INAF - Osservatorio Astrofisico di Arcetri
INAF - I.R.A.

Progetto Fasti

Manuale d'uso del Software - III
Il programma Ftest

C. Baffa, E. Giani

Rapporto Interno di Arcetri N° **4/2005**
Firenze, Settembre 2005

Sommario.

Per lo sviluppo e l'uso dell'elettronica di controllo per rivelatori infrarossi bidimensionali Fasti *sono stati sviluppati diversi strumenti software. Questo rapporto ne descrive l'uso e fornisce le ulteriori informazioni necessarie per il test e l'utilizzo di* Fasti.

Il presente rapporto interno è il terzo di una serie in cui vengono descritti tutti gli strumenti software di Fasti. *Nel seguito descriveremo il programma di controllo generale,* **Ftest**.

*In precedenti rapporti sono stati descritti gli altri componenti della collezione di strumenti software. Il primo rapporto[5] copre sia il linguaggio che il compilatore (***svb1asm***) dello assembler per il sistema di generazione di sequenze per* Fasti, *lo* SVB1, *che l'uso dell'emulatore software dello* SVB1, **EmuSvb**. *Il secondo rapporto[10] descrive il programma di visualizzazione delle forme d'onda,* **GsvbPlot**, *il programma di visualizzazione di un flusso ininterrotto di dati* **FreeRun**, *ed il programma di controllo remoto* **GClient**.

Capitolo 1

Introduzione

Il gruppo infrarosso di Arcetri ha realizzato una elettronica *leggera* per l'acquisizione dati con rivelatori bidimensionali infrarossi, Fasti. La caratteristica di Fasti è l'appoggiarsi più su standard industriali e *disegni concettuali* che su devices specifici, per la facilità di sviluppo e per evitare una prematura obsolescenza. La struttura dettagliata di Fasti si può trovare in diversi documenti[1, 3, 4].

Uno dei componenti cruciali che abbiamo sviluppato è il generatore di sequenze, lo SVB1[7]. Lo SVB1 è un *disegno concettuale* che è stato realizzato tramite la tecnologia dei componenti programmabili. Nella versione per NICS è implementato tramite due chip Xilix XC95288. All'interno vi è della logica schematizzabile come un microprocessore specializzato nella produzione di sequenze. Abbiamo definito uno *pseudoassembler* per la programmazione di tali sequenze, per agevolare il loro sviluppo. Tramite il compilatore **svb1asm** si possono tradurre questi *programmi* di generazione di forme d'onda in un formato direttamente comprensibile dallo SVB1. Il programma **svb1asm** fornisce gli usuali strumenti di un'assembler, come la definizione di simboli e label.

Per facilitare ulteriormente lo sviluppo di forme d'onda adeguate, sono stati sviluppati anche due programmi ausiliari: un programma di emulazione software del generatore di sequenze, **EmuSvb**, ed un programma che permette di esplorare gaficamente i segnali prodotti, **GsvbPlot**.

I segnali generati dal rivelatore vengono convertiti da quattro ADC a 16 bit, posti su due schede. I dati risultanti vengono trasmessi tramite quattro bus seriali veloce ad una scheda *Buffer* che si occupa del colloquio con il calcolatore di controllo e, quando richiesto, della sottrazione tra prima e seconda lettura. I dati vengono infine letti dal PC di controllo[8] tramite una scheda NI PCI-DIO32HS, per cui è stato sviluppato appositamente un device driver per Linux[9].

Il programma che permette il controllo di Fasti, sia in fase di test che in fase di utilizzo regolare è **Ftest**. Questo programma ha due modi principali di funzionamento, il modo interattivo ed il modo *demone*. Nel primo caso permette un controllo assai dettagliato dell'elettronica, consentendo sia

operazioni elementari di test sia acquisizioni, display e acquisizioni continue. Nel secondo caso agisce come un programma demone di Unix, eseguendo i comandi che riceve tramite una socket di rete. Questo è anche il modo usuale di funzionamento al telescopio, rendendo Fasti un *device di rete*, con gli ovvi benefici in termini di versatilità.
Il programma **GClient** permette il test di questo tipo di funzionamento ed un limitato uso di Fasti in questa modalità.

Il presente rapporto descrive l'uso del programma **Ftest** e tutte le informazioni necessarie per la compresione della sua struttura e del modo di funzionamento.

Capitolo 2

Il programma Ftest

Il disegno di Fasti prevede che l'elettronica di piano focale sia controllata da un PC[8] che si occupa dell'inizializzazione e del controllo dei vari sottosistemi e della comunicazione con il calcolatore centrale che gestisce la interazione con l'utente e il colloquio con il telescopio.

Il PC *embedded* è un normale PC commerciale montato in un cassetto da rack. È dotato di un disco a stato solido da cui viene effettuato il boot di un sistema Linux minimale, che, alla fine delle operazioni di boot descritte nel relativo rapporto interno[8], lancia il programma di controllo **Ftest**.

Il programma **Ftest** come già detto in precedenza, è disegnato per operare in due modalità distinte: come demone per il controllo del sistema al telescopio, e in modalità interattiva per lo sviluppo ed il test sia in laboratorio che al telescopio. La selezione del modo operativo avviene tramite una opzione sulla linea di comando. Sempre tramite opzioni sulla linea di comando si può modificare pesantemente il comportamento del programma.

Capitolo 3

Le opzioni e i modi operativi di Ftest

Le opzioni specificabili sulla linea di comando del programma **Ftest** possono essere ottenute con il comando `ftest h`. Nella figura 3.1 diamo il risultato di questo comando per la revisione corrente al momento della scrittura di questo documento.

La prima opzione di cui parliamo è l'opzione `x`. Se viene specificata nella line di comando, **Ftest** commuta al modo demone e si sconnette dal teminale. Da quel momento ulteriori interazioni possono solo avvenire tramite la porta di rete 8081 del PC embedded. Normalmente tale opzione viene usata dalla procedura di boot del PC embedded, oppure nel debug del software. Se **Ftest** deve essere usato interattivamente, occorre prestare attenzione al fatto che **non** deve essere già in esecuzione, magari come demone. Infatti una seconda istanza del programma non riuscirebbe ad allocare le risorse del sistema e potrebbe corrompere i files di log.

Il secondo gruppo di opzioni è relativo al modo di operare nei confronti del rivelatore. Con le opzioni `q1, q2 e q4` si specifica il numero di quadranti attivi del rivelatore. Il programma **Ftest**, in base a questa scelta, seleziona il tipo di comportamento più appriato.

Il modo `q1` è inteso per il funzionamento con i rivelatori L^3CCD, per cui viene forzata la dimensione del rivelatore a 128×128, sono disabilitati i dispositivi `SVB1` ed il dsplay LCD (opzioni `s` e `l`). Questa opzione abilita anche tutte le procedure per il trattamento dei dati relative a questo tipo di rivelatore (come la soppressione delle prime 8 colonne *cieche*).

Il modo `q2` implica l'assenza della scheda di *Buffer* e la lettura diretta di una sola scheda ADC da parte del PC embedded. Questo modo è inteso per poter acquisire i dati senza passare attraverso la scheda *Buffer*, tipicamente in laboratorio, durante il test delle schede ADC, e per la caratterizzazione dei rumori con un rivelatore ingegneristico. I dati vengono quindi duplicati per avere un file in uscita delle stesse dimensioni di quello che si avrebbe leggendo un frame intero (2048×2048).

```
ftest           (called as >ftest<  $Revision: 1.11 $)

Available options:

 h    this help text
 1/2  svb serial selection
 l    LCD off
 s    svb off
 i    I/O board off
 v    really verbose mode
 a    all message logging
 m    matlab out-file format
 n    normal menu
 d    debug menu
 x    Server mode (force all message logging)
 q1   l3ccd detector (force 128*128 detector)
      Also force l (LCD off) and s (svb off)
 q2   One ADC Board
 q4   Full 4 quadrantl3ccd detector
```

Figura 3.1: Opzioni da linea di comando

Il modo `q4` è il modo di operazione standard, con tutta la catena di acquisizione montata (scheda ADC e *Buffer*) ed implica la lettura di tutti e quattro i quadranti. Forza anche Fasti a decodificare i dati ricevuti secondo lo schema di auto-sincronizzazione generato dalla scheda *Buffer*. Tale schema ha lo scopo di risincronizzare *in corsa* un frame cui sia mancato uno o più pixels, ottenendo lo scopo di non perdere l'intera misura.

Il terzo gruppo di opzioni è relativo al modo interattivo. Le due opzioni `n` e `d` specificano quale menù venga mostrato alla partenze nel modo interattivo. L'opzione `n` forza la visualizzazione del menu di acquisizione normale (vedi Fig. 3.2), attraverso cui si possono acquisire un numero specificato di frames interi o lanciare un modo di acquisizione continua (free-run). È anche possibile specificare delle acquisizioni senza immagazzinarle sul disco.

L'opzione `d` forza la visualizzazione del menu di debug (vedi Fig. 3.3). Tale menu dà accesso ad una nutrita serie di opzioni che permettono di verificare il funzionamento delle varie parti del sistema, eseguendo operazioni elementari o permettendo un controllo fine del modo di funzionamento del sistema Fasti. Tali menu saranno descritti in modo più ampio nel seguito.

L'opzione `a` forza la scrittura di tutti i messaggi generati da **Ftest** nel file `ftest.log` nella directory corrente. Questa opzione viene forzata dal modo demone e in tale caso il file di log è salvato nella directory `\tmp`.

L'opzione `v` abilita un maggiore dettaglio nei messaggi di stato (opzione di verbosità).

L'opzione `m` abilita la scrittura dei dati in un formato compatibile con i programmi Matlab ed Octave. Se questa opzione non è presente i dati sono scritti in formato FITS.

L'opzione `h` stampa un testo di aiuto e termina il programma.

Le due opzioni `1` e `2` permettono di selezionare quale delle due porte seriali (COM1, default, o COM2) è connessa allo SVB1.

L'opzione `l` fa partire il programma disabilitando il collegamento con il display LCD. Le funzionalità vengono conseguentemente ridotte.

L'opzione `s` fa partire il programma disabilitando il collegamento con lo SVB1. Le funzionalità vengono conseguentemente ridotte.

L'opzione `i` fa partire il programma disabilitando il collegamento con la scheda di acqusiszione parallela. Le funzionalità vengono conseguentemente ridotte.

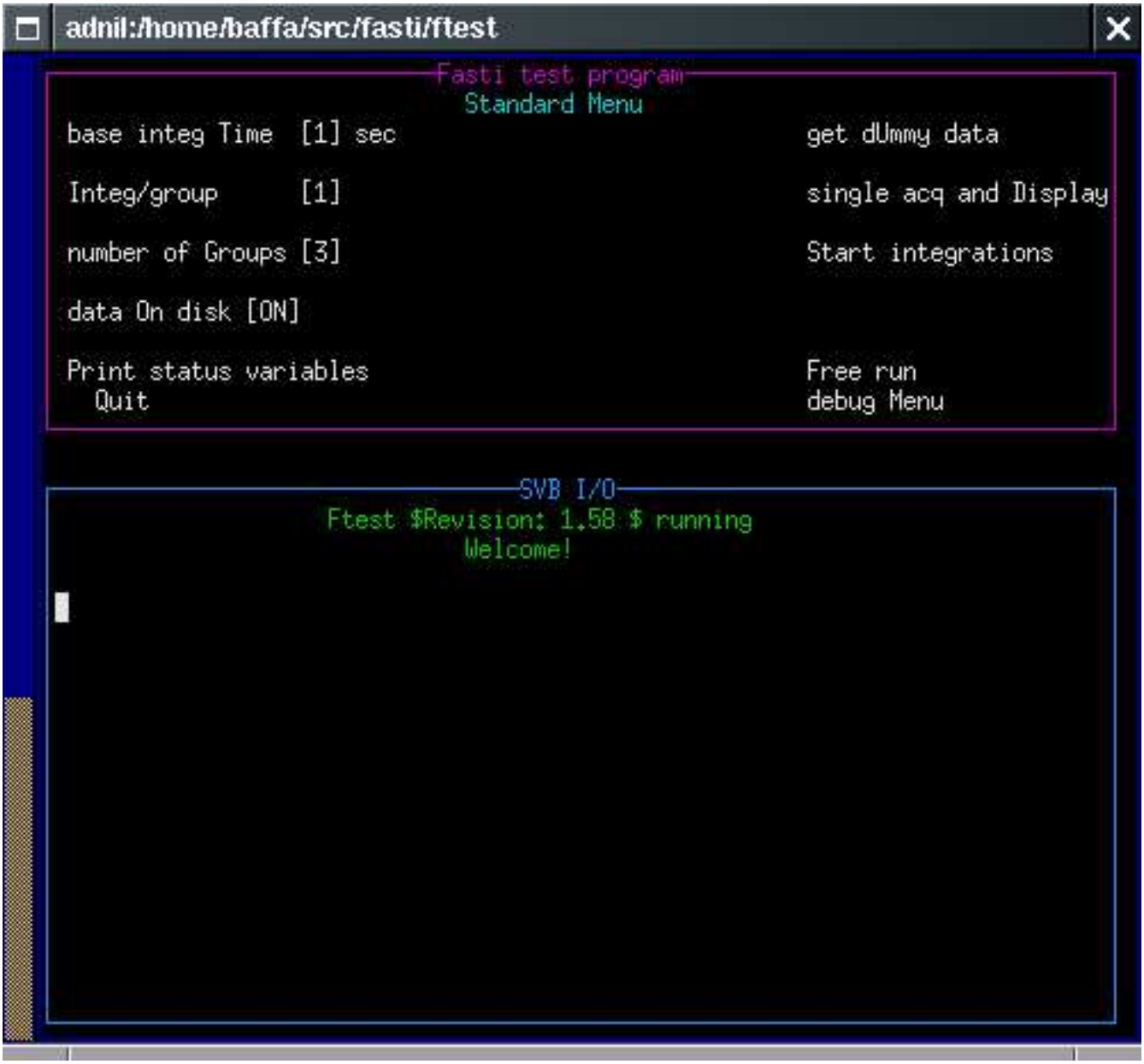

Figura 3.2: Menu di acquisizione normale

Figura 3.3: Menu di debug

Capitolo 4

Il menu di acquisizione

Il menu di acquisizione permette di eseguire diversi tipi di acquisizione di immagini. Questi modi sono organizzati in modo da poter essere utili soprattutto in fase di test e non sono pensati per acquisizioni scientifiche; le modatlità di presa dati sono comunque le stesse.

Nel nome delle varie opzioni può essere inclusa una lettera maiuscola (come la 'O' in `data On disk`), tale lettera è il tasto 'attivo' che permette l'esecuzione della relativa opzione in modo diretto, senza doverla preventivamente selezionare.

Quando il programma si trova in attesa di un comando, il sistema riprogramma lo SVB1 in modo che il rivelatore esegua una serie continua di integrazioni alla massima frequenza possibile, per svuotare le capacità parassite e ridurre il rumore.

4.1 Le voci del menu di acquisizione

Nel menu di acquisizione vi è una voce da utilizzarsi per test preliminari: è `get dUmmy data`. Questa opzione è pensata per il test del software e della catena di acquisizione: il sistema genera un'immagine artificiale e la invia al PC di controllo.

Le opzioni proprie del menu sono:

- `base integ Time`. Permette di specificare il tempo elementare di integrazione sul rivelatore (usualmente indicato come DIT o TINT). Ha una granularità di 0.1 secondi, e viene programmato con un algoritmo formato da tre loop annidati. In appendice vengono dati alcuni dettagli in più. In rete, sul sito di Fasti è possibile consultare un memo che riporta la procedura utilizzata e i grafici degli errori che si ottengono (al massimo lo 0.08%). in ogni caso nello header del file fits viene riportato il tempo *effettivo* di integrazione. Il valore minimo dipende dalla frequenza di scansione ed è tipicamente di 1-2 secondi. Il valore massimo è di 5000s.

- `Integ/group`. Specifica il numero di immagini che vengono sommate prima di scrivere il risultato su disco. Il valore raccomandato è 1.
- `number of Groups`. Specifica il numero di gruppi di immagine che vengono misurati e quindi scritti su disco. Ad ogni gruppo corrisponde un file fits. Data l'efficienza di acquisizione di Fasti, tali misure vengono eseguite in modo continuo, senza interruzione, e il tempo impiegato in totale è appena superiore al tempo di misura più il tempo di scansione dello array.
- `data On disk`. specifica se i dati devono essere effettivamente scritti su disco o no.
- `Print status variables`. Mostra lo stato interno delle variabili dello SVB1. Un esempio commentato è presentato più avanti, nella trattazione del controllo remoto (pagina 20).
- `single acq and Display`. Acquisisce una singola immagine con il tempo di integrazione elementare specificato e la mostra tramite un programma di visualizzazione (DS9 o Saoimage).
- `Start acquisition`. Fa partire le acqusisizioni, ottenendo il numero di gruppi specificato, ciascuno del desiderato numero di integrazioni elementari. Non ci sono pause o interruzioni tra le singole integrazioni.
- `Free run`. Fa partire una serie ininterrotta di immagini e le mostra tramite un programma di display appositamente sviluppato, e descritto in [10].

4.2 La generazione del DIT

Il tempo di integrazione elementare (DIT o TINT) è definito come il tempo che intercorre tra il reset di un pixel e la effettiva lettura. Il DIT è quindi pari al tempo di attesa programmato più il tempo di scansione del pixel. Siccome il reset è assai veloce (circa 1.2 msec) il tempo che intercorre tra il reset del pixel e la prima lettura varia lungo l'array, mentre il tempo di integrazione, compreso tra due scansioni uguali è costante lungo l'array. Il tempo di scansione è dipendente da due parametri definiti nella forma d'onda specifica `shold` (il tempo di assestamento del segnale prima della conversione, e `pixwait` (il tempo di attesa prima della commutazione al pixel successivo). Il tempo per pixel è quindi $(shold + pixwait + 3.)/(8 \cdot 10^6)$ secondi ed il tempo di scansione dello array è

$$\frac{shold + pixwait + 3}{30.51}$$

(nota: $30.51 = (8 \cdot 10^6)/512^2$). I valori correnti di `pixwait` e `shold` si ottengono dal comando status, valori tipici possono essere 4 e 21 (scansione a 275Mhz, tempo totale .91sec).

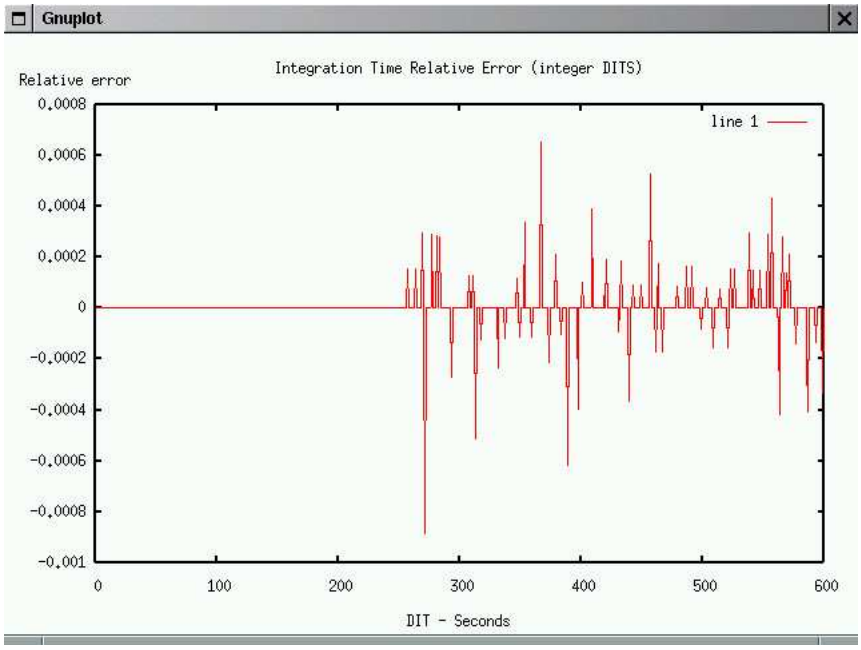

Figura 4.1: Errore relativo sul DIT per tempi di integrazione interi

La gestione del tempo di integrazione è realizzata tramite l'uso di tre loop annidati, ciascuno limitato a 255 iterazioni. La granularità temporale e' pari a 4ms, coincidenti con il minimo intervallo. Il massimo intervallo programmabile è invece 125000s.

La struttura a tre loop annidati ci fornisce facilità di implementazione e lunghi tempi massimi, ma ha un difetto. Il tempo di integrazione da programmare (DIT), espresso in unità di 4ms, deve essere espresso come il prodotto di tre numeri interi inferiori a 256. Questo non è sempre possibile, abbiamo quindi sviluppato una procedura che ricerca il numero più vicino a quello desiderato che possa essere scomposto nel modo voluto. Il valore (approssimato) così ottenuto viene reso al programma Xnics e riportato negli header dei dati, che risultano correttamente etichettati.

Abbiamo eseguito delle simulazioni sulla generazione di tempi di integrazione con un passo di 0.1 secondi e di 1 secondo. Nel primo caso l'errore massimo (0.4% del tempo di integrazione) si ha per 28.1s. Per i DIT interi, il massimo errore è sensibilmente inferiore (0.08%).

Sul sito di fasti[6] sono riportati i grafici delle due simulazioni, con una breve discussione della casistica. Riportiamo in Fig. 4.1 il grafico relativo ai tempi interi fino a 10 minuti. Notiamo come l'inviluppo degli errori sia grosso modo iperbolico e l'errore relativo diminuisce abbastanza regolarmente.

Capitolo 5

Il menu di debug

Il menu di debug permette l'esecuzione di una serie di operazioni il cui fine è la verifica del funzionamento dell'elettronica di acquisizione.

Le opzioni del menu possono essere divise in più parti, rispetto ai sottosistemi su cui operano. Vi sono alcune sovrapposizioni e noi duplicheremo le relative parti per completezza.

Nel nome delle varie opzioni può essere inclusa una lettera maiuscola (come la 'O' in `dump prOgram memory`), tale lettera è il tasto 'attivo' che permette l'esecuzione della relativa opzione in modo diretto, senza doverla preventivamente selezionare.

5.1 Funzioni di test dello SVB1

Uno dei punti di forza di Fasti è il generatore di forme d'onda programmabile SVB1([7]), schematizzabile come un microprocessore specializzato nella produzione di sequenze. Questo *microprocssore virtuale* ha due banchi separati di memoria, uno per il programma ed uno per i dati da generare ed un registro che ne mostra lo stato istantaneo.

I comandi di verifica elementare delle memorie sono:

- `test Program memory`. Verifica il funzionamento della memoria di programma scrivendo un pattern variabile prestabilito e rileggendolo.
- `test Data memory`. Verifica il funzionamento della memoria dati scrivendo un pattern variabile prestabilito e rileggendolo.
- `continous sVb test`. Esegue alternativamente i test sulle due memorie registrando il numero totale di errori, fino all'ordine di fermata dato dall'operatore.
- `dump prOgram memory`. Elenca il contenuto della memoria di programma, per ispezione manuale del contenuto.
- `dump dAta memory`. Elenca il contenuto della memoria dati, per ispezione manuale del contenuto.

- `fill memories with 0`. Scrive su tutte le celle di memoria degli zeri, per avere un pattern iniziale noto.
- `load obj File`. Legge da disco un file con una forma d'onda in formato già compilato (.obj) e lo scrive sullo SVB1.

5.2 Funzioni relative alle forme d'onda

La flessibilità dello SVB1 viene pienamente apprezzata tramite la programmazione delle forme d'onda, tramite uno *pseudo-assembler* che permette grande flessibilità. La sintassi di questo linguaggio è già stata descritta in un apposito rapporto interno[5], qui descriveremo i comandi che permettono l'effettiva applicazione di questi *pseudo-programmi*.

I comandi relativi alla programmazione delle forme d'onda sono:

- `load obj File`. Carica dal disco dello host il file contente la forma d'onda da programmare. Il programma chiede il nome del file da caricare e suggerisce, come default `nics.obj`.
- `fLip svb lines`. Carica ed esegue una forma d'onda particolare che permette di generare un'onda quadra a $0.5Mhz$ su tutte le linee di uscita.

5.3 Funzioni di controllo dello SVB1

Per verificare le funzionalità della scheda SVB1, vi sono numerosi controlli sul suo stato di funzionamento.

- `continous Run`. Abilita la generazione delle forme d'onda in modo ininterrotto.
- `sTop` SVB1. Ferma lo SVB1 al successivo raggiungimento dell'indirizzo zero (tipicamente fine della forma d'onda).
- `Halt` SVB1. Ferma immediatamente lo SVB1.
- `step` SVB1`(2)`. esegue solo un passo dello pseudoprogramma. Il '2' indica solamente il tasto che attiva l'opzione.
- `fLip` SVB1`lines`. Carica ed esegue una forma d'onda particolare che permette di generare un'onda quadra a $0.5Mhz$ su tutte le linee di uscita.
- SVB1`Status` mostra lo stato interno delle variabili dello SVB1. Un esempio commentato è presentato più avanti, nella trattazione del controllo remoto (pagina 20).

- `run on page X` esegue la forma d'onda presente nella pagina `'X'` (compresa tra 0 e 3) della memora di programma dello SVB1 (una descrizione estesa delle pagine dello SVB1è presente in [7] e in [5]).

5.4 Funzioni di controllo delle schede *Buffer* ed ADC

Questi comandi permettono un controllo dello stato di funzionamento della scheda *Buffer*.

- `multi aquisZ+compare`. Abilita la generazione dei pattern di autotest della scheda buffer, legge i dati relativi e ne verifica la correttezza.
- `adC test`. Abilita la lettura continua di uno streaming di dati da parte della scheda ADC e lancia il programma di visualizzazione ([10]).
- `Word to read`. Specifica il numero di campioni acquisiti nel test precedente.

5.5 Funzioni di acquisizione dati

Questi comandi sono relativi all'acquisizione di dati in modo di test. Non sono pensati per una acquisizione *normale*, per la quale occorre far riferimento al menu standard.

- `dummY acquisition`. Il programma genera un'immagine artificiale. Questa opzione è pensata per il test del software e della catena di acquisizione.
- `single acqUisition`. Viene fatto partire il programma di generazione delle forme d'onda e viene acquisito un frame.
- `[no] data on disK`. Abilita o disabilita la scrittura dei dati acquisiti su disco. Il formato è FITS o matlab, a seconda della opzione su riga di comando. Il default è FITS.
- `sinGle read`. Disabilita la sottrazione tra le due letture (fine integrazione - inizio integrazione) e abilita la scrittura di *entrambe* su disco.
- `douBle read`. [default] Abilita la sottrazione tra le due letture (fine integrazione - inizio integrazione) e abilita la scrittura della *differenza* su disco.
- `acquisition tIme` . Permette di specificare il tempo elementare di integrazione sul rivelatore (TINT o DIT).

Capitolo 6

Ftest come demone

Un demone è un processo che viene eseguito in *background* in attesa che si verifichi qualche evento o in attesa di eseguire un certo compito specifico su base periodica. Ad esso non è associato un terminale di controllo o una login di shell.

Un demone può essere avviato in diversi modi, **Ftest** quando eseguito come demone, viene lanciato durante l'avviamento del sistema. In fase di debug pu essere avviato da un terminale di utente come job in foreground o background.

6.1 Il server demone Ftest

Ftest è stato modificato in modo che venga eseguito come processo *server* demone sul sistema *embedded* di Fasti.

Ftest elabora le richieste provenienti dall' applicazione Xnics, il programma utente di acquisizione, che viene eseguito su una macchina remota con una connessione di rete diretta al sistema embedded.

6.2 Trasformazione di Ftest in demone di sistema

Il programma **Ftest** viene eseguito come demone di sistema chiamando la funzione **daemon()**. Questa funzione esegue i primi 4 passi descritti in A.2.

daemon() accetta due argomenti *nochdir* e *noclose* che se posti uguali a zero cambiano rispettivamente la *directory* corrente di lavoro nella *root directory* e redirigono lo *standard input,output* ed *error* su */dev/null*.

Come riferimento diamo una implementazione standard della funzione *daemon()*:

```
#include <fcntl.h>
#include <paths.h>
#include <unistd.h>
```

```
int daemon(int nochdir, int noclose) {
        int fd;

        switch (fork()){
                case -1: return -1;
                case 0: break;
                default: _exit(0);
        }
        if (setsid() == -1) return -1;
        if (!nochdir) chdir("/");
        if (noclose) return 0;
        fd = open(_PATH_DEVNULL, O_RDWR, 0);
        if (fd != -1) {
                dup2(fd, STDIN_FILENO);
                dup2(fd, STDOUT_FILENO);
                dup2(fd, STDERR_FILENO);
                if (fd > 2) close(fd);
        }
 return 0;
}
```

Il processo demone **Ftest** viene successivamente configurato in modo da ignorare i segnali di terminale SIGTINT, SIGQUIT, SIGHUP, SIGTTIN e SIGTTOU.

In corrispondenza del segnale SIGTERM viene invece registrata la routine *termination_handler()* che garantisce una conclusione corretta del demone **Ftest** .

6.3 Funzioni server di Ftest

Il processo **Ftest** gestisce la richiesta proveniente dal solo cliente Xnics, quindi lo possiamo classificare come un *server iterativo*.

La struttura generale di un *server iterativo* orientato alla connessione, è la seguente:

1. creazione di una socket
   (a) sock =socket(AF_INET,SOCK_STREAM,0);
2. *bind* ad un indirizzo noto
   (a) - bind(sock,localaddr,addrlen);
   (b) - port: può essere un indirizzo noto oppure 0. In questo caso il sistema assegna automaticamente un numero di porta, nell'intervallo da 1024 a 5000.

(c) - address: generalmente viene usato INADDR_ANY. Con tale costante il sistema accetta una connessione su qualsiasi interfaccia Internet;

3. configurazione della socket in modalitá passiva

   (a) - listen(sock,queuelen)

   (b) - queuelen: la lunghezza massima dipende dall'implementazione

4. accetta una connessione dal cliente

   (a) - new_sock =accept(sock,addr,addrlen)

   (b) - la chiamata accept() blocca finchè non c'è almeno una richiesta di connessione [1]

5. interazione con il cliente

   (a) - read(new_socket,....)

   (b) - write(new_socket,...)

6. chiusura della socket e ritorno alla chiamata *accept()*(punto 4)

Nel caso specifico, **Ftest** apre due *socket*, una per i comandi ed una per i dati, in corrispondenza della porta 8081 e si pone in attesa delle connessioni da parte dell'applicazione cliente Xnics.

Una volta stabilite le connessioni, queste vengono usate dai due processi per lo scambio di comandi e dei dati, per cui ci riferiamo ad esse con il termine di *socket* di comando e *socket* dei dati.

Le *socket* rimangono aperte per tutta la durata del programma cliente Xnics. Quando questo termina, le connessioni vengono chiuse da entrambe le parti.

6.4 Interazione con il cliente

Nel programma **Ftest** l'interazione con il cliente viene gestita dalla funzione *MainLoopServer()*.

La routine *MainLoopServer()* legge dalla *socket* di comando le richieste inviate dall'applicazione Xnics ed esegue le corrispondenti operazioni.

I comandi e gli eventuali parametri spediti attraverso la connessione di comando, devono essere specificati seguendo il protocollo di comunicazione stabilito per le due applicazioni: questo richiede che ciascun stringa di comando sia correttamente terminata con il carattere nullo *'\0'*.

All'interno della routine *MainLoopServer()* la funzione *parse_cmd()* si occupa di individuare i comandi all'interno della stringa letta dalla *socket*.

[1] Questo è vero solo se *sock* è in modalitá bloccante.

6.5 Trasferimento dei dati acquisiti

I dati vengono spediti all' applicazione Xnics sulla *socket* dati. I dati sono immagazzinati in una matrice di 1024 righe per 1024 colonne e viene spedita una riga per volta (2048 bytes). Dopo la spedizione il programma attende la conferma della corretta ricezione dei dati.

Nel caso in cui si verifichi un errore la riga viene rispedita nuovamente per intero.

6.6 Terminazione di Ftest

In corrispondenza del segnale SIGTERM è stata registrata la routine di gestione *termination_handler()* che stampa un messaggio sul *file* di log ed azzera la *flag keep_gooing* che controlla l'iterazione del server.

Durante la fase di *test* del programma demone, ci siamo accorti che il processo demone **Ftest** non termina propriamente quando il *server* esegue la chiamata *accept()* e la socket di controllo[2] è in modalità bloccante. La *man page* di *accept()* riporta che quando la chiamata è interrotta da un segnale restituisce *-1* con *errno* posto uguale ad EINTR.

Nel nostro caso ciò non accade perchè *Linux* usa i segnali in stile BSD, cioè alcune chiamate di sistema vengono automaticamente riavviate. Ne segue che il demone può essere terminato solo dal segnale SIGKILL.

Per evitare questa situazione possiamo agire nei seguenti modi:

1. configurare la *socket* di controllo come non bloccante. In questo caso la chiamata *accept()* non blocca, e restituisce -1 con *errno* uguale ad EGAIN se non ci sono richieste di connessione.

2. usare la chiamata *select()* prima di effettuare *accept()*. In caso di interruzione da parte di un segnale, la chiamata *select()* restituisce -1 con *errno* uguale ad EINTR, e quindi siamo in grado di gestire l'interruzione della chiamata da parte del segnale SIGTERM.

3. registrare il gestore del segnale SIGTERM con la funzione *sigaction()* (invece di *signal()*) non utilizzando la *flag* SA_RESTART. L'assenza di questa opzione implica che la chiamata di sistema non viene riavviata automaticamente.

Noi abbiamo usato il secondo metodo, ma abbiamo verificato come anche gli altri due funzionino correttamente.

[2] quella sulla quale viene effettuata la chiamata accept()

Capitolo 7

Il protocollo di comunicazione con Ftest

Agendo da demone, **Ftest** viene controllato dal programma di interfaccia utente tramite due socket **permanenti** sulla porta 8081. La prima socket è il canale di controllo, la seconda permette lo scambio dei dati. Il protocollo che corre sulla porta di controllo e' molto semplice: sono stringhe ascii terminate dal carattere ';' e dal carattere *'\0'*. I comandi che Xnics invia sono:

STATUS	stato delle variabili interne
TINT 23	tempo di integrazione in decimi di secondo
DIT 23	tempo di integrazione
GROUP 1	numero di gruppi di integrazioni da eseguire
ITER 3	numero di integrazioni elementari da eseguire
RUN	partenza integrazioni
DATI	invia un frame dummy, per test
DOUBLE [0-1]	seleziona double/sigle sampling
DUMMYDATA	invia un frame dummy, per test
END	fine del programma cliente - xnics
STOP	kill del programma server - ftest
REINIT	reinit delle periferiche
ABORT	fermo delle integrazioni in corso
WAVE	programma il file nics.obj
LOAD ¡nomefile¿ ¡size¿	riceve un file wave di lunghezza size e lo usa

Descriviamo brevemente il comportamneto dei principali comandi.

7.1 Il comando STATUS

Il comando STATUS richiede ad **Ftest** l'insieme di tutte le informazioni rilevanti al funzionamento del sistema di acquisizione, riporta infatti sia lo stato dell'elettronica che del programma.

Il formato delle informazioni che il comando STATUS invia è:

```
Msg: printstatus1: Sernum=2       (serial line selector)
Msg: printstatus2: Serial=1       (serial enabled (0 = off))
Msg: printstatus3: Verbose=1      (Flag to print verbose output)
Msg: printstatus4: Svbstatus=0    (Flag of svb status (0 = ok))
Msg: printstatus5: Iostatus=0     (Flag of i/o status (0 = ok))
Msg: printstatus6: Lcdstatus=0    (Flag of LCD status (0 = ok))
Msg: printstatus7: Bufstatus=0    (Flag of Buffer status (0 = ok))
Msg: printstatus8: DoubleOn=0     (Flag double/single reads setting)
Msg: printstatus9: Logging=1      (Flag of log enabled(0 = ok))
Msg: printstatus10: Matlab=0      (Flag of matlab/octave file format(0=off))
Msg: printstatus11: Data_on_disk=0 (Flag enabling write of data on disk)
Msg: printstatus12: Quadrants=4   (number of quadrants to handle)
Msg: printstatus13: file_n=60     (sequential number of save file)
Msg: printstatus14: tint=10       (integration time in tenth of a second)
Msg: printstatus15: acquired=0    (number of frames acquired)
Msg: printstatus16: lastfile=     (last data file saved)
Msg: printstatus17: N_ROW=1024    (number of detector rows)
Msg: printstatus18: N_COL=1024    (number of detector columns)
Msg: printstatus19: Menu=0        (which menu)
Msg: printstatus20: shold=4       (sample and hold time on svb)
Msg: printstatus21: pixwait=21    (wait time on pixel)
Msg:  SVB status stopped - Program Counter    0 - Program sign C723
Msg:  Loop registers - A =    0 B =    0 C =    0 D =    0
```

Le voci sono spiegate brevemente direttamente nello output. Riportiamo alcune precisazioni. Il valore del Menu è sempre 0 (indica il funzionamento in modo demone). Le voci `lastfile`, `logging`, `Matlab` sono destinate a rimenere inattive in quanto relative al modo di funzionamento interattivo. Allo stesso modo `Data_on_disk` vale sempre zero. Il valore di `Quadrants` deve essere 4 durante il funzionamento ordinario.

Le variabili `Serial`, `Svbstatus`, `Iostatus`, `Bufstatus` e `Lcdstatus` indicano lo stato di funzionamento dei vari sottosistemi e **devono** essere tutte attive durante il funzionamento normale.

7.2 I comandi TINT, DIT, GROUP e ITER

I comandi TINT, DIT, GROUP e ITER specificano le operazioni di integrazione da eseguire.

TINT e DIT sono sinonimi ed esprimono il tempo di integrazione in decimi di secondo.

7.3 il comando RUN

Un run invece dá, sul canale messaggi il seguente traffico (3 integrazioni):

```
Msg: Run on page 0
Msg:  bytes read 2097152
Msg: Mean-> =   5030.695    3289.826    4558.797    2834.126
Msg: Std->  =    571.626    535.0897    579.1168    542.4465
Msg:  acquisition speed 0.874196 MB/sec
```

```
Msg:  bytes read 2097152
Msg: Mean-> =   5338.289    3598.655    4865.139    3142.095
Msg: Std->  =   481.9913     432.406    491.8841    441.9599
Msg:  acquisition speed 1.35798 MB/sec
Msg:  bytes read 2097152
Msg: Mean-> =   5279.834    3543.501     4806.82    3086.934
Msg: Std->  =   474.6114    424.1807     484.644    433.9807
Msg:  acquisition speed 1.33482 MB/sec
Msg: We stop SVB at once.
```

Mentre un REINIT da':

```
Msg:  daemon mode started
Msg: initdev: I/O board initialization
Msg:  Done!
Msg: initdev: Lcd Display init
Msg: welcomelcd: 1
Msg:  2
Msg:  3
Msg:  4
Msg: initdev: Done!
Msg: initdev: Serial line init (
Msg: /dev/ttyS1
Msg: )
Msg: initdev: SVB reset
Msg: (resetsvb 3) Clean of input buffer
Msg: (resetsvb 8) T/R pair found at iteration 0
Msg: (resetsvb 14) read char # 1
Msg: (resetsvb 8) T/R pair found at iteration 0
Msg: (resetsvb 14) read char # 1
Msg:  2
Msg:  3
Msg: (resetsvb 15) Sycronization succeded
Msg: initdev: Next file will be ./data/data0060
Msg: initdev: Two quadrants active
Msg: Handling file nics.obj
Msg:  File programmed.
Msg: Main: We use daemon mode
Msg: run_daemon_socket: connected socket 8
Msg: MainLoopServer: starting
```

Il dettaglio dei messaggi puó cambiare in qualche caratteristica minore, ad esempio la modifica della spaziatura o le correzioni di errori tipografici.

