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

Progetto Giano

Il programma `server104` ed il sistema embedded

C. Baffa, V. Biliotti, E. Giani

Rapporto Interno di Arcetri N° **2-2010**
Firenze, 10/2010

Sommario

Questo documento descrive il sistema di controllo dell'elettronica dello spettrometro infrarosso GIANO.

Il controller è realizzato con un sistema embedded basato su hardware PC/104, uno standard largamento diffuso in campo industriale. Su tale macchina è installato il sistema operativo GNU/Linux, basato sulla distribuzione Slax 6.

Il controller è visto dalla macchina che esegue l'applicazione utente come un device di rete e cui si accede tramite un socket standard del dominio Internet.

Capitolo 1

Hardware e software del sistema

Il sistema embedded di GIANO ha la caratteristica di fornire bus e interfacce di tipo standard, requisito che riteniamo fondamentale nella realizzazione di un sistema embedded.

Lo hardware del *controller* è costituito da una scheda PC104 PFM-620S della AAEON con CPU Celeron 400/650MHz a basso consumo, controller video, controller IDE, interfaccia di rete, porte seriale RS232, parallela e USB 1.1 e il supporto per moduli di memoria Type I CompactFlash. Tale sistema ha tutte le funzionalità di un classico PC, ma ha un fattore di forma più compatto.

Il sistema embedded di GIANO non prevede l'uso di dischi con parti mobili, ma una cartuccia di memoria tipo CompactFlash sulla quale è stato installato il sistema operativo e il software.

1.1 Lo standard PC104

Il PC104 è uno standard per sistemi PC compatibili, molto compatti, pensati per la realizzazione di sistemi embedded, che prevede la possibilità di avere schede con diverse funzioni *impilati* uno su l'altra per creare un sistema più complesso.

Macchine PC104 sono spesso usate nell'industria e nei laboratori per fornire sistemi di controllo programmabile per sistemi complessi. Le schede PC104 sono progettate con un basso consumo e per essere modulari ed espandibili.

La popolarità del formato meccanico del PC104 è dovuta alla sua compattezza, all'implementazione di un'interfaccia standard aperta e matura come quello ISA e alla vasta gamma di moduli PC104 disponibili.

Il bus PC104 è un bus derivato dalle specifiche dei bus PC/AT (16-bits)

e PC/XT (8-bits) dell'IBM, è caratterizzato da bassi consumi e modificato meccanicamente per la massima compattezza.

Questi sistemi sono normalmente basati su di CPU con architettura compatibile X86 e possono essere programmati con gli stessi tools di sviluppo usati per un PC classico.

1.2 L'elettronica di acquisizione

Il rivelatore di GIANO, è un HAWAI-II, 2048 x 2048 pixels della Teledyne Technologies. Esso è costituito da quattro quadranti indipendenti sia per quanto riguarda i segnali di controllo, sia per i segnali di uscita.

L'elettronica di piano focale, o scheda fredda, ospita 4 circuiti elettronici indipendenti, che eseguono la sottrazione tra il segnale di uscita ed un valore di riferimento (*BIAS*), ed amplificano il risultato. Il risultato dell'amplificazione viene poi inviato alle schede analogiche con un circuito differenziale. Ognuno di questi circuiti è controllato da una scheda analogica che lavora a temperatura ambiente e che ospita:

- la circuiteria per la generazione dei clock, delle tensioni di bias e delle alimentazioni
- il convertitore analogico digitale a 18 bit
- un adattatore seriale che, attraverso una fibra ottica si collega alla scheda di interfaccia (`scheda buffer`).

I segnali in uscita dal quadrante, una volta convertiti, sono inviati dalla scheda analogica alla scheda di interfaccia, attraverso il link ottico.

La `scheda buffer` immagazzina i pixels inviati dalle schede analogiche in una memoria organizzata in 4 banchi di memoria, ciascuno associato ad un quadrante, in grado di accumulare fino a quattro immagine di quadro. La scheda di interfaccia è connessa al controller per mezzo del bus ISA a 16 bit attraverso il quale sono scambiati dati e comandi.

Il controllo dell'elettronica avviene tramite alcuni comandi che vengono scambiati attraverso l'interfaccia PC104 e sono documentati in [1].

Alcuni dei comandi riguardano l'interfaccia stessa, come ad esempio quelli relativi alla lettura del quadrante o al test di memoria. Altri comandi o dati sono relativi alla programmazione di una specifica scheda analogica, come ad esempio la configurazione dei bias di riferimento, la selezione dei filtri, i dati per la programmazione del *sequencer*. I comandi possono essere indirizzati ad un singolo quadrante, oppure a tutti in contemporanea (comando broadcast).

I pixels sono trasmessi alla scheda di interfaccia appena disponibili all'uscita dell'ADC e sono immagazzinati in una memoria di buffer in modo che il PC104 li possa leggere alla velocità massima, secondo una logica FIFO.

Poichè ogni quadrante del rivelatore è connesso con un circuito ad esso dedicato, vi è una grande flessibilità sulle condizioni di lavoro del rivelatore infrarosso. Per esempio ciascun quadrante potrebbe essere programmato con tempi di integrazioni e/o di scansione diversi, oppure il numero di letture per misura potrebbe essere differente, consentendo tecniche di misura sofisticate.

L'elettronica fornisce anche alcuni segnali ne caso di malfunzionamenti interni. Se la conversione non produce risultati, l'elettronica inserisce automaticamente un valore 3. Se invece vi è un errore di trasmissione sulla fibra ottica, il valore inserito è 15. Nelle misure di laboratorio, in assenza di guasti, il tasso di occorrenza di questi eventi è risultato inferiore a 10^{-12}.

1.3 Il sistema operativo

La scelta di sistemi operativi per il mondo embedded è molto ampia, ma la tendenza è verso l'affermazione di sistemi *Open Source* tra cui Linux. L'alternativa consiste nello scegliere un sistema proprietario o commerciale che puó offrire il vantaggio di migliori prestazione per il real time, ed una migliore armonizzazione delle applicazioni e dell'hardware. Spesso peró tale vantaggio è più che compensato da una maggiore possibilità di innovazione oppure dalle competenze tecniche già acquisite.

Grazie al libero accesso dei sorgenti, alla sua robustezza, affidabilità, flessibilità, personalizzazione e configurazione (tanto che puó essere reso piccolo), la disponibilità di molti protocolli di comunicazione, Linux è entrato a far parte del mondo embedded.

In ambito GNU/Linux vi sono molte distribuzioni specificatamente adattate per lavorare con sistemi embedded. È possibile inoltre crearne una personalizzata, come abbiamo giá fatto in altra occasione.

Abbiamo comunque ritenuto più semplice, soprattutto per motivi di manutenzione, appoggiarsi ad un sistema già sviluppato e provato La nostra scelta è ricaduta sulla distribuzione GNU/Linux **Slax**[4], di piccole dimensioni e basata su Slackware, distribuzione di struttura semplice e razionale ed a noi già familiare. Inizialmente nata come distribuzione Linux Live CD, **Slax** inizialmente era pensata per non essere installata sul disco rigido, ma partire da CD o da Pendrive USB. Avendo riscosso notevole successo, **Slax** è stata successivamente addattata per essere installata su disco e funzionare come una qualcunque altra distribuzione Linux.

L'opzione di lavorare completamente in RAM, lasciando cosí libero l'accesso al disco, rende **Slax** particolarmente adatta ai sistemi embedded.

Un'altra caratteristica di **Slax**, che l'ha resa ancora più interessante ai nostri occhi, è la possibilità di essere estesa tramite moduli aggiuntivi, permettendo la disponibilità di ulteriori software oltre a quelli forniti.

Capitolo 2

Il programma `server104`

Il programma che permette il controllo di GIANO, sia in fase di test sia in fase di utilizzo regolare è `server104`. Rimandiamo ad un successivo rapporto la descrizione dettagliata della sua struttura dal punto di vista della programmazione.

`server104` è disegnato per operare in due modalità distinte: come demone per il controllo del sistema al telescopio, e in modo interattivo per lo sviluppo ed il test sia in laboratorio sia al telescopio.

Nel modo interattivo, `server104` è controllato da un piccolo gruppo di menù pensati per l'uso da terminale ANSI.

Nel modo di utilizzo non interattivo, il programma funziona come un demone Unix, eseguendo i comandi che riceve tramite un *socket* di rete. Questo è anche il funzionamento standard al telescopio.

In entrambi i casi l'applicazione consente un controllo dettagliato dell'elettronica, sia attraverso operazioni elementari di test, sia acquisizioni, display e acquisizioni continue.

L'applicazione `server104` comprende una parte di basso livello per le operazioni di I/O verso la `scheda buffer`, e una parte di più alto livello per la comunicazione con l'utente o con l'applicazione *middle-ware* `gbridge` a seconda che lavori in modo *menù* o *demone*.

L'applicazione `server104` presenta una grande flessibilità di lavoro per adattarsi ad un'elettronica altamente configurabile ed altrettanto flessibile.

2.1 Esecuzione del `server104`

La selezione del modo operativo avviene tramite un'opzione a linea di comando e, in modo analogo, si può modificare il comportamento del programma.

Le opzioni accettate dal programma `server104` possono essere visualizzate lanciando comando `server104 -h`, il cui output è il seguente:

```
-h          Output this help
-v <level>  verbosity level (Default 1)
-x          daemon mode
-m          menu mode
-b          board number (1-4, 5 == all)
-l          LCD not present
-t          test mode
```

Se specificata l'opzione `x`, il `server104` commuta al modo demone e si sconnette dal teminale. Da quel momento ulteriori interazioni possono solo avvenire tramite le porte di rete 8082 e 8083 del PC embedded: la prima è usata per la ricezione e conferma dei comandi ricevuti dal programma `gbridge`, la seconda per il trasferimento dei *frames*.

L'opzione opposta `m` permette l'esecuzione del `server104` in modo interattivo, con un *menu* di acquisizione in stile `ncurses`, da cui l'utente puó selezionare varie voci.

L'opzione bX (`b1, b2, b3, b4 o b5`) permette di selezionare quante e quali schede analogiche sono attive per l'acquisizione[1].

Il modo `b5` è il modo di operazione standard ed implica la lettura di tutti e quattro i quadranti. In base a questa opzione, il programma seleziona il tipo di comportamento più appriato.

Prendiamo infine in esame le ultime opzioni:

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

L'opzione `l` fa partire il programma disabilitando il collegamento con il display LCD.

L'opzione `t` fa partire il programma in modalità *offline* o modo *test*, cioè senza l'elettronica. Ovviamente le funzionalità in questo caso sono estremamente ridotte, ma in questo modo, specialmente quando il `server104` lavora in modo demone, è possibile testare la comunicazione di rete con l'applicazione *middle-ware*.

2.2 Interfacciamento del PC104 alla scheda buffer

La maggior parte delle schede ISA hanno la memoria configurata per locazioni fisiche comprese nel range tra 640 e 1024 KB.

La `scheda buffer` è stata progettata per rispondere agli indirizzi compresi nel range 0xD0000 - 0xDFFFF, intervallo che nel PC embedded non risulta riservato ad altro hardware[2].

[1] Vogliamo sottolineare la distinzione che esiste tra specificare la scheda e il quadrante. L'opzione consente di selezionare la scheda analogica e non il quadrante perchè la corrispondenza tra queste e i quadranti è tale che la scheda analogica 1 risulta associata al quadrante 3, mentre la 3 al quadrante 1. Per gli altri due quadranti, invece, abbiamo una corrispondenza esatta

[2] L'intervallo di indirizzi liberi è visualizzabile eseguendo un cat del file */proc/iomem*.

L'area di memoria è suddivisa in due parti: la zona 0xD0000-0xD7FFF è riservata all'accesso dei registri e delle funzionalità della `scheda buffer`, mentre la zona 0xD8000-0xDFFFF è riservata all'accesso delle funzionalità delle schede analogiche calde. In particolare per ogni scheda, i primi 2Kbytes sono destinati alla memoria di sequenza, i restanti ai registri specifici della scheda analogica.

Rimandiamo al documento [2] per maggiori dettagli sugli indirizzi dei registri e le loro funzioni.

2.2.1 Programmazione e lettura dello stato della `scheda buffer` e delle schede analogiche

Il `server104` accede in lettura e scrittura alla scheda di interfaccia *custom* attraverso la mappatura degli indirizzi della scheda nel proprio spazio di indirizzamento.

Il sistema di protezione di Linux consente ad un processo con privilegi di esecuzione opportuni, l'accesso via *mmap* al device */dev/mem*. Questo specifico *device* rappresenta l'immagine della memoria principale del PC. Gli indirizzi dei bytes in */dev/mem* sono quindi interpretati come indirizzi della memoria fisica. L'operazione di *mmap* restituisce un indirizzo nello spazio di indirizzamento del processo, attraverso il quale il processo accede in modo diretto ai registri e alla memoria della scheda di interfaccia.

Questo approccio ha l'indiscutibile vantaggio di evitare la scrittura di un device driver specifico per la scheda buffer e di lavorare completamente in *user space*. D'altra parte presenta un punto debole dal punto di vista della sicurezza: il processo `server104` deve infatti essere eseguito con i permessi di *superuser*, indispensabili per accedere al device di memoria.

Nel nostro caso questo non rappresenta un problema, perché l'applicazione è in esecuzione su un sistema embedded dedicato, il cui accesso è circoscritto e limitato.

Come appena descritto, l'accesso alla scheda di interfaccia e alle 4 schede analogiche, avviene attraverso dei registri mappati nell'aria del processo [3].

Alcuni di questi registri sono di sola lettura, e forniscono informazioni come il numero identificativo della `scheda buffer` lo stato delle alimentazione e del *sequencer*, lo stato del link ottico insieme all'eventuale *error rate*.

I registri in scrittura, invece, permettono la configurazione dell'elettronica (bias, offset, periodo di scansione del pixel etc.), oppure di eseguire precisi comandi come l'avvio del *sequencer*, operazioni di autotest dei sottositemi dell'elettronica, accensione o spengimento del sensore etc.

Lo sviluppo del codice ha seguito un approccio *bottom-up*, sono prima state scritte le funzioni elementari per la scrittura e lettura della memo-

[3] Gli *offset* di questi registri rispetto all'indirizzo base, sono elencati nel file include `define.h` e documentati in [1].

ria della `scheda buffer` e successivamente è stata creata una raccolta di funzioni via via più complesse associate alle funzionalità dell'elettronica.

La presenza di registri *broadcast* consente di eseguire un comando in contemporanea per tutte le schede, per evitare dissincronizzazioni dovute alla necessità di eseguire la stessa operazione quattro volte, una per ogni scheda.

