INAF-OAA Gruppo Strumentazione Infrarossa

Progetto Giano

Xill: il software di controllo del sistema della pre-slit di Giano

E.Giani, C.Baffa

Revision : 1.10

Rapporto Interno di Arcetri N° 8/2011

Sommario

Questo documento descrive il complesso hardware del sistema ottico di pre-slit di GIANO, e l'applicazione Xill *preposta al suo controllo.*

Il sistema di controllo è costituito da un pc ultracompatto, `xill`*, e dal software applicativo vero e proprio,* Xill*, per la parte di controllo. A questo si affianca un'applicazione separata per il controllo della camerina di guida.*

Il nome di questa applicazione è stato ripreso da quello di una razza di creature del gioco di ruolo "Dungeons & Dragons".[5]

Figura 1: *Aspetto di uno Xill*

1 Introduzione

La struttura generale del software di controllo di Giano è stata recentemente semplificata [6]. Il sistema di pre-slit di Giano è gestito da un PC embedded dedicato, `xill`, che riceve, da remoto, i comandi dalla GUI dello strumento.

Le funzionalità dell'unità di pre-slit, tranne quelle riguardanti il sistema di autoguida, sono gestite dall' applicazione *server* Xill.

Il sistema dell'autoguida è costituito dalla camera MaxCam CMD-9 della della *Finger Lakes Instrumentation* (Flicam), e dal relativo software.

Il software Xill si interfaccia sia con i moduli *software* del sistema di controllo di Giano, sia con i sotto-sistemi hardware della pre-slit come la `PowerBoard` e il controller `Thorlabs`.

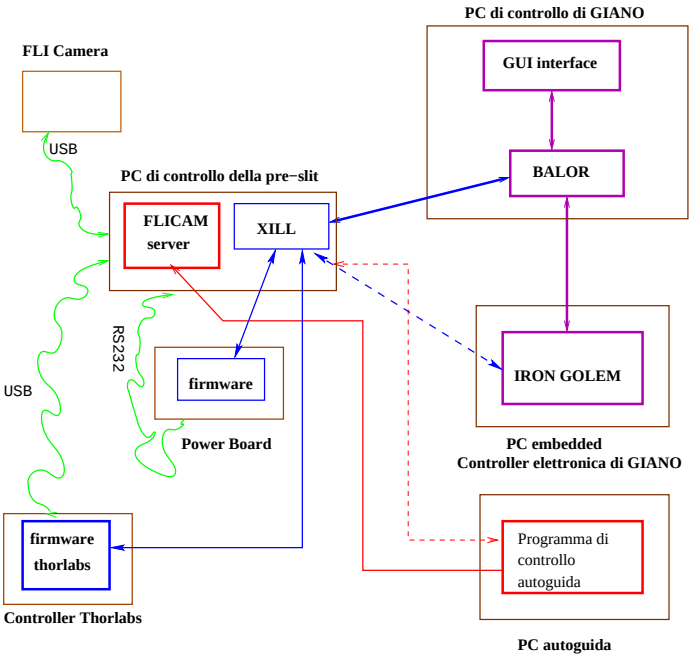

Figura 2: *Struttura dei moