'opportuna struttura dati del software del DSP ed effettivamente utilizzati a partire dal successivo START dopo aver fermato con STOP la precedente acquisizione. In altre parole, se si modificano i parametri di un'osservazione mentre questa è in corso, questi influenzeranno solamente l'osservazione successiva.

- Infine tutti i comandi che specificano azioni sono stati raggruppati nella categoria **PROG_SPECIAL**. A ciascuno di questi comandi corrisponde uno specifico caso d'uso:
  - **START**: inizio di una osservazione
  - **STOP**: termine di una osservazione. Il collegamento tra *Client* e *Server* viene interrotto
  - **RESTART** ovvero **(STOP + START)**: il collegamento tra *Client* e *Server* viene mantenuto
  - **KILL**: il programma *Server* termina.

3.3 Caso d'uso: START

Precondizione

La subroutine **Leggi_Comandi** ha ricevuto un comando di START dal *Client*.
Flusso normale degli eventi
il thread **Track** esegue le seguenti operazioni:

- acquisisce il tempo dell'osservazione dall'oggetto *clock* ramite il metodo `clock_gettime()` [1]
- converte i parametri acquisiti nel formato necessario alla programmazione del ricevitore, ovvero come comandi che il programma del DSP può interpretare.
- Invia il comando al software di basso livello scrivendo nella memoria condivisa.
- Calcola la frequenza attesa del segnale al centro del secondo *successivo*, e invia al DSP i comandi necessari a programmare l'osservazione
- invia il comando di START al software del DSP
- legge l'eventuale codice d'errore ritornato nella fase di programmazione hardware relativa ai comandi START e lo segnala al *Client* inviandolo attraverso il thread **Leggi_ Buffer**.

  In caso di assenza di errori acquisisce la frequenza effettivamente programmata (in hw units) e costruisce un header relativo ad un secondo di dati. L'Header è una struttura con i seguenti campi:
  - codice (per distinguere l'header dal dato)
  - secondo (0-59)
  - Numero di campioni in un secondo
  - Tempo di integrazione
  - UTC Time (in secondi)
  - Frequenza effettivamente programmata (in unità hardware).
- Invia l'header cosí formato al thread **Leggi_Buffer**, per segnalare che sono in arrivo dati relativi al tempo UTC indicato.

Terminata la programmazione dello START, il **Track** controlla ciclicamente la coda di messaggi relativa alla comunicazione con la routine **Leggi_Comandi**. Attraverso questa coda, oltre a nuovi comandi, arriva periodicamente un segnale dal thread **Leggi_Buffer** che lo informa, leggendo il campo di tempo presente nei dati letti, che è scoccato un nuovo secondo e occorre riaggiornare la frequenza. [2]

[1] In teoria l'oggetto *clock* dovrebbe sincronizzare l'orologio interno al tempo UTC di un time server utilizzando il protocollo SNTP (chiamata di sistema `sntpcTimeGet()`). Nella versione attuale del programma, tramite questa funzione vengono restituiti i secondi a partire dalla prima chiamata della routine.

[2] la frequenza é aggiornata al rate di 1 Hz

In questo caso il thread esegue le seguenti operazioni:

- aggiorna l'istante di integrazione e la frequenza dalla routine che modella la predizione di frequenza.
- invia la nuova frequenza al DSP, ed esegue il controllo sull'eventuale codice di errore, come nel caso precedente
- forma un nuovo Header relativo al prossimo secondo di programmazione e lo invia alla coda di messaggi del thread **Leggi_Buffer**.

Nel caso si immetta in modo asincrono un nuovo valore (offset) di frequenza, o un nuovo valore di guadagno, il thread **Track** memorizza questi valori in un attributo interno, e lo utilizza nel programmare il sistema nel secondo successivo.

In questo caso d'uso, il thread **Leggi_Buffer** controlla alternativamente la coda di messaggi che lo lega al thread **Track**, e la coda dei dati trasmessi dal DSP nella memoria condivisa.

Se è presente un messaggio dal thread **Track**, controlla se si tratta di un errore (che viene inoltrato immediatamente), un header o un comando speciale. Se viene ricevuto un header, il thread interpreta l'evento come uno START implicito. Il thread quindi:

- immagazzina l'header in un vettore che contiene un header per ogni canale.
- Memorizza il tempo a cui si riferiscono i dati

Se è arrivato un dato, il **Leggi_Buffer** controlla che il tempo associato al dato si riferisca all'header e memorizza i dati ricevuti in un record della struttura che contiene l'header relativo. Nel funzionamento normale sono presenti header relativi a uno o due secondi, il secondo corrente ed eventualmente il successivo. Se il tempo associato al dato letto si riferisce al secondo successivo:

- il record vene accodato ala struttura relativa al secondo successivo
- la struttura dati relativa al secondo corrente viene considerata completata, ed inviata al *Client* tramite la socket
- Il secondo corrente diventa quello della struttura dati in fase di riempimento.
- Viene inviato un messaggio al task **Track**, per segnalare l'avvenuto scoccare del secondo, ed effettuare la programmazione del DSP relativa al prossimo secondo.

Come conseguenza, verrà generato un nuovo header, relativo al secondo seguente, (che diventa il secondo successivo). Lo scambio di messaggi tra i thread è schematicamente illustrato in figura 1 con un diagramma di sequenza.

3.4 Caso d'uso: STOP

Precondizione:

La subroutine **Leggi_Comandi** ha ricevuto un comando di STOP dal *Client*.

Flusso normale degli eventi:

In questo caso il thread **Track**:

- invia il comando opportuno al software caricato nel DSP.
- Invia un messaggio di STOP alla coda di messaggi del thread **Leggi_Buffer**

Nel caso che di ricezione di uno di STOP il thread **Leggi_Buffer**

- Invia alla socket l'Header con l'indicazione che quello sarà l'ultimo secondo acquisito
- Termina l'acquisizione (join).
- Rilascia il semaforo sulla socket

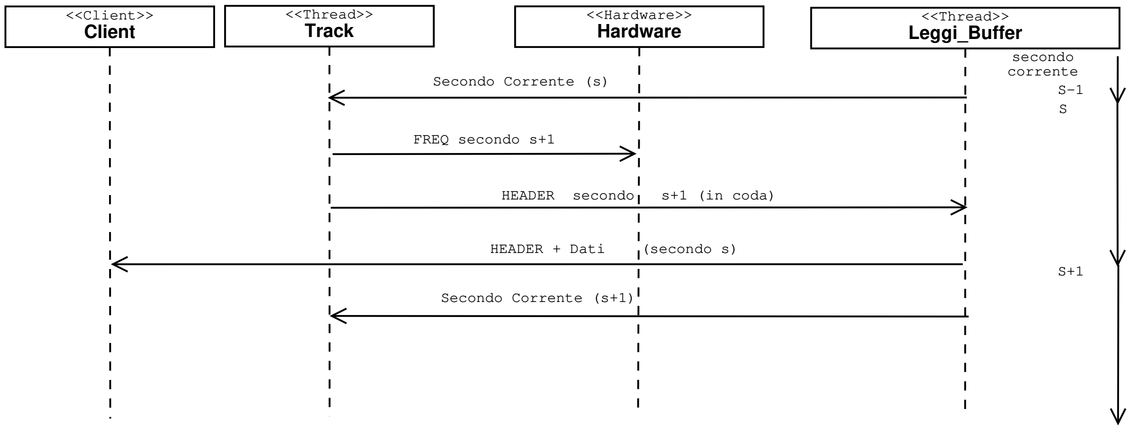

Figura 1: Diagramma di sequenza relativo all'aggiornamento di frequenza

- Esce dal ciclo.

Nel thread principale *Servente*:

- La routine di acquisizione dei comandi termina.
- Il thread attende il rilascio del semaforo sulla socket da parte del thread **Leggi_Buffer**
- La socket viene chiusa e il thread rimane in attesa (listen) di una nuova connessione.
- NB. Entrambi i thread **Leggi_Buffer** e **Track** rimangono attivi.

3.5 Caso d'uso: KILL

Precondizione:
La subroutine **Leggi_Comandi** ha ricevuto un comando di KILL dal *Client*.
Flusso normale degli eventi:
Il thread principale *Servente* esegue le seguenti operazioni:

- Il messaggio di KILL viene inviato ai rimanenti thread, che interrompono immediatamente le operazioni e terminano.
- L'interprete di comandi termina, e il thread attende che i rimanenti thread terminino, utilizzando la chiamata di sistema `join()`
- Tutti gli oggetti creati dal sistema (threads, code di messaggio, semafori, socket) vengono distrutti, e il programma termina

4 Software di Controllo del DSP

4.1 Descrizione del sistema

In questo capitolo viene sommariamente descritto il programma di basso livello, ovvero il software di controllo del viene caricato all'atto dell'esecuzione nella memoria condivisa del DSP C40 della scheda 4284 Pentek.

Questo software comunica con il *Server* attraverso la memoria condivisa della scheda 4284. La casa costruttrice ha sviluppato un software di comunicazione proprietario, che permette di emulare un file system distribuito, ma questo software non è disponibile per la scheda VME utilizzata. Pertanto è stato realizzato un protocollo di comunicazione *ad hoc*, in cui il software di controllo del DSP e il *Server* condividono una piccola area della memoria condivisa Global SDRAM della scheda 4284. La sincronizzazione tra i due programmi avviene tramite una locazione di memoria utilizzata come semaforo. Il protocollo e l'organizzazine della memoria condivisa sono descritti dall'oggetto `Comando`.

Attraverso questa specifica serie di comandi, il software programma i registri della scheda 4271 (Digital Receiver) in modo da impostare frequenza di ingresso del segnale, fattore di decimazione, guadagno dei filtri.

I dati, acquisiti a blocchi di un secondo, vengono marcati con il segnale di tempo proveniente dal deformatter (e quindi dallo stream di dati) (oggetto `Clock_Card`), e immagazzinati in un buffer circolare allocato sempre nella SDRAM condivisa della memoria 4284 (oggetto `Buffer`).

Per semplificare l'analisi del software, il sistema è stato descritto come una macchina a stati con uno stato di **START**, ovvero acquisizione dati in corso e **IDLE**, in cui l'acquisizione dati è sospesa. Siccome l'osservazione è sempre sincronizzata con lo scoccare di un secondo, i comandi di START e STOP modificano un attributo intermedio, e lo stato del sistema viene modificato nel metodo di esecuzione dell'osservazione, all'istante corretto.

Il programma è implementato come un oggetto `Sistema`, con due metodi che idealmente eseguono in modo simultaneo, la ricezione ed esecuzione di comandi e l'esecuzione dell'osservazione. Non essendo disponibile un sistema operativo, queste due funzioni sono realizzate da due metodi, ciascuna delle quali esegue in un tempo massimo prefissato. Il programma principale *Main* è quindi costituito da un *loop* che ciclicamente:

1. controlla se è arrivato un comando e lo esegue
2. acquisisce dati per un decimo di secondo, gestendo in modo opportuno l'inizio e la fine dell'osservazione

L'interprete di comandi non esegue operazioni, ma semplicemente memorizza i parametri del comando ricevuto in un oggetto che rappresenta lo stato dell'hardware (oggetto `Hardware`), o cambia lo stato di funzionamento del programma. Il metodo di esecuzione calcola, all'inizio di un'osservazione, il numero massimo di campioni che ci si aspetta in un intervallo di 0.1 secondi, e termina dopo aver letto questi campioni. In questo modo il massimo tempo di esecuzione del ciclo è appunto di circa 0.1 secondi.

Ogni oggetto del sistema implementa un metodo di inizializzazione, che viene invocato all'inizio del programma. La funzione corrisponde al metodo di creazione dell'oggetto, in linguaggio OOP.

Una descrizione grafica del sistema e degli oggetti che lo compongono è mostrata in fig: 2.

4.2 Casi d'uso

Sulla base della descrizione del sistema possiamo quindi esaminare i *casi d'uso*, secondo la terminologia del linguaggio di modellizzazione UML.

Il caso d'uso fondamentale o requisito primario del sistema è l' **esecuzione della misura**, ovvero l'acquisizione di campioni digitali. Possiamo individuare quattro possibili sottocasi d'uso:

- Immissione di Comandi con parametri (Configurazione)
- Comando di START
- Esecuzione della misura
- Comando di STOP

4.2.1 Caso d'uso Ricezione di un comando

Il caso d'uso comincia quando il *Client* immette un comando con dei parametri; il *Server* scriverà tale comando in memoria condivisa: avremo perciò la seguente sequenza di eventi:

Precondizione:

il sistema ha eseguito l' Inizializzazione (vedi 4.3.2) [3]

[3] per sistema si intende in questo contesto il software del DSP nel suo complesso

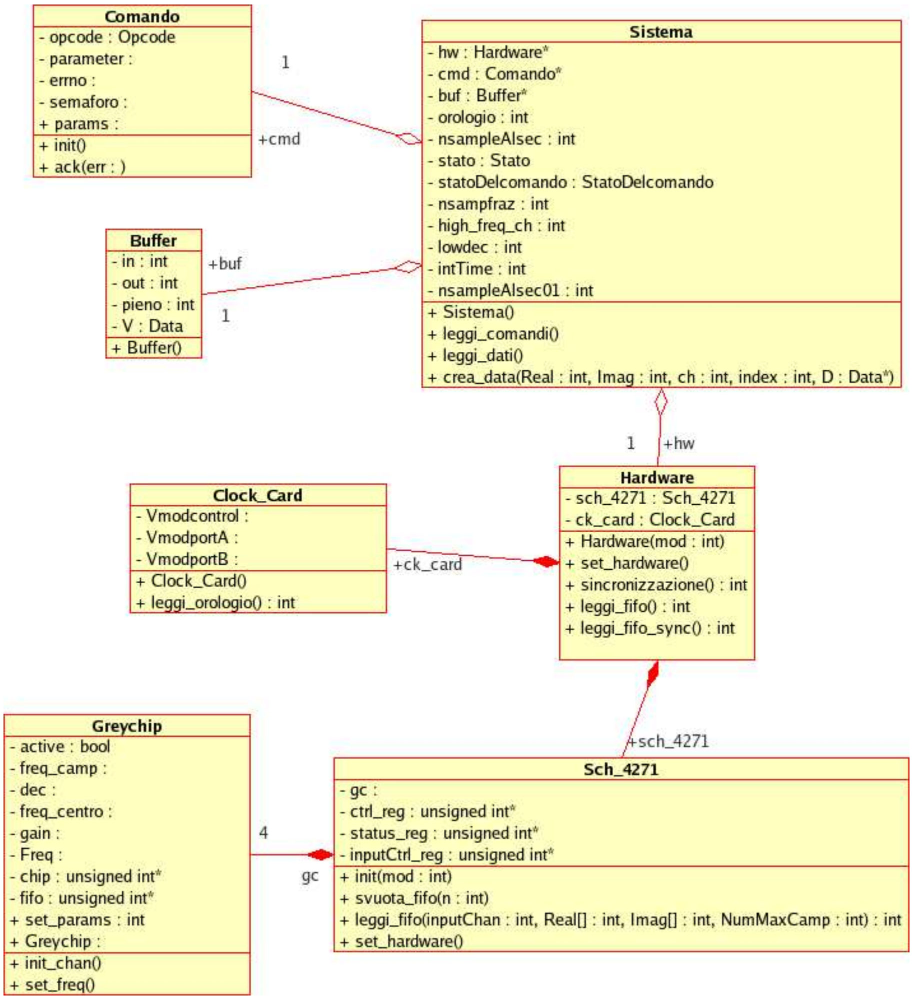

Figura 2: Diagramma ad oggetti del programma di controllo del DSP

Flusso normale degli eventi:
il sistema esegue il protocollo di comunicazione (vedi 4.3.1) ed acquisisce il comando. I parametri del comando sono memorizzati nell'oggetto `Hardware`. I comandi di START e STOP modificano corrispondentemente lo stato del sistema.

I comandi di aggiornamento della frequenza dell'oscillatore locale e di aggiustamento del guadagno causano anche la modifica del relativo registro nel ricevitore digitale. Il chip è programmato in modo da aggiornare la frequenza all'inizio del secondo successivo, garantendo una corretta sincronizzazione della fase dell'oscillatore locale. Il guadagno invece viene modificato immediatamente.

4.2.2 Caso d'uso START

Per quanto riguarda il caso d'uso START abbiamo le seguenti *precondizioni*:

- *Precondizione 1*: il sistema ha eseguito l' inizializzazione.
- *Precondizione 2*: L'utente ha già immesso i comandi con parametri relativi ai canali da utilizzare. I paramentri non specificati assumono il loro valore di default.

Flusso normale degli eventi:
il sistema:

- esegue il protocollo di comunicazione ed acquisisce il comando. Il comando segnala al thread di esecuzione della misura l'inizio del'osservazione, modificando l'attributo di `statoDelComando`.
- Il thread di osservazione programma la scheda 4271 con i valori dei parametri immessi in precedenza.
- Calcola alcune quantità dipendenti dal tempo di integrazione, come il numero di campioni attesi in un secondo, e il massimo numero di campioni consecutivi da acquisire (per limitare il tempo massimo del ciclo principale e quindi la latenza ai comandi in ingresso).
- Esegue la sincronizzazione, attendendo lo scoccare di un secondo (vedi 4.3.3), eliminando gli eventuali campioni ricevuti nel frattempo nella FIFO.
- Aggiorna lo stato del sistema a START.
- Legge le FIFO dei canali abilitati e li scrive nel relativo buffer circolare posto in memoria condivisa (Acquisizione dati).

Flusso eccezionale degli eventi:
se lo stato risulta **RUN** il comando non ha alcun effetto sul sistema.

4.2.3 Caso d'uso STOP

Il flusso degli eventi sarà il seguente:
precondizione 1: Il sistema ha eseguito l' inizializzazione
Flusso normale degli eventi:
Il sistema:

- esegue il protocollo di comunicazione ed acquisisce il comando. Di conseguenza modifica l'attributo `statoDelComando` a **IDLE**.
- Il thread di osservazione attende la fine del secondo corrente acquisendo e memorizzando i dati ricevuti.
- Modifica lo stato a **IDLE**, e termina l'osservazione.

Flusso eccezionale degli eventi:
se lo stato risulta **IDLE** il comando non ha alcun effetto sul sistema.

4.2.4 Caso d'uso: esecuzione dell'osservazione

precondizione: Il sistema è nello stato RUN

La routine di acquisizione legge la FIFO, per un numero di campioni massimo calcolato in modo da:

- limitare il tempo di esecuzione a 0.1 secondi
- non oltrepassare lo scoccare di un secondo

Se lo scoccare del secondo è atteso entro pochi campioni, legge in continuazione il tempo ricevuto dal deformatter, in modo da rilevare lo scoccare del secondo e attribuire correttamente il tempo a ciascun campione.

4.3 Funzioni principali

Per realizzare i tre casi d'uso indicati sopra, si possono individuare le seguenti funzioni principali (non ancora atomiche) ovvero non ancora immediatamente traducibili in metodi di classi:

- Protocollo di comunicazione tra il programma operante sotto VxWorks e il programma caricato nella SDRAM del DSP.
- Inizializzazione.
- Programmazione della scheda 4271 e dei Greychip.
- Sincronizzazione.
- Acquisizione Dati.

4.3.1 Protocollo di comunicazione

Questa funzione permette al sistema di acquisire i parametri e/o i comandi e stabilisce un protocollo di sincronizzazione tra il *Server* e il programma del DSP.

Il protocollo è implementato dall'oggetto `Comando`.

Il comando è costituito da quattro locazioni di memoria a 32 bit della Global SDRAM in cui sono memorizzati:

- *OPCODE*: codice identificativo del comando.
- *Parametri* associati al comando.
- *ERRNO*: codice d'errore ritornato dal DSP .
- *Semaforo*: per sincronizzare l'accesso.

La word che codifica il comando codifica, oltre al codice del comando stesso, il canale cui tale comando si riferisce. Il canale è identificato dai due bit meno significativi della word. Ad esempio i comandi con codici 12-15 specificano la frequenza di centro banda dei canali da 0 a 3, rispettivamente.

La sequenza degli eventi del protocollo sarà la seguente:

Precondizione:

La locazione di memoria che realizza il semaforo è stata precedentemente azzerata dal programma del DSP, per segnalare la disponibilità dell'oggetto al *Server*.

Flusso normale degli eventi:

- il *Server* scrive il codice del comando e i relativi parametri. Quindi setta con un valore diverso da zero (ad es.1) il semaforo
- Il programma del DSP rileva che il semaforo è diverso da zero e quindi legge il comando e i parametri.
- Il metodo del sistema di esecuzione comandi esegue il comando, e rileva eventuali errori. Nel caso di impostazione della frequenza, il valore ritornato come errore è costituito dalla frequenza effettivamente programmata.

- l'oggetto `Comando` imposta il codice di errore (pari a zero nel caso di esecuzione normale senza errori), e azzera il semaforo per segnalare al *Server* il termine dell'operazione di esecuzione del comando.

I comandi e i relativi parametri che possono essere inviati al sistema sono:

- **0** *No operation*: Comando nullo, serve essenzialmente a verificare il funzionamento dell'interfaccia.
- **4** *Start*: Inizia l'osservazione. Il campo relativo al canale viene ignorato.
- **8** *Stop*: Termina l'osservazione. Il campo relativo al canale viene ignorato.
- **28** *Abilitazione del canale*: serve per abilitare o disabilitare ciascuno dei 4 canali della scheda 4271.
- **12** *Frequenza di Centro-banda*: imposta il valore dell'oscillatore locale. Siccome l'uscita del ricevitore è una quantità complessa, questa frequenza rappresente il centro della banda passante registrata. Il valore è espresso in Hz. Il valore effettivamente programmato viene calcolato in base alla frequenza di campionamento, con una risoluzione di 28 bit. Il valore, in unità hardware, effettivamente programmato, viene restituito nel codice di errore. Lo step di frequenza S è pari a:

$$S = \frac{f_c}{2^{28}} \tag{1}$$

  dove f_c è la frequenza di campionamento, tenendo conto che l'accumulatore di frequenza è costituito da 28 bit.
- **16** *Frequenza di campionamento*: Specifica la frequenza di campionamento del segnale in ingresso, espressa in Hz.
- **20** *Fattore di Decimazione*: imposta il valore di decimazione D, espresso come un intero compreso nel range [16:16384]. Il fattore finale di decimazione è pari a $2D$
- **24** *Guadagno*: imposta i possibili valori di guadagno dell'attenuatore digitale posto all'uscita del filtro. Il valore viene specificato come un numero intero di bit, e pertanto il il guadagno può essere incrementato con un passo di 6 dB da 0 fino a 90dB.
- **32** *Lettura frequenza di campionamento*: Questo comando non richiede parametri, e ritorna nel codice di errore l'ultimo valore programmato per la frequenza di oscillatore locale, in unità hardware.

4.3.2 Inizializzazione

L'inizializzazione del sistema è gestito dal costruttore dell'oggetto `System`, che a sua volta chiama in cascata i costruttori di tutti gli oggetti contenuti. In questo capitolo si descrivono le operazioni eseguite durante questa fase.

Flusso normale degli eventi:

- Inizializzazione del bus VME. Viene impostato il modo di trasferimento del bus VME; per permettere al *Server* di accedere alla memoria condivisa sulla scheda.
- Inizializzazione della scheda VMD-IO che fornisce il secondo. Viene impostato il modo di lettura (input diretto senza handshaking) della scheda di IO digitale.
- Inizializzazione registri della scheda 4271 per tutti i canali. In particolare viene abilitata la sincronizzazione hardware dell'aggiornamento del registro di frequenza, che quindi viene automaticamente sincronizzato al segnale di 1 picco al secondo esterno.
- Si setta in **IDLE** la variabile di stato.
- Si svuotano le FIFO di ciascun canale.

4.3.3 Programmazione della scheda 4271 e dei Greychip

La descrizione della scheda 4271 e le programmazione relativa sono implementate dall'oggetto `sch_4271`.

Dopo la fase di inizializzazione, in cui vengono programmate le funzionalità generali, l'oggetto memorizza nei suoi attributi i parametri specificati dagli appositi comandi. Quando l'utente specifica il comando START, l'oggetto provvede a programmare effettivamente l'hardware, attraverso il metodo `set_hardware`.

La fase successiva a quella di programmazione della scheda 4271 può essere definita di sincronizzazione, in quanto prima di cominciare la vera e propria acquisizione dei dati devo attendere lo scoccare del secondo fornito dal segnale di sincronizzazione PPS, tenendo vuote le FIFO dei canali abilitati, tramite il metodo `svuota_fifo`.

La gestione del tempo viene effettuata tramite l'oggetto `Clock_Card`.

Terminata questa fase comincia la lettura dalle FIFO dei canali abilitati e l'inserimento dei dati nel buffer circolare posto nella memoria SDRAM condivisa.

4.3.4 Buffer FIFO

L'oggetto `Buffer` implementa il protocollo per la trasmissione dei dati in uscita.

Si tratta esenzialmente di un buffer ad anello, in cui vengono scritti record di dati corrispondenti ai campioni ricevuti. I dati memorizzati nel buffer sono i seguenti:

- valore dell'orologio al secondo (0-59).
- Indice dei dati all'interno del secondo.
- Tempo in microsecondi dall'inizio del secondo
- Canale del ricevitore
- Valore del campione, parte reale.
- Valore del campione, parte immaginaria.

I campioni relativi ai 4 canali sono memorizzati in ordine sparso, in quanto la frequenza di campionamento può essere differente per i differenti canali. L'ordine temporale per ciascun canale è comunque garantito.

Nello stato di **RUN** l'operatore può ancora impostare i 5 parametri descritti in precedenza, tuttavia tra questi solo la frequenza di centro banda può essere caricata nello stato di **RUN**, infatti il registro accumulatore della frequenza verrà poi caricato in modo sincrono con il segnale PPS che pilota l'AS (Address Strobe) perciò allo scoccare del secondo successivo a quello di impostazione del parametro.

5 Il Client

In questo capitolo viene descritto il *Client* che in questa versione è un programma di prova che testa il funzionamento del *Server*, del programma di basso livello e dell'hardware nel suo complesso.

Il software effettua le operazioni minimali di interprete dei comandi immessi dall'utente in forma testuale e acquisizione dati che vengono scritti su disco.

5.1 Uso del Client

Il *Client* è un programma che funziona in modo testuale, quindi va lanciato da una finestra di testo. Prima di lanciare il programma, l'utente deve lanciare dal computer *host* l'ambiente di sviluppo di VxWorks residente sul *Server*, e, attraverso una sessione remota, caricare il programma di basso livello e lo stesso *Server*.

5.2 Architettura del Client a threads

Il programma è strutturato in due thread concorrenti, rispettivamente per ricevere (da terminale o da file) comandi che vengono inviati al *Server*, e per ricevere dati dal *Server* che vengono formattati e scritti in un file su disco. Per implementare questi thread sono state impiegate le librerie dei threads di Posix.

Allo startup del sistema viene creata la socket di comunicazione, che viene connessa alla rispettiva socket del *Server*. Vengono quindi creati i due thread, *leggi_comandi* e *leggi_dati*.

Il thread *leggi_comandi* legge i comandi dal file `parmfile`, e, una volta esaurito questi, dalla consolle, trasmettendoli alla socket. Il programma termina quando viene ricevuto un comando KILL, attendendo il termine dell'esecuzione del thread *leggi_dati*.

Il thread *leggi_dati* riceve dalla socket i dati provenienti dal *Server*, le informazioni di header all'inizio di ogni secondo, eventuali codici di errore, e infine il messaggio di STOP e KILL che segnalano rispettivamente il termine di un'osservazione e la terminazione del programma. In questi ultimi due casi viene scritto un messaggio sul terminale, e in seguito al messagio di KILL il thread termina.

5.3 Struttura dei comandi per il Client

Il *Client* legge i parametri dell'osservazione dal file di nome fisso `parmfile`. Questo file è costituito da una riga per ogni comando da inviare al *Server*. Ogni riga contiene una serie di campi numerici, di cui i primi due identificano il tipo di comando, ed un codice specifico, seguiti da un numero appropriato di parametri.

I possibili tipi di comandi, e il relativo codice numerico, sono i seguenti:

- **1** programmazione completa di un singolo canale. I successivi campi specificano il canale da programmare, e i relativi parametri: banda passante, frequenza centrale, guadagno, predizioni di frequenza.
- **2** programmazione parziale di offset di frequenza e guadagno, per modificare in tempo reale solo questi parametri
- **3** comandi speciali: START, STOP, KILL, RESTART, che hanno lo stesso significato degli omologhi comandi del *Server*

Il secondo campo permettere di distinguere in maniera univoca i comandi:

- **0** ALL_PROGRAM La cui riga di comando ha la seguente struttura:
  `1 0 ch LO MULT C[0] C[1] C[2] C[3] C[4] C[5] intTime`
  `gain abilita is id`, dove i campi dopo i primi 2 indicano:
  - `ch` il codice del canale da programmare $[0:3]$
  - `LO` la frequenza dell'oscillatore locale in Hz.
  - `MULT` un eventuale fattore di moltiplicazione per le predizioni di frequenza. Nella maggior parte dei casi, i diversi canali trasmessi dalla sonda sono agganciati in fase tra di loro, quindi è possibile usare lo stesso modello di predizioni di frequenza, moltiplicando per un opportuno fattore che dipende dal canale. Ad esempio i canali in banda X e S sono in un rapporto 3/11.
  - `C[0] - C[5]` I parametri del modello di predizione di frequenza.
  - `intTime` Il tempo di integrazione (o intervallo di campionamento) per i dati in uscita dallo strumento, espresso in microsecondi
  - `gain` Guadagno del canale
  - `abilita` 1 se il canale è abilitato, 0 se è disabilitato.
  - `is` Codice numerico identificativo della sonda
  - `id` Giorno dell'anno a cui si riferiscono le predizioni di frequenza.
- **1** FREQ Permette di inserire un offset di frequenza. I parametri sono il numero del canale, e l'offset, in Hz

- **2** GAIN Permette di impostare un nuovo valore per il guadagno di un canale
- **3** START
- **4** STOP
- **5** RESTART
- **6** KILL

Il programma nella sua versione attuale legge dal file di parametri esattamente 4 righe, che corrispondono ai comandi di programmazione relativa ai 4 canali.

Ad esempio un possibile file di parametri è il seguente:

```
1 0 0 0 1 50300 0 0 0 0 0 1000  0 0 0 0
1 0 1 0 1 70020 0 0 0 0 0  100  0 1 0 0
1 0 2 0 1 70020 0 0 0 0 0  100  0 1 0 0
1 0 3 0 1 50300 0 0 0 0 0 1000  0 0 0 0
```

Nell' esempio il *Type* **1** e il *Code* **0** identificano univocamente il comando *ALL_PROGRAM*. Inoltre vengono programmati solo i canali abilitati dal campo *abilita_canale* ovvero i canali 1 e 2 (in quanto i canali 0 e 3 avevano problemi relativi al PLD che gestiva la logica combinatoria necessaria all'acquisizione del segnale di sincronismo da parte del Greychip).

Da notare che i C i coefficienti del modello di frequenza a 6 parametri relativi alle predizioni di frequenza (in Hz), sono tutti nulli ad eccezione del termine C_0: ovvero in fase di test abbiamo operato a frequenza costante. Per semplicità, si assume pari a zero la frequenza degli oscillatori locali della catena di conversione, cioè il modello di frequenza si riferisce alla frequenza attesa del segnale all'ingresso del ricevitore digitale.

Il tempo di integrazione è stato posto pari a 100μs per i due canali utilizzati. Il programma di controllo del DSP utilizza questo valore per calcolare il fattore di decimazione nel ricevitore digitale.

Il parametro *is* viene utilizzato per identificare la sonda utilizzata, e quindi non ha senso in un test. Il parametro *id*, che nell'uso normale identifica il giorno dell'anno a cui sono riferiti i parametri del modello, pure non ha senso per un modello in cui la frequenza è costante.

Dopo aver letto il file di parametri, ed inviato i relativi comandi al *Server*, il programma si mette in attesa di comandi da parte dell'utente.

Se vogliamo ad esempio cambiare in tempo reale la frequenza a Per esempio se si vuole inserire un offset di 10 KHz per il canale 1 al prompt del *Client*, dobbiamo digitare:

`Enter command:` *1*

Dopo aver immesso il corretto campo *Code* il *Client* chiedera' al prompt per i comandi FREQ e GAIN:

`Enter type+command+channel+value:` *2 1 1 10000*

quindi occorre specificare il tipo di comando, il codice, il canale e il parametro relativo.

5.4 Struttura del file di dati

I dati vengono scritti su disco in un file di testo con una struttura corrispondente in modo stretto alla struttura dei dati ricevuti dal *Server*. I record sono di due tipi, record di header, che iniziano con il carattere **#**, e di dati, che iniziano con un campo numerico eventualmente preceduto da spazi bianchi. Ogni riga è composta da 5 campi numerici, in cui il primo e il secondo indicano il tempo, in secondi, e il canale del ricevitore digitale (numerato da 0 a 3).

I rimanenti campi contengono le informazioni:

- Record di header
  - Campo 3: Frequenza programmata, in unità hardware (vedi cap. 4.3.3)
  - Campo 4: Tempo di integrazione, in microsecondi
  - Campo 5: Tempo UTC, in secondi a partire dal tempo di riferimento (nominalmente, dalla mezzanotte del giorno di riferimento *id*, nella versione corrente del *Server*, dall'accensione del sistema)

- Record di dati
  - Campo 3: Indice sequenziale (da 0 a $N-1$, con N il numero di campioni al secondo in uscita dal ricevitore digitale)
  - Campo 4 e 5: Parte reale ed immaginaria del campione, in unità hardware. L'ampiezza dei campioni in uscita al ricevitore è espressa come intero signed a 16 bit, e dipende dal valore del parametro di guadagno utilizzato.

I campioni relativi a canali differenti sono intercalati in modo non predicibile, e vanno separati dal software di analisi. L'indice dei campioni cresce comunque in modo monotono: per ciascun canale i campioni sono sempre registrati in sequenza temporale corretta. Il tempo assoluto associato a ciascun campione è pari al tempo UTC indicato nell'header, più il valore dell'indice moltiplicato per il tempo di integrazione, più un offset costante dovuto al ricevitore digitale, e dipendente dal fattore di decimazione, che è stato valutato nel cap. 6.3.

Un esempio di file di dati, con valori dei campioni puramente indicativi, puó essere il seguente:

Secondo	canale	indice	Parte Reale	Parte Immaginaria
0	1	0	100	20
0	2	0	100	20
0	1	1	120	100
0	2	1	120	100
0	1	2	120	120
0	2	2	120	120
...	...	...	...	...
Secondo	canale	freq_program	int_time	UTC
1	1	53000	40	1002
1	2	53000	40	1002
Secondo	canale	indice	Parte Reale	Parte Immaginaria
1	1	1000	120	35
1	2	1000	120	35

Tabella 1: struttura del file di out

5.5 Condizioni di errore

Il sistema è in grado di individuare alcune condizioni di malfunzionamento, riassunte nei seguenti casi:

1. il buffer circolare in memoria condivisa è pieno cioè il programma *Server* con il thread *Leggi_Buffer* non è riuscito a svuotarlo ad una velocità sufficiente. Il thread *Leggi_Buffer* invia alla socket un messaggio di errore ma l'acquisizione prosegue regolarmente fino ad arrivare alla condizione di STOP.

2. errore hardware dovuto ad un parametro di programmazione errato. L'oggetto `hardware` invia l'errore al thread *Leggi_Buffer* che a sua volta lo invia tramite socket al *Client*. In questo caso la programmazione non è andata a buon fine e va ripetuta.

In caso di errore, il relativo codice viene scritto su terminale, e in un file denominato *logfile*, assieme ad un *Timestamp* dell'evento errore ricavato dal clock interno del PC.

6 Test dello strumento

Tramite questo *Client* è stato possibile programmare lo strumento ed acquisire brevi sequenze d dati, utilizzando come segnale di test un'onda quadra di fase nota. Il segnale di test presenta una frequenza

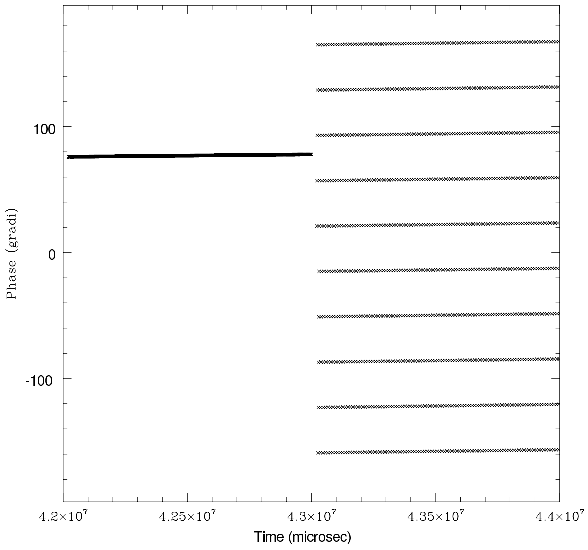

Figura 3: Fase dei campioni in funzione del tempo per un secondo di acquisizione, con 100 Hz di offset

di 10 KHz, e un anticipo noto rispetto al segnale di 1PPS (1 picco per secondo) utilizzato come riferimento temporale. Utilizzando una banda passante ragionevolmente stretta, é possibile estrarre le singole armoniche di questo segnale, e confrontare ampiezza e fase con quella prevista dalla teoria.

In generale sono state utilizzate bande passanti da 100 a 10.000 Hz, e tempi di osservazione dell'ordine di alcuni minuti. L'oscillatore locale del ricevitore è stato programmato a frequenze che differivano da quella dell'armonica in esame da una decina di millihertz (cioè la risoluzione dell'oscillatore digitale), fino a qualche centinaio di Hz.

Con questi test si è voluto valutare sia il corretto funzionamento del sistema, che misurare alcuni parametri non ricavabili dal data sheet dei componenti usati.

6.1 Test di funzionamento

Come primo test, abbiamo osservato un'armonica del nostro segnale, con una banda passante di 1 KHz, aggiungendo un offset di frequenza di 100 Hz e di 30 Hz ad un tempo noto. Il risultato è visibile in figura 3 e 4.

Questa operazione è di cruciale importanza per il tracking della sonda la cui frequenza subisce un drift per effetto doppler e pertanto deve essere periodicamente aggiustata, in sistemi di ricezione a banda stretta come questo, inserendo un opportuno offset di frequenza.

Il sistema risponde correttamente. Mentre prima della modifica la fase è praticamente costante, dopo la modifica ruota con un periodo di rispettivamente 10 campioni per periodo (10 ms) e di 100 campioni ogni tre periodi.

Ripetendo il test sui 4 canali del digital receiver, abbiamo verificato che i canali 0 e 3 non producono nessuna uscita. Il difetto sembra riconducibile ad un mancato funzionamento della logica di sincronizzazione del Greychip. Da un'analisi dell'hardware, il difetto è stato localizzato in un componente guasto nella scheda. La ditta costruttrice non ha tuttavia riconosciuto il difetto durante il periodo di garanzia della scheda, imputandolo ad un nostro errore di programmazione. Dopo l'individuazione del guasto hardware, ci ha preventivato un costo di riparazione elevato, e proposto di riparare da noi la scheda, cosa dificoltosa per l'uscita di produzione dei componenti utilizzati. I test sono quindi proseguiti utilizzando solamente i canali 1 e 2 del digital receiver.

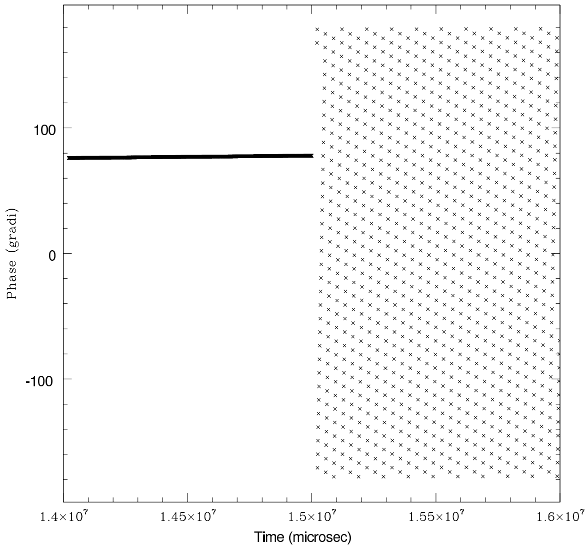

Figura 4: Fase dei campioni in funzione del tempo per un secondo di acquisizione, con 30 Hz di offset

6.2 Test di velocità

Abbiamo quindi provato a verificare la massima velocità del sistema, e quindi la massima banda passante complessiva utilizzabile.

Utilizzando un *Client* funzionante sotto una SUN e con il *Server* VxWorks caricato su una CPU Heurikon MC 68030 si sono effettuate prove dell'ordine del minuto con tempi di integrazione da 1 msec fino a 350 μ sec corrispondenti a bande passanti da 1 a 3 KHz circa, prima di evidenziare perdite di dati. Sommando i due canali ottengo quindi un data rate complessivo massimo da 2K sample/s a circa 5.7 Ksample/s di acquisizione. Oltre questo valore il *Server* non riusciva a svuotare il buffer della memoria condivisa e ad inviare i campioni via socket al *Client*, causando una perdita di dati.

Con una CPU più veloce, overo un PPC 603 su scheda MV 2100 Motorola il *Server* veniva velocizzato fino ad arrivare ad acquisire 40K sample/s complessivi. Oltre questo valore il C40 residente nella scheda 4284 Pentek non riusciva a programmare correttamente il ricevitore digitale, anche se il *Server* non perdeva campioni almeno fino a 50 KHz.

6.3 Calibrazione del Ricevitore

Data la insufficiente documentazione del produttore relativamente alla rotazioni di fase deterministiche introdotte dal ricevitore, abbiamo dovuto stimarle misurando la fase del segnale ottenuto con in ingresso un segnale di test di fase e frequenza nota, in funzione sia della sua frequenza che di quella dell'oscillatore locale.

Come si è detto, il segnale di test consiste in un'onda quadra di fondamentale pari a 10 KHz ritardata di 0.5 μs (2 campioni) rispetto al segnale di riferimento di tempo.

In tal modo si sono valutati i ritardi di fase introdotti dal filtro FIR-decimatore, e dall'oscillatore locale.

Supponendo di avere in ingresso un' onda quadra di frequenza fondamentale f_1, e indicando con f_i la sua armonica di ordine i, ritardata di un tempo t_0 rispetto al riferimento dei tempi: si può dimostrare che lo sviluppo in serie di Fourier contiene solo armoniche dispari e che la fase di tale segnale è:

$$\phi_{0i} = -\pi(2f_it_0 + 1/2) \tag{2}$$

Dopo il battimento con l'oscillatore locale una generica armonica puo' essere scritta nella forma seguente:

$$\cos(2\pi f_it + \phi_{0i})\exp(+2\pi f_L(t - t_{0l})) \tag{3}$$

dove f_L e' la frequenza dell'oscillatore locale e t_{0l} e' l'istante in cui la fase dell'oscillatore locale viene azzerata.

Se t_f è il ritardo introdotto dal filtro t_f che segue il mixer, la (2) diventa:

$$\cos(2\pi f_i(t - t_f) + \phi_{0i}) \exp(+2\pi f_L(t - t_f - t_{0l})) \tag{4}$$

Dopo il mixer, supponendo che sia f_L positiva, la componente a frequenza somma viene filtrata e percio' la fase complessiva (in radianti) è data da:

$$\phi = 2\pi f_L(t - t_f - t_{0l}) - (2\pi f_i(t - t_f) + \phi_{0i}) \tag{5}$$

dove ϕ_{0i} e' data dalla (1), da cui otteniamo:

$$\phi = 2\pi(f_L - f_i)(t - t_f) - 2\pi f_L t_{0L} - \phi_{0i} \tag{6}$$

Sostituendo ϕ_{0i} ed esprimendo la fase in cicli, otteniamo:

$$\phi = \frac{1}{4} + (f_L - f_i)(t - t_f - t_{0L}) - f_i(t_{0L} - t_0) \tag{7}$$

Essendo t_0 nota, e f_L, f_i parametri modificabili, questa relazione ci consente di valutare i parametri di interesse.

Per valutare t_{0L} e t_f abbiamo sintonizzato il ricevitore sulla fondamentale ($f_L = 10$ kHz, figura 5), sulla frequenza 70 KHz (7-ma armonica, $f_L = 10$ KHz, figura 6), e infine impostando l'oscillatore locale con un offset di 30 Hz ($f_L = 70030$ Hz). Per via dello step finito con cui può essere sintonizzato l'oscillatore locale, l'effettivo valore dell'offset di frequenza è circa $+5.36$ mHz, -7.15 mHz, e 30.03786 Hz rispettivamente.

La quantità t_{0L} può essere stimata direttamente, a meno di errori di arrotondamento, confrontando la fase estrapolata a $t = 0$ delle misure a 10 e 70 KHz.

Analizzando la (7) osserviamo che, per valori di f_L molto vicini a f_i e $t \approx 0$, avremo che il termine in $(f_L - f_i)$ è trascurabile rispetto al termine $f_i(t_{0L} - t_0)$, per cui la (6) diventa:

$$\phi \approx 0.25 + f_i t_0 - f_L t_{0L} \tag{8}$$

Siccome ϕ e' adesso funzione solo della frequenza f_i possiamo determinare la quantita' incognita $(t_{0L} - t0)$ variando f_i e calcolandone il rispettivo valore di $\phi(f_i)$.

Dai residui di fase per i casi con f_l pari a 10 e 70 kHz, abbiamo ottenuto:

$$\phi(f_{70Hz}) = -9deg \tag{9}$$

$$\phi(f_{10Hz}) = 76deg \tag{10}$$

quindi la differenza tra le fasi per i due valori di frequenza vale (in cicli):

$$\Delta\phi = \phi(f_{70Hz}) - \phi(f_{10Hz}) = -\frac{76 + 9}{360} = 0.236 \tag{11}$$

Risolvendo per t_{0L} ottengo

$$-(t_{0L} - t_0) \approx -3.93\mu s \rightarrow t_{0L} = 4.43\mu s \tag{12}$$

Una volta ricavato il termine $t_{0L} - t_0$, utilizzando un'osservazione in cui la differenza $(f_L - f_i)$ non sia trascurabile, si può ricavare $(t_f - t_{0L})$, e quindi il ritardo del filtro t_f.

Dall'analisi del grafico riportato in figura 7 e nella successivo figura 8 si nota che che la fase assume valore uguale a zero per il campione con indice pari a 712 e quindi per $t = 0.712$ secondi.

Al tempo $t = 0$ la fase estrapolata risulta quindi pari a

$$-(30.003786 * 0.712)\ mod\ 1.0\ = -0.362696 \tag{13}$$

da cui, sostituendo i valori di t_{0L} e t_0, nella (7) al tempo $t = 0$ si ricava $t_f = 11.25$ms.

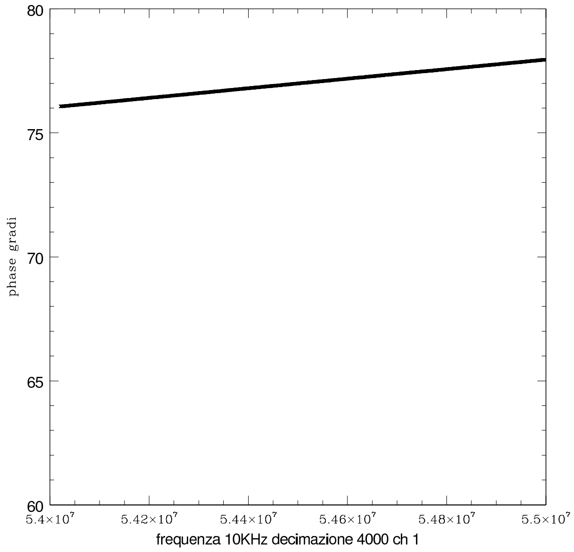

Figura 5: Fase dei campioni in funzione del tempo, frequenza programmata 10 KHz, (in ascissa è riportato il tempo in μsec)

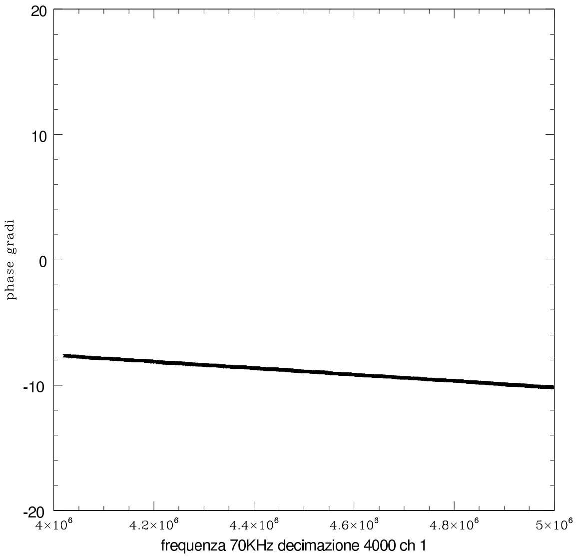

Figura 6: Fase dei campioni in funzione del tempo, frequenza programmata 70 KHz,(in ascissa è riportato il tempo in μsec)

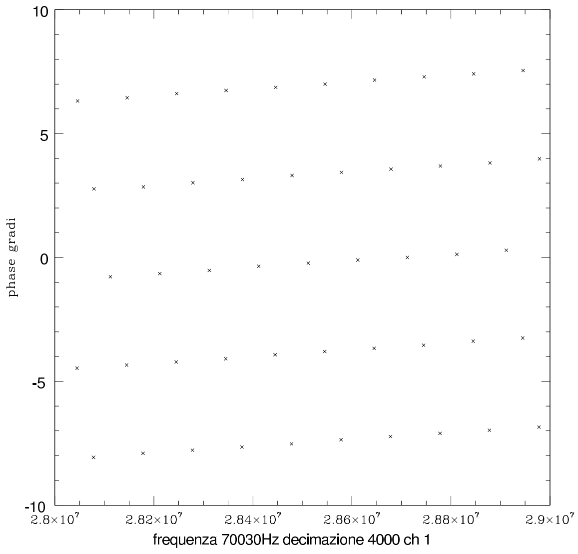

Figura 7: Fase dei campioni in funzione del tempo, frequenza programmata 70.030 KHz (in ascissa è riportato il tempo in μsec)

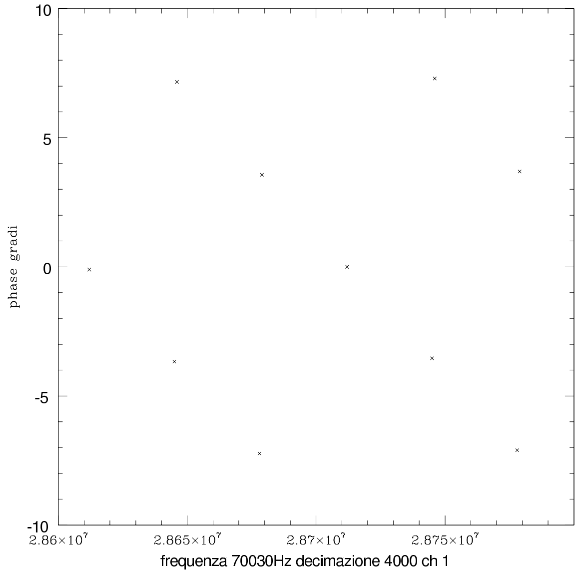

Figura 8: Fase dei campioni in funzione del tempo, frequenza programmata 70.030 KHz (in ascissa è riportato il tempo in μsec)

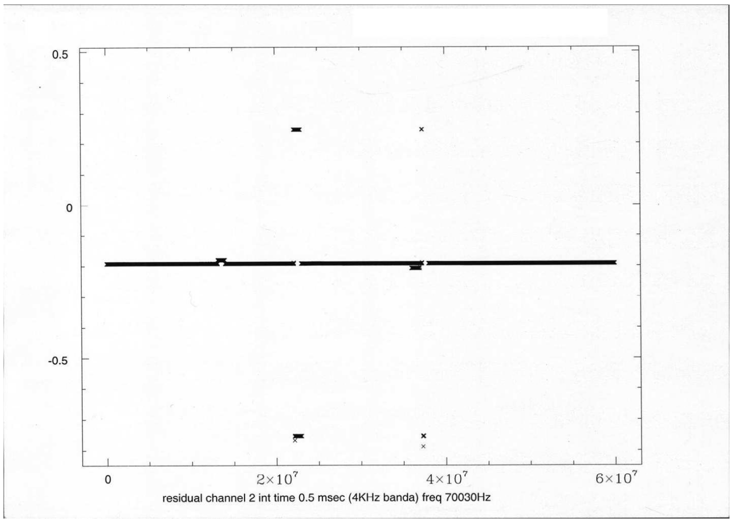

Figura 9: Residui di fase, ovvero differenza tra fase programmata e fase ricevuta in un minuto di acquisizione. (in ordinata è riportata la fase in giri, mentre in ascissa compare il tempo di acquisizione in μsec)

6.4 Phase slip

Confrontando, su un dataset di alcuni minuti, la fase programmata con quella attesa, si sono evidenziati sporadici errori di fase, del tutto impredicibili, della durata di un secondo ciascuno. Un esempio di questi errori si può vedere in figura 9.

L'errore corrisponde ad un errore nella programmazione iniziale della fase dell'oscillatore digitale, che assume un valore differente da zero. L'evento avviene in modo non correlato nei due canali utilizzati del ricevitore.

Il problema rende sostanzialmente inutilizzabile il sistema, in quanto la predicibilità della fase è l'elemento più critico per l'utilizzo del sistema negli esperimenti di radio science.

La durata dell'acquisizione variava da 1 minuto circa a 30 sec, a seconda della decimazione utilizzata sulla macchina SUN, mentre una volta trasferito il *Client* di prova sotto Linux, con un disco quindi di capacità molto superiore si sono effettuate acquisizioni anche di 10 minuti. In tali acquisizioni condotte con il PPC 6030 si manifestavano anche errori di frequenza dovuti all'errata programmazione della 4284, probabilmente imputabili ad una errata lettura del semaforo dal bus VME.

7 Conclusioni

La realizzazione del ricevitore è risultata molto più problematica del previsto, per malfunzionamenti hardware delle schede Pentek. Riassumendo i problemi riscontrati sono stati:

- **Errori di fase**: dall'analisi dei residui di fase tra fase teorica e fase programmata si osserva che la fase iniziale dell'oscillatore digitale non sempre viene resettata correttamente a zero.

  Il problema sembra presumibilmente dovuto ad una non corretta acquisizione del segnale di sincronismo da parte del Graychip per la presenza di spikes di corrente nella scheda.

- **Specifiche del bus VME**: utilizzando la scheda MV 2100 Motorola si sono riscontrati problemi di lettura dal bus VME, che sono risultati dovuti ad una non rispetto delle corrette temporizzazioni del segnale DTACK da parte della scheda 4284 Pentek. La scheda funziona correttamente solo quando l'indirizzo generato dalla CPU viene mantenuto costante, e non variato dopo che la scheda ha rilasciato il segnale DTACK.

Il problema è stato parzialmente risolto modificando la sequenza di programmazione da parte della CPU sulla memoria condivisa.

- **Canali non funzionanti**: la scheda 4271 presentava due integrati di servizio malfunzionanti, il che ha impedito di utilizzare 2 dei 4 canali durante le misure.

I problemi sono stati affrontati con patches hardware e software, che però hanno limitato parzialmente la funzionalità del ricevitore, e hanno richiesto un cospicuo allungamento dei tempi di sviluppo. Purtroppo non è stato possibile trovare una soluzione al problema dei glitches di fase.

Riferimenti bibliografici

[1] G. Comoretto, B. Bertotti, L. Iess and R. Ambrosini: Doppler Experiments with Cassini Radio System, Il NUOVO CIMENTO, Vol.15C, N. 6 Novembre-Dicembre 1992.

[2] B. Bertotti, R. Ambrosini, J.W. Armstrong, S.W. Asmar, G. Comoretto, G. Giampieri, L. Iess, Y. Koyama, A. Messeri, A. Vecchio and H.D. Wahlquist: Search for gravitational wave trains with the spacecraft UlYSSES Astron. Astrophys. 296 13-25 (1995)

[3] G. Comoretto: Doppler receiver for Cassini radio science experiments , Planet. Space Sci., Vol 46, No 9/10 pp.1415-1416, 1998

[4] R. Ambrosini, G. Comoretto, M. Mercolino, P. Persi: Manuale operativo per l'inseguimento Doppler di sonde interplanetarie da Noto, IRA 333/03 (Bologna, 2003).

Indice