Appendice A

Processi demoni

Diamo qui uno schema di riferimento ai processi *demoni*.

Un demone è un processo che viene eseguito in *background* in attesa che si verifichi qualche evento o in attesa di eseguire un certo compito specifico su base periodica. Ad esso non è associato un terminale di controllo o una login di shell.

A.1 Avviamento di un processo demone

Un demone può essere avviato in diversi modi:

1. durante l'avviamento del sistema. La maggior parte dei demoni di sistema sono avviati dallo script di inizializzazione /etc/rc che viene eseguito da /etc/init quando il sitema è predisposto per la multiutenza. In questo caso il demone avrá i privilegi del superuser.
2. Dal file di sistema crontab su base periodica
3. Dal file di utente crontab su base periodica
4. eseguendo il comando at che pianifica un job per l'esecuzione in qualche istante successivo
5. da un terminale di utente come job in foreground o background. Di solito questo avviene durante la fase di test del demone.

A parte quest'ultimo caso, gli altri esempi generano un demone che non è connesso con un terminale. In effetti sono l'interazione normale di un teminale di login, il gruppo di processo associato a quel terminale e tutte le relative interazioni dei segnali che generano i problemi che devono essere gestiti correttamente quando si scrive un demone.

A.2 Codifica di un demone

La caratteristica dei processi demoni è che il loro tempo di vita è l'intero tempo in cui il sistema è in funzione. Generalmente vengono avviati al *bootstrap* del sistema e vengono terminati solo quando viene eseguito lo *shutdown*.

Esiste un insieme di regole per la codifica di un demone. Queste regole sono volte a fornire un demone "robusto" che possa essere invocato in uno qualsiasi dei modi elencati precedentemente.

1. La prima operazione consiste nell'eseguire la chiamata di sistema *fork* e fare in modo che il processo genitore esca. In questo modo se il demone viene eseguito come semplice comando di shell, la sua terminazione conduce la shell a ritenere che il comando abbia finito. Inoltre il processo figlio eredita il numero identificativo del processo di gruppo (PGID) del padre ma ottiene un diverso numero di processo (PID). In questo modo viene garantito che il processo figlio non è il *leader* del gruppo del processo. Questo costituisce un prerequisito per la chiamata a *setsid*

2. Viene chiamata *setsid()* per creare una nuova sessione. Il processo generato diventa *leader* di sessione di una nuova sessione e *leader* del gruppo di un nuovo gruppo di processo. Inoltre ad esso non è associato alcun terminale.

3. Cambiare la *directory* corrente di lavoro nella *directory* di *root*. La *directory* corrente di lavoro viene ereditata dal processo padre e pu'o trovarsi su un *filesystem* montato. Dato che il demone generalmente non termina fino a che il sistema non esegue il *reboot*, se il demone si trova su una partizione montata, il *filesystem* non pu'o essere smontato.

4. azzerare la maschera di creazione dei *files* (*umask(0)*). La maschera di creazione dei *files* viene ereditata dal processo genitore e questa potrebbe negare alcuni permessi che il processo demone potrebbe invece volere abilitare nei *files* creati.

5. I decrittori dei files non utilizzati devono essere chiusi. Ciò vale specialmente per lo *standard input*, lo *standard output* e lo *standard error* che il processo può avere ereditato dal processo genitore.

A.3 Terminazione del demone

Il segnale SIGTERM viene usato dal sistema per notificare a tutti i processi in esecuzione che il sistema sta passando dalla multiutenza alla monoutenza. Il segnale viene inviato dal processo *init*, che aspetta la terminazione dei

processi. Se un processo non è terminato dopo un certo lasso di tempo, il segnale SIGKILL viene inviato al processo che non può ignorarlo. Un demone dovrebbe catturare il segnale SIGTERM ed utilizzarlo per arrestare gradualmente le sue operazioni.

Bibliografia

[1] Baffa, C., Fasti un controller veloce per l'Infrarosso, 1998, Memo del Gruppo Infrarosso di Arcetri.

[2] Baffa, C., C., Biliotti, V., Progetto Fasti - Un assembler per lo SVB1, Rapporto interno dell'Osservatorio Astrofisico di Arcetri, **N°1/2000.**

[3] Baffa, C., The Fasti Project, Memorie della Società Astronomica Italiana, 2003, **74**, p165.

[4] Baffa, C., Biliotti, V., Checcucci, A., Gavrioussev, V., Gennari, S., Giani, E., Lisi, F., Marcucci, G., Sozzi., M., "The Fasti Project", ADASS XII ASP Conference Series, Vol. **295**, 2003, Payne, H., Jedrzejewski, R., Hook, R. Eds., p.355

[5] Baffa, C., Giani, E., Progetto Fasti - L'assembler e l'emulatore, Rapporto interno dell'Osservatorio Astrofisico di Arcetri, **N°1/2004.**

[6] Baffa, C., Giani, E., Sito Web ddi Fasti, 2001-2005, http://www.arcetri.astro.it/irlab/fasti/

[7] Biliotti, V., Il generatore di sequenze SVB1 per Fasti, 2000, Rapporto Interno dell'Osservatorio di Arcetri, **N°2/2000**

[8] Checcucci, A., Baffa, C., Giani, E., "Progetto Fasti – Fasti Global Controller, Rapporto interno dell'Osservatorio Astrofisico di Arcetri, **N°3/2001**.

[9] Giani, E., Checcucci, A., Baffa, C., "Progetto Fasti – Driver Linux per la scheda di acquisizione NI PCI-DIO32HS, Versione 2, Rapporto interno dell'Osservatorio Astrofisico di Arcetri, **N°3/2002**.

[10] Giani, E., Baffa, C., Progetto Fasti - Il programma di plot e le interfacce grafiche, Rapporto interno dell'Osservatorio Astrofisico di Arcetri, **N°2/2004.**

Indice