2.3 La memoria di sequenza

Per generare i segnali di clock per la scansione dell'array, il *sequencer* esegue la serie di macro-istruzioni (*il programma*) caricata nell'area di memoria opportunamente predisposta.

Ciascuna scheda analogica calda ha una propria area di memoria di dimensione 2KB, all'interno della quale viene eseguito l'*upload* del programma. Tale quantità di memoria è ben sufficiente, tenuto conto del livello delle macro-istruzioni che possono essere definite[1].

La programmazione del *sequencer* può avvenire in tre distinti modi:

- on fly: i valori specificati dall'utente sono processati dall'applicazione e la programmazione del *sequencer* viene fatta al volo, traducendo in linee di comando del programma i valori introdotti
- attraverso un file di sequenza, costituito da un elenco di comandi direttamente interpretabili dal *sequencer*
- attraverso un file che specifica una sequenza di operazioni complesse. Questa funzionalità è prevista per implementare o sequenze arbitrarie di multicampionamenti o misure molto lunghe alternate a calibrazioni.

Il file che definisce la sequenza di operazioni complesse è scritto in una sorta di meta linguaggio. Le operazioni specificabili sono:

- **reset**, comprendente anche operazioni di reset-read
- **letture**, pura lettura
- **eventi di calibrazione**, normalmente con lettura dell'array
- **restart**, riesecuzione del programma

Ogni riga specifica due valori: il primo è un carattere identificativo del comando, il secondo è un parametro aggiuntivo. Per un'operazione di reset (carattere **R**, il parametro specificato rappresenta il numero di reset da effettuare, per un'operazione di lettura (carattere **L** o **l**) invece, rappresenta il tempo di integrazione espresso in secondi. Altre lettere (come **AaBb** ecc.) specificano eventi di calibrazione, come l'inserimento nel percorso ottico di lampade o celle di calibrazione.

La procedura di scrittura del *sequencer* prevede una serie di operazioni di sicurezza, tra cui il controllo della dimensione del programma da scrivere che non deve eccedere i 2 KBytes e la verifica della coerenza del contenuto della memoria con quello del programma caricato.

2.3.1 Specifiche relative alla programmazione della memoria di sequenza

La memoria di sequenza è ampia 1024 *words* e in ogni locazione viene memorizzata un'istruzione di programma (di 16 bit). I primi due bit dell'istruzione di programma indicano il tipo di operazione, mentre i restanti 14 bit, interpretati come due gruppi di 7 bit ciascuno, assumono significati diversi a seconda del tipo di operazione. Per una discussione dettagliata del calcolo dei tempi si veda [3].

- `SHORT RESET`
  - bit `15-14`: **00**
  - bit `7-13` sono tutti 0
  - bit `0-6` non significativi
- `RESET-READ`:
  - bit `15-14`: **00**
  - bit `7-13` forniscono la durata del semiperiodo di clock secondo la formula $(2^7 + 16) * 61ns$. La commutazione del pixel si ha sempre sul fronte di salita del clock di pixel per cui il tempo di lettura del pixel esattamente il doppio di questo valore. Il valore minimo si ha quando tutti i bit sono a zero e corrisponde esattamente a 2μs.
  - bit `0-6`: rappresentano il ritardo di campionamento rispetto al clock, in particolare il ritardo rispetto al fronte di discesa del clock di pixel,calcolato con la formula precedente. Se questo valore risulta maggiore di quello ottenuto con i 7 bit più significativi, otteniamo una lettura senza conversione. Il ritardo di campionamento rispetto alla commutazione del pixel dato da questo piu il semiperiodo di clock.
- `READ`:
  - bit `15-14`: **01**
  - bit `7-13`: forniscono la durata del semiperiodo di clock secondo la formula $(2^7 + 16) * 61ns$

  - bit `0-6`: rappresentano il ritardo di campionamento rispetto al clock, calcolato usando la formula precedente. Se questo valore risulta maggiore di quello ottenuto con i 7 bit più significativi, otteniamo una lettura senza conversione.

- `INTEGRAZIONE` :
  - bit `15-14`: **10**
  - bit `0-13`: forniscono il tempo di integrazione, il cui valore è calcolato come $(2^{14} - 1) * 10ms$. Il tempo minimo, ottenuto quando tutti i 14 bit sono a zero, vale *10 ms*.

- `RESTART` :
  - bit `15-14`: **11**
  - bit `0-13`: non sono significativi

Capitolo 3

`server104` in modo menu

Il `server104` in modalità *menu* permette di eseguire diversi tipi di test dell'elettronica e di acquisizione di immagini. Questi comandi sono organizzati da risultare utili soprattutto in fase di test e non sono pensati per acquisizioni scientifiche; le modalità di presa dati sono comunque compatibili con le acquisizioni scientifiche.

Il programma presenta due *menu* distinti, uno più tecnico, detto di *debug*, l'altro di più alto livello detto di *acquisizione*. È possibile passare da uno dei due menù all'altro tramite il tasto `/`.

Per entrambi, nel nome delle varie opzioni è inclusa una lettera maiuscola (come la 'S' in `system Status`): tale lettera è il tasto 'attivo' che permette l'esecuzione della relativa opzione in modo diretto, senza doverla preventivamente selezionare con il cursore tramite i tasti freccia.

3.1 Il menu di debug

Il *menu* di debug è il menu che compare all'avvio del programma e permette una serie di operazioni di verifiche di basso livello dell'elettronica. Alcune di queste operazioni possono essere avviate quando è attiva una singola scheda analogica, altre anche con tutte e 4 le schede selezionate.

Possiamo suddividere le opzioni del menu in più sezioni, secondo i sottosistemi su cui operano.

Funzioni di analisi dello stato

Alcune operazioni restituisco le informazione e lo stato relativo alle schede analogiche.

- `svb Id continuos test:` esegue la lettura continua del numero identificativo della `scheda buffer`, per il numero di volte specificato
- `0 generic read:` esegue la lettura da 8 indirizzi consecutivi a partire dall'indirizzo di memoria specificato dall'operatore

- `9 generic write:` esegue una scrittura a un indirizzo generico di memoria
- `opTical link status:` Resituisce lo stato attivo (A) oppure disattivo (-) del link ottico relativo alla scheda analogica selezionata.
- `chAnnel to operate:` consente di selezionare la scheda analogica su cui operare. 5 indica che sono attive tutte le quattro schede.
- `read voltage Log:` legge il log delle alimentazioni per la scheda selezionata. Lavora anche con tutte le schede attive.

Funzioni di test del *sequencer*

Per verificare le funzionalità del *sequencer* di ogni scheda analogica, vi sono alcuni test che operano su singole funzionalità.

- `test Program memory`. Verifica il funzionamento della memoria di programma scrivendo un pattern variabile prestabilito e rileggendolo.
- `dump sequencer memOry:` elenca il contenuto della memoria di programma, per ispezione manuale del contenuto. Questo test è eseguito su una sola scheda per volta.
- `Zero sequencer memory:` scrive su tutte le celle di memoria degli zeri, per avere un pattern iniziale noto.
- `eXercise memory location:` scrive iun numero specificato di volte un pattern prestabilito all'indirizzo della memoria di sequenza inserita e rilegge controllando la conformità del risultato.
- `test sequencer Memory:` esegue per il numero di volte inserito dall'operatore il test della memoria di sequenza della scheda analogica selezionata. Questo test puó essere eseguito su tutte e quattro le schede in contamporanea.

Funzioni di programmazione del *sequencer*

- `Program sequencer:` carica dal disco dello host il file con le istruzioni di un programma da scrivere nella memoria del *sequencer*Viene chiesto il nome del file da caricare e come default si ha `sequenza.txt`.
- `sYntetize program:` programma la memoria di sequenza al volo, costruendo le istruzioni di progamma a partire dai parametri dell'acquisizione selezionati dall'utente.

Funzioni di controllo del *sequencer*

Per verificare le funzioni del *sequencer* di ogni scheda analogica, e la correttezza dei programmi scritti, vi sono numerosi controlli sul suo stato di funzionamento.

- `3 start sequencer:` fa partire il *sequencer*
- `4 stop sequencer:` ferma il *sequencer* all'istruzione RESTART
- `5 abort sequencer:` ferma immediatamente il *sequencer*.

Configurazione dei parametri dell'elettronica

Questi comandi consentono la configurazione dei paremtri dell'elettronica analogica e di acquisizione.

- `6 sensor on/off [....]:` commuta (accende e spenge) lo stato dell'alimentazione del canale attivo. Lo stato di accensione di tutti e quattro canali viene visualizzato.
- `8 Offset 1-4 set:` modifica la configurazione dei valori delle tensioni di offset per il canale attivo.
- `Filter(s) [....]:` seleziona il filtro che opera sul canale attivo. Mostra il filtro attivo per tutti e quattro i quadranti.
- `quicK level set:` consente la configurazione dei valori delle tensioni di reset e bias del canale attivo tramite cursore in modalità semigrafica.
- `loW level prog data:` permette una configurazione dettagliata dei parametri della scansione. Possono essere specificati il numero di short reset, il semiperiodo del clock di scansione del pixel, il ritardo di conversione e il tempo di integrazione sul rivelatore.

Funzioni di controllo della `scheda buffer`

Questi comandi permettono un controllo dello stato di funzionamento della `scheda buffer`.

- `test acqUisition:` abilita la generazione dei pattern di autotest della scheda buffer, legge i dati relativi dalla FIFO selezionata e ne verifica la correttezza.
- `Read test image:` abilita la generazione di più frame di autotest della scheda buffer. Legge i dati dalla FIFO relativa alla scheda analogica 1 e ne verifica la correttezza.

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.

- `Noise acquisition:` esegue un'acquisizione con i quadranti specificati senza accendere l'alimentazione dell'elettronica fredda. Questa misura permette una valutazione del rumore intrinseco dell'elettronica.
- `Read test image:` avvia il programma di generazione delle forme d'onda e viene acquisito un frame.
- `rEad image:` esegue la programmazione del *sequencer* con il programma selezionato, ed avvia la presa dati dal quadrante specificato. Questo comando funziona su un solo quadrante per volta.

3.1.1 Il menu di acquisizione

L'opzione `/` del *menu* debug, sposta il controllo al *menu* di acquisizione, e, di default, rende attivi tutti e quattro i quadranti del rivelatore.

Le opzioni proprie del menu sono:

- `base integ Time:` permette di specificare il tempo elementare di integrazione sul rivelatore. Ha una granularità di 0.01 secondi. Il valore minimo dipende dalla frequenza di scansione, ed è tipicamente di 8 secondi mentre il valore minimo programmabile è di 2.1 secondi. Il valore massimo accettato dal menu è di 5000 secondi.
- `Dit integ time:` consente di specificare il tempo totale di integrazione della misura, dato dalla somma del tempo di integrazione sul rivelatore (`tint`) e del tempo di scansione del rivelatore.
- `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.
- `doubLe read:` configura il tipo di campionamento, singolo o doppio.
- `mUltifile:` consente la configurazione del file con le meta istruzioni usato nella programmazione *sequencer* per le operazioni di multicampionamento.
- `staRt integrations:` fa partire le acquisisizioni, ottenendo il numero di gruppi specificato, ciascuno del desiderato numero di integrazioni elementari. Non ci sono pause o interruzioni tra le singole integrazioni.
- `multisamPle start:` fa partire il multicampionamento in base ai parametri programmati.

- `randoM action test:` questo test è stato studiato per essere eseguito in modalità batch. All'avvio viene richiesto per quanto tempo il test deve essere eseguito. Una volta avviato, il programma esegue in modo *random* le diverse funzioni di test e acquisizione, modificando sempre in modo casuale, i parametri del sistema. L'esecuzione non si interrompe nel caso di segnalazione di errori. Questi vengono registrati e analizzati successivamente.
  Il test termina allo scadere del tempo di esecuzione specificato dall'operatore, oppure su un'esplicita richiesta di interruzione (ABORT) da parte dell'utente.

- `Base scan time:` permette di programmare il tempo di scansione del pixel. Il range di valori è compreso tra un minimo di 2μs ed un massimo di 17.5 μs. Per avere una maggiore granularità nella selezione del *Base scan*, i valori sono espressi in decimi di μs.

- `pixel semiperiod clocK:` specifica il valore usato per programmare il tempo di scansione del pixel.

- `piXel delay:` specifica il valore per programmare il ritardo per la conversione del pixel. Questa opzione serve per l'ottimizzazione delle forme d'onda in laboratorio.

- `Fsync/lsync delays:` specifica i valori per programmare i ritardi dei segnali di FSYNC e LSYNC. Tale opzione serve per l'ottimizzazione delle forme d'onda in laboratorio.

Capitolo 4

Il `server104` in modalità 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: in particolare l'applicazione `server104` viene avviata al *boot* del sistema dallo script di inizializzazione/etc/rc.d/rc.local. In fase di debug il `server104` puó essere avviato da un terminale di *root* come job in foreground o background.

Il `server104` eseguito in modalità demone elabora le richieste provenienti dall' applicazione `gbridge`, il programma di medio livello eseguito sulla macchina remota di controllo.

4.1 Interfaccia al software `gbridge`

Il sistema embedded viene visto, e a sua volta vede il sistema di controllo come un classico device di rete. La connessione avviene su rete Ethernet a 100Mbit e la comunicazione è completamente gestita dal sistema operativo attraverso lo stack TCP/IP. A livello applicativo il programma usa l'interfaccia di *socket* per gestire i protocolli della suite TCP/IP.

Il demone `server104` viene controllato tramite due socket **permanenti**: uno è il canale di controllo (porta `8082`), l'altro quello dei dati (porta `8083`).

Il protocollo della porta di controllo è descritto nel documento [1] insieme all'elenco dei comandi accettati dalle due applicazioni.

4.1.1 Event driven I/O

Il programma `server104` è strutturato come un'applicazione *event-driven*, cioé un'applicazione in cui il verificarsi di una serie di eventi guida l'evolversi

del sistema.

In questo tipo di programmi gran parte delle funzionalità del sistema sono gestite all'interno di un ciclo infinito detto *ciclo degli eventi*. Durante la sua esecuzione il sistema rimane in attesa degli eventi di I/O e una volta rilevati li processa attraverso funzioni di *callback* registrate in corrispondenza dell'evento stesso.

Nel caso specifico, tra gli eventi di I/O sono classificabili le operazioni di accesso in lettura e scrittura sui *socket*.

4.1.2 Le funzioni di `callback`

Le funzioni di `callback` attivate dagli eventi per operazioni normali di I/O sui *socket* sono distinte a secondo che si tratti di un *socket* di `tipo comandi` oppure `dati` in quanto il protocollo di comunicazione differisce nei due casi.

Si ricorre alla medesima funzione solo quando abbiamo eventi riconducibili ad errori non recuperabili sui due *socket*. In questo caso l'azione registrata corrisponde alla chiusura del *socket*, all'uscita dal *loop* principale e al successivo riavvio della procedura di *listening* sulle porte di ascolto.

Operazioni di I/O sul *socket* di comando

Le informazioni sul *socket* di `comando` viaggiano, in lettura e scrittura, sotto forma di pacchetti di dimensione nota, come descritto in modo approfondito nel documento [1] e a cui rimandiamo per maggiori informazioni.

È sufficiente dire che tale pacchetto prevede un' intestazione di dimensione fissa di 16 bytes, organizzata in 8 campi, contenente:

- un codice caratteristico che garantisce la validità ed integrità del pacchetto
- l'applicazione di destinazione che puó essere `gbridge` oppure **Guilab**
- la descrizione del tipo di pacchetto, come ad esempio `comando`, `info`, `messaggio`, `errore`, `ackwnoledge`
- informazioni relative al tipo del pacchetto. Ad esempio per un pacchetto di tipo `comando` troviamo il codice del comando, per uno di tipo `errore`, il codice di errore
- un'eventuale *payload* la cui dimensione in *bytes* è specificata in un altro campo dell'intestazione

Nel caso di lettura del pacchetto, questa avviene in tre fasi: prima è accertata la validità del pacchetto attraverso l'individuazione della *magick mask*, poi viene letta l'intestazione per accedere al tipo di pacchetto e alla dimensione dell'eventuale `Area dati` che, se presente, viene letta e interpretata secondo le specifiche del protocollo.

Ogni pacchetto valido di tipo `comando` deve essere confermato dall'applicazione ricevente per mezzo di un pacchetto di *acknowledgment*.

Nel caso di scrittura, il pacchetto viene costruito in modo analogo: prima viene scritta l'intestazione con le idicazioni necessarie poi, se prevista dal protocollo, l'`Area dati`.

Una volta assemblato il pacchetto viene trascritto nel buffer di uscita del *socket* `comando` e inviato all'applicazione di destinazione.

Operazioni di I/O sul *socket* dati

Il `socket dati` è un canale su cui vengono eseguite esclusivamente operazioni di scrittura. Ogni operazione di scrittura sul canale `dati` corrisponde all'invio di una riga completa dell'immagine, secondo il protocollo di trasferimento delle immagini descritto al § 5.2.

Ad esclusione del rilevamento della condizione di `End of File`[1], l'applicazione `server104` non effettua operazioni di lettura su questo canale.

4.2 Il protocollo di comunicazione con il `server104`

Descriviamo brevemente il comportamento dei principali comandi che ricalcano alcune delle funzioni vistte nel modo interattivo.

4.2.1 Comandi di stato

STATUS

Il comando richiede al `server104` l'insieme di tutte le informazioni rilevanti al funzionamento del sistema di acquisizione, riporta infatti sia lo stato dell'elettronica, sia quello relativo ai parametri interni del programma.

L'output del comando STATUS differisce a seconda che sia attiva un solo quadrante (o scheda analogica), oppure tutti e quattro.

Di seguito riportiamo il formato delle informazioni che il comando STATUS invia con tutti e quattro i quadranti operanti.

```
status 1: Verbose        =1 (verbosity level <9)
status 2: Menu           =0 (Menu/Daemon flag)
status 3: Logfile        =1 (1= Logfile Active)
status 4: Sendmsg        =1 (Minimum msg lev sent)
status 5: VBuff_id       =EBB3 (buffer board id)
status 6: Acq_type       =0 (single/double read)
status 7: Extra_sub      =0 (Extra pixels subtraction)
status 8: Channel        =5 (channel to operate)
status 9: Acquired       =21126 (images acquired)
status10: Ncol           =1024 (column number)
status11: Nrow           =1024 (row number)
status12: Ngroup         =6 (integrations/group)
status13: Tint           =2 (physical integration time in cs)
```

[1] Questa condizione si verifica alla chiusura del *socket* da parte dell'applicazione *cliente*.

```
status14: Dit            =993 (total integration time in cs)
status15: Base_scan      =93 (pixel scan time)
status16: Scan_time      =9.91 (array scan time in sec)
status17: R_endframe     =2 (end of frame delay)
status18: R_fsync_lsync  =2 (fsync delay)
status19: D_lsync        =2 (lsync duration)
status20: R_lsync_vclk   =2 (lsync delay)
status40: C1.Id          =7 (Board Id)
status41: C1.Status      =0 (Board status)
status42: C1.Filtro      =1 (ADC filter used)
status43: C1.Sensore     =1 (Sensor status)
status44: C1.Link        =1 (optical link status)
status45: C1.V_reset     =23300 (Value of V_reset)
status46: C1.V_bias      =34000 (Value of V_bias)
status47: C1.Offset12    =62920 (offset 1 e 2)
status48: C1.Offset34    =62920 (offset 3 e 4)
status49: C1.VccOn       =1 (alimentation status)
status50: C1.seqfile     =syntetic (sequencer filename)
status51: C1.Prog_dryres =0 (dry reset performed)
status52: C1.Prog_resnum =1 (reset performed)
status53: C1.Prog_readclk=58 (duration of pixel)
status54: C1.Prog_readdel=57 (delay of conversion)
status55: C1.Prog_dit    =2 (integration time)
status56: C1.Prog_fsync  =0 (fsync delays)
status57: C1.Prog_lsync  =0 (lsync delays)
status58: C1.Prog_resclk  =1000 (reset times)
status59: C1.Time_cycle  =10.0022 (total cycle time in seconds)
status70: C2.Id          =1 (Board Id)
status71: C2.Status      =0 (Board status)
status72: C2.Filtro      =1 (ADC filter used)
status73: C2.Sensore     =1 (Sensor status)
status74: C2.Link        =1 (optical link status)
status75: C2.V_reset     =30000 (Value of V_reset)
status76: C2.V_bias      =65000 (Value of V_bias)
status77: C2.Offset12    =62600 (offset 1 e 2)
status78: C2.Offset34    =62600 (offset 3 e 4)
status79: C2.VccOn       =1 (alimentation status)
status80: C2.seqfile     =syntetic (sequencer filename)
status81: C2.Prog_dryres =0 (dry reset performed)
status82: C2.Prog_resnum =1 (reset performed)
status83: C2.Prog_readclk=58 (duration of pixel)
status84: C2.Prog_readdel=57 (delay of conversion)
status85: C2.Prog_dit    =2 (integration time)
status86: C2.Prog_fsync  =0 (fsync delays)
status87: C2.Prog_lsync  =0 (lsync delays)
status88: C2.Prog_resclk  =1000 (reset times)
status89: C2.Time_cycle  =10.0022 (total cycle time in seconds)
status100: C3.Id          =2 (Board Id)
status101: C3.Status      =0 (Board status)
status102: C3.Filtro      =1 (ADC filter used)
status103: C3.Sensore     =1 (Sensor status)
status104: C3.Link        =1 (optical link status)
status105: C3.V_reset     =24000 (Value of V_reset)
status106: C3.V_bias      =34500 (Value of V_bias)
status107: C3.Offset12    =63000 (offset 1 e 2)
status108: C3.Offset34    =63000 (offset 3 e 4)
status109: C3.VccOn       =1 (alimentation status)
status110: C3.seqfile     =syntetic (sequencer filename)
status111: C3.Prog_dryres =0 (dry reset performed)
status112: C3.Prog_resnum =1 (reset performed)
status113: C3.Prog_readclk=58 (duration of pixel)
status114: C3.Prog_readdel=57 (delay of conversion)
```

```
status115: C3.Prog_dit    =2 (integration time)
status116: C3.Prog_fsync  =0 (fsync delays)
status117: C3.Prog_lsync  =0 (lsync delays)
status118: C3.Prog_resclk  =1000 (reset times)
status119: C3.Time_cycle  =10.0022 (total cycle time in seconds)
status130: C4.Id          =10 (Board Id)
status131: C4.Status      =0 (Board status)
status132: C4.Filtro      =1 (ADC filter used)
status133: C4.Sensore     =1 (Sensor status)
status134: C4.Link        =1 (optical link status)
status135: C4.V_reset     =23000 (Value of V_reset)
status136: C4.V_bias      =34000 (Value of V_bias)
status137: C4.Offset12    =63100 (offset 1 e 2)
status138: C4.Offset34    =63100 (offset 3 e 4)
status139: C4.VccOn       =1 (alimentation status)
status140: C4.seqfile     =syntetic (sequencer filename)
status141: C4.Prog_dryres =0 (dry reset performed)
status142: C4.Prog_resnum =1 (reset performed)
status143: C4.Prog_readclk=58 (duration of pixel)
status144: C4.Prog_readdel=57 (delay of conversion)
status145: C4.Prog_dit    =2 (integration time)
status146: C4.Prog_fsync  =0 (fsync delays)
status147: C4.Prog_lsync  =0 (lsync delays)
status148: C4.Prog_resclk  =1000 (reset times)
status149: C4.Time_cycle  =10.0022 (total cycle time in seconds)
```

READLOG

Esegue la lettura del log delle alimentazioni. Produce come output una serie di stringhe che riportano lo stato delle alimentazioni dei quadranti, evidenziando le eventuali anomalie e in tal caso di intervenire sulla parte specifica che ha generato il malfunzionamento.

4.2.2 Comandi di acquisizione

DIT, GROUP, DOUBLE e ITER

I comandi DIT, GROUP e ITER specificano le i parametri per le operazioni di integrazione.

DIT è il tempo di integrazione complessivo, espresso in secondi, dato dalla somma del tempo di integrazione sul rivelatore e il tempo di scansione. Quest'ultimo viene calcolato usando i parametri relativi al dettaglio delle forme d'onda (`resnum, readclk, fsync, lsync e resclk`).

GROUP rappresenta il numero di frames consecutivi da acquisire con lo stesso tempo di integrazione specificato.

Il parametro specificato dal comando ITER non è usato al momento dal programma, ed è posto di default ad 1.

Il comando DOUBLE configura la modalità di acquisizionbe in singolo o doppio campionamento.

MULTI

Questo comando avvia la procedura di multicampionamento.

DUMMYACQ

Questo comando procede alla generazione e invio sul `socket dati` di un frame sintetico. Questa funzionalità risulta utile essenzialmente per analizzare il comportamento del codice relativo al trasferimento dei dati.

RUN

Il comando RUN avvia il processo di presa dati.

FREERUN

Il comando FREERUN avvia l'acquisizione in modalità free-run.

REINIT

Questo comando esegue una re-inizializzazione dell'elettronica di controllo.

4.2.3 Comandi del *sequencer*

DUMPMEM

Il comando esegue il dump del contenuto della memoria di sequenza relativa alla scheda analogica attiva e non viene eseguito se sono abilitate tutti e quattro i canali.

FILLMEM0

Il comando azzera il contenuto della memoria di sequenza della scheda analogica attiva

LOADWAVE

Il comando avvia la programmazione della memoria di sequenza delle schede analogiche attive. Per maggiori dettagli, rimandiamo al § 2.3.

ABORT

Il comando ABORT consente all'utente di interrompere immediatamente la scansione del *sequencer*. Ció comporta l'interruzione brusca dell'operazione in corso e il non completamento di un'eventuale scansione del rivelatore.

STOP

Il comando STOP permette di fermare il *sequencer* al termine del programma, prima del comando RESTART. In questo modo il *sequencer* viene lasciato in uno stato noto così come accade per il rivelatore.

4.2.4 Comandi configurazione sistema

READPARM/WRITEPARM

Questi comandi consentono di accedere in lettura, il primo, e in scrittura il secondo, ai parametri caratteristici dell'elettronica.

I dati da leggere o configurare sono organizzati in una lista di coppie indirizzo-valore: il primo numero specifica il registro o un numero identificativo del parametro, il secondo il valore associato. L'elenco degli indirizzi può essere trovato in [2].

QUADRANTS

Il comando configura i quadranti attivi con cui il sistema lavora. Il sistema lavora o con un solo quadrante attivo, oppure con tutti e quattro in modalità *full frame*.

MSGLEVEL

Questo comando configura il livello dei messaggi che `server104` invia al programma cliente. Messaggi con severità minore del valore configurato, non verranno inviati al programma `gbridge`.

4.2.5 Comandi di test

NOISE

Questo comando abilita le misure di rumore con il sensore spento.

SYNCHRO

Il comando avvia l'acquisizione di frames completi dell'intestazione generata dall'elettronica (header di frame). Questa è costituita da 4 word a 16-bit che specificano oltre una parola di inizio riga (0xFFFF), il numero del frame, il numero di riga e una parola di fine frame (0x0000).

L'analisi del frame completo, ci consente di verificare eventuali errori di dissincronizzazione nella generazione o trasmissione dell'immagine.

SVBTEST

Il comando abilita la generazione interna da parte della scheda buffer di un'immagine sintetica. L'immagine di test ha un pattern ben preciso: i pixels di ogni riga assumono valori da 1 per il primo pixel, fino a 2048 per l'ultimo pixel.

L'analisi del frame ricevuto consente di individuare eventuali errori di lettura o trasmissione del frame.

SVBCHECK

Il comando abilita a livello di scheda `scheda buffer`, il controllo dell'immagine di test ricevuta.

Se viene rilevato un errore, la scheda di interfaccia sostituisce il pixel errato con il valore di default 0x0F. L'attivazione del comando SVBCHECK comporta, necessariamente, l'attivazione del comando precedente per la generazione dell'immagine sintetica.

SEQMEM

Permette di eseguire test di funzionalità della memoria di sequenza di ciascun canale (o scheda analogica).

Il test scrive un pattern noto in corrispondenza dei vari indirizzi di memoria e ne esegue successivamente la lettura, confrontando i valori scritti con quelli letti.

FIFOTST

Questo comando consente di verificare il funzionamento delle FIFO sulla scheda buffer, come pure le varie condizioni di errore.

L'elettronica viene programmata per avviare la funzione di autotest delle fifo. Questa operazione genera un pattern ricorsivo di dati che viene scritto in una, oppure in tutte e quattro le FIFO contemporaneamente. La lettura, invece, puó essere eseguita da una sola FIFO per volta. Il confronto fra i dati letti e quelli attesi, ci permette di controllare il corretto funzionamento dei diversi sottositemi, come pure il verificarsi delle varie condizioni limite di operatività, come il caso dell'overflow.

EXPERT

Questo comando abilita la modalità di operazione uso laboratorio. In tale modalità l'operatore ha un completo accesso ai dettagli di configurazione dell'elettronica ed inoltre è implementata una diversa gestione delle acquisizioni idle.

VERBOSE

Configura la verbosità dei messaggi interni generati dal sistema embedded.

4.3 Gestione dei segnali

4.3.1 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.

4.3.2 Rotazione del file di log

Come risposta al segnale SIGHUP viene registrata la funzione di gestione *restart_daemon()*. Questa funzione chiude e riapre il file di log, ma prima della riapertura, il file viene rinominato usando come estensione la data e l'ora dell'operazione di rotazione.

Questa funzionalitá collegata al segnale SIGHUP ci consente di tenere sotto controllo le dimensioni del file di log. Ricordiamo infatti, che il sistema *embedded* lavora esclusivamente in RAM. Se questo viene usato in modo ininterrotto senza essere riavviato, i files di log possono raggiungere dimensioni tali da ostacolarne il funzionamento.

Capitolo 5

Il sistema di acquisizione

Come descritto nel pararafo principale la funzionalità di ogni quadrante è gestita in modo indipendente da una scheda analogica. In particolare l'associazione quadrante del rivelatore, scheda analogica, è la seguente:

- Q1 C3
- Q2 C3
- Q3 C1
- Q4 C4

È importante sottolineare che ogni scheda calda puó essere programmata in modo indipendente, con valori diversi dalle altre.

In generale, esclusi i livelli di bias e reset, tutte e quattro le schede saranno normalmente programmate con gli stessi valori e con lo stesso programma di sequenze.

5.1 Struttura dei dati del frame *elettronico*

L'elettronica di acquisizione immagazzina i dati nelle FIFO di quadrante etichettando ogni riga di ogni quadrante con un *header* di 4 *word* in cui:

- la prima è il codice identificativo di inizio frame e vale `0xFFFF`
- la seconda è il conteggio dei frame arrivati, conteggio che parte da 0 con il reset iniziale o col test della memoria.
- la terza *word* è il conteggio progressivo delle righe
- la quarta è il codice di fine intestazione e vale `0000`.

Una logica dedicata comprime la dinamica dei dati da 0-65535 a 1-65534 in modo da essere certi che i valori 0 e 65535 (FFFF) non possano essere dei pixel validi.

Il contatore progressivo di frame è di tipo pre-incremento. In altre parole il primo frame ricevuto dopo il reset è etichettato `1`. Anche il contatore progressivo di riga è di tipo pre-incremento, per cui anche la prima riga è etichettata col numero `1`.

5.2 Acquisizione del frame immagine

Prima dell'avvio della presa dati, il sistema esegue una serie di operazioni tra cui: la programmazione dei livelli di Vbias e Vreset, la programmazione della memoria di sequenza, il calcolo dei parametri dell'acquisizione, come il tempo di ciclo etc., l'accensione dell'alimentazioni del sensore ed infine la partenza del *sequencer*. Il successo di ognuna di questa operazioni è condizionato dallo stato dell'elettronica.

Una volta avviato il *sequencer* l'applicazione aspetta per un tempo di poco inferiore al tempo di integrazione e incomincia a verificare se vi siano dati pronti. Quando viene rilevata la presenza di un frame completo, viene effettuata la lettura dalle FIFO.

Durante queste operazioni viene sempre monitorato lo stato dei link ottici e delle alimentazioni e nel caso vengano individuati malfunzionamenti dell'elettronica, la misura in corso viene interrotta.

La misura puó essere sospesa anche dall'operatore. In punti non critici per l'acquisizione, essenzialmente durante l'integrazione e il trasferimento dell'immagine, la routine di acquisizione controlla l'eventuale presenza di richieste di interruzione e, in caso affermativo, procede alla corretta terminazione della misura.

In punti di esecuzione diversi da quelli sopra specificati, il sistema risulta completamente insensibile a qualunque richiesta di interruzione, richiesta che è presa comunque in esame all'uscita dalla sezione critica dell'acquisizione.

Esiste conseguentemente un leggere ritardo tra l'istante in cui il comando di interruzione viene inviato dall'utente e il momento in cui questo viene realmente processato dal programma e l'applicazione *middle-ware* ne deve tenere conto.

L'aquisizione opera in due modalità distinte: *singolo quadrante* e *full-frame*.

5.2.1 Acquisizione da singolo quadrante

L'acquisizione di un frame da singolo quadrante si differenzia sopratutto nella parte di processamento dei dati. I pixels prelevati dalla FIFO di quadrante sono infatti trascritti nella memoria allocata cosí come sono ricevuti dall'elettronica, senza cioè la preliminare operazione di sbroglio, riportando

anche le 4 word di intestazione che qualificano ogni riga. Inoltre l'unica misura possibile è in *singolo campionamento*.

Il frame risultante contiene informazioni utili per un'analisi dettagliata di eventuali malfunzionamenti dell'elettronica e/o del software.

5.2.2 Acquisizione *full frame*

In modalità *full array* è possibile eseguire sia misure di tipo *singolo campionamento*, *double-correlated* e *multisampling*. Le ultime due opzioni non sono disponibili nel modo ad un solo quadrante.

Double-correlated

Questa tecnica di misura consente di rimuovere il valore dell' offset introdotto dal reset iniziale dell'*array* e consiste nel leggere due volte l'uscita dal sensore: una volta subito dopo il reset e dopo un intervallo di tempo pari al tempo di integrazione.

L'operazione di pre-processamento dell'immagine finale, che consiste nella sottrazione tra i due frames, viene eseguita direttamente dall'applicazione `server104` subito dopo lo sbroglio dei dati.

La tecnica di eliminare i possibili valori estremi (vedi 2.3), insieme all'osservazione che a parte il rumore non sono possibili valori negativi, permette di ottenere risultati a 16 bit. Per non nascondere la statistica del rumore (cosa che introdurrebbe un bias sistematico) per i pixel che hanno un valore prossimo a zero, al risultato viene aggiunto un valore costante, pari a 500, spostando la finestra dei valori dell'1%. Tale spostamento non ha effetto sul range dinamico, che risulta molto più limitato dalla capacità intrinseca del rivelatore.

Multicampionamento

La tecnica di multicampionamento consiste nella lettura dei pixels dell'*array* ad istanti successivi del processo di integrazione, senza il reset dell'array dopo la lettura.

Questa metodo di misura è ampiamente usato per la riduzione del rumore di lettura ed è applicabile nel campo dei rivelatori infrarossi perché la carica accumulata in ciascun pixel viene letta da un circuito separato e non trasferita ad un nodo integratore come accade per i CCD ottici.

Per garantire la maggiore flessibilità del sistema di multicampionamento, abbiamo deciso di elencare i passi della misura all'interno un file ASCII, descrivendo le istruzioni di programma del *sequencer* attraverso un metalinguaggio.

All'interno del file, oltre a operazioni proprie del *sequencer* sono alternati comandi per la calibrazione, tramite l'accensione o lo spengnimento di lampade di opportune o inserzione/rimozione di una cella di assorbimento,

azioni che sono portate a termine da un sottositema esterno, che riceve i comandi tramite linea seriale.

Di seguito riportiamo come esempio, un file di multicampionamento, il cui funzionamento è descritto nei commenti riportati nel codice.

```
128 r   # 128 reset. Ogni files comincia con un reset
1 R     # Un reset con scansione senza conversione (dry read)
0. l    # prima lettura comprensiva di reset
12.24 L # seconda lettura dopo 12.24 secondi - il tempo di scansione
25.19 L # terza lettura

11.00 A # Calibrazione: accensione lampada A, integrazione e lettura
10.00 B # Calibrazione: accensione lampada B, integrazione e lettura
30.00 A # Calibrazione: accensione lampada A, integrazione e lettura

12.00 L # settima lettura
25.00 L # lettura
12.00 L # lettura
25.00 L # lettura

10.00 C # Calibrazione: accensione lampada C, integrazione e lettura
Z       # Fine programma e restart
```

Esiste un limite al numero di istruzioni elencabili dato dalla dimensione in bytes della memoria di sequenza, pari a 1024 Word (vedi § 2.3) che si traduce in un numero di istruzioni massimo dell'ordine di 500. Infatti ogni operazione elementare richiede una media di due passi, in dipendenza del tipo di operazione (ad esempio il reset può richiederne molti di più) e della durata richiesta (il massimo tempo di integrazione per singolo passo è di 160 secondi circa.

Il file di multicampionamento viene scandito linea per linea e le meta-istruzioni tradotte in comandi intelleggibili dal *sequencer*. Le informazioni elaborate, dopo le necessarie operazioni di verifica e coerenza, vengono trascritte nella memoria interna del *sequencer* di ogni scheda analogica.

Il programma di controllo segue l'esecuzione temporale delle operazioni di multicampionamento attraverso la lettura del contatore di passo di programma del *sequencer*, attivando le operazioni di calibrazione al momento opportuno.

Il frame immagine generato ad ogni singola lettura dell'array viene inviato al programma di alto livello, e sarà compito di quest'ultimo processare in modo appropriato il set di immagini generate.

5.2.3 Lettura e trasferimento del frame

Come accennato nella sezione 5.2, la lettura del frame da parte dell'applicazione software ha inizio quando ciascuna FIFO di quadrante presenta un frame di quadro completo.

I pixels vengono prelevati in sequenza, uno dopo l'altro, a partire dalla FIFO del primo canale (Q3). Per ognuno l'applicazione `server104` controlla

la correttezza dello *header* di frame confrontando il numero di frame e di riga letti, con quelli attesi. Nel (rarissimo) caso venga appurata una dissincronizzazione tra questi valori, il software cerca di risincronizzare l'immagine: gli eventuali pixels persi risultano avere valore nullo identificabili quindi dal software di riduzione.

Successivamente viene eseguito lo sbroglio dei dati assegnando ciascun pixel letto alla cella di memoria con coordinate corrispondenti al quadrante in esame.

Al termine dell'operazione di lettura, i pixels risultano organizzati in una matrice quadrata 2048x2048 di tipo *unsigned short* che una riga alla volta, viene spedita all'applicazione *middle-ware* `gbridge`.

Nel trasferimento, ogni riga dell'immagine è preceduta da un'intestazione ASCII di 8 bytes che riporta il corrispondente numero di riga dell'immagine.

Per velocizzare l'operazione di trasferimento dei *frames*, l'applicazione `gbridge` non conferma la ricezione di ciascuna riga al processo mittente, ma si limita ad accertare la corrispondenza tra il numero di riga ricevuto e quello atteso.

L'invio dell'immagine puó non andare a buon fine in tre distinti casi: quando si verificano condizioni ripetute di timeout sul `socket dati` in uscita, quando il programma *middle-ware* ha problemi di ricezione, oppure se l'osservatore ha esplicitamente richiesto con un comando di ABORT o STOP, la fine della misura in corso.

Capitolo 6

Il codice del `server104`

Il codice dell'applicazione `server104` è scritto in C e fa uso di librerie standard. Rimandiamo ad un successivo rapporto la descrizione dettagliata della sua struttura dal punto di vista della programmazione.

Quando eseguito in modalità *menù* necessita delle presenza della libreria *ncurses* per la parte grafica e della libreria *cfitsio* per la gestione dei files in formato FITS.

6.1 Struttura del codice

Il codice del programma `server104` è suddiviso in più files sorgenti e di intestazione (*header files*).

Abbiamo suddiviso idealmente il sistema nelle proprie funzionalità base (inizializzazione, comunicazione, acquisizione etc.) e a ognuna abbiamo associato un file sorgente con il codice operante su quella specifica parte, ottenendo la seguente struttura:

- inizializzazione ed avvio del sistema nelle due diverse modalità
- interpretazione ed esecuzione dei comandi diretti o ricevuti dal programma *middle-ware*
- analisi, lettura e scrittura dei pacchetti del protocollo comandi (modalità demone)
- gestione della comunicazione di rete a basso e medio livello (modalità demone)
- acquisizione e trasferimento dei *frames* immagine
- inizializzazione, programmazione e controllo dell'elettronica, comprensivo dei test di funzionamento
- funzioni varie di utilità

- gestione dell'interfaccia (modalità *menu*)

I file *header* comprendono le dichiarazioni delle funzioni, le strutture dati globali usate dal programma per rappresentare e configurare lo stato complessivo del sistema, le definzioni delle costanti caratteristiche del sistema, come gli indirizzi dei registri delle schede,gli identificativi numerici dei comandi,i codici di errore.

6.2 Compilazione automatizzata del codice

Per la compilazione del codice ci siamo appoggiati agli *autotools*, gli strumenti sviluppati dal progetto GNU per creare un sistema di compilazione indipendente dalla piattaforma e dal sistema operativo in cui opera, ovvero *autoconf*, *automake* e *libtool*.

Assieme, questi strumenti vengono utilizzati per creare uno script di shell portabile chiamato *configure*. Questo viene generato dal tool *autoconf* a partire da un file `configure.in`, composto da una serie di chiamate a macro m4, che si espandono verso script di shell a seconda dei parametri che vengono passati.

Lo script *configure* esplora l'ambiente di compilazione per determinare quali sono le librerie disponibili, le caratteristiche della piattaforma, dove si trovano le librerie e i loro header, e cosí via. Basandosi su queste informazioni, lo script modifica i flag per il compilatore, genera i Makefile e il file `config.h` con i simboli appropriati per il preprocessore.

I file generati da *configure* sono perció adattati allo specifico sistema dell'utente.

In particolare lo script *configure* crea ciascun `Makefile` da un modello di base, chiamato `Makefile.in`. Il tool *automake* è lo strumento per creare questi modelli e genera i file `Makefile.in` a partire dal file `Makefile.am` creato manualmente dallo sviluppatore.

Il pacchetto libtool è il terzo importante strumento GNU. Il suo fine quello di astrarre le idiosincrasie delle librerie condivise (le librerie statiche sono molto simili fra i vari sistemi UNIX-like, ma le condivise presentano maggiore facilità nella portabilità).

Il lavoro dello sviluppatore si riduce essenzialmente alla scrittura di due files: `configure.in` e `Makefile.am`.

Indice

Bibliografia

[1] "Protocollo per la comunicazione tra `gbridge` e `Giano GUI`", C.Baffa, E.Giani, memo, 2006, http://www.arcetri.astro.it/irlab/doc/giano/gbridgeprotocol_1_08.pdf

[2] "Tabelle dei valori e degli indirizzi dell' elettronica di Giano" Memo, Vers 1.6 8/2007, C.Baffa, V.Biliotti, E.Giani http://www.arcetri.astro.it/irlab/doc/giano/elettronica/Protocollo_ISA.pdf

[3] "Calcolo tempi sistema di acquisizione, V.Biliotti, memo, 2008, http://www.arcetri.astro.it/irlab/doc/giano/elettronica/tempi_sequencer.pdf

[4] La disribuzione **slax**, `http://www.slax.org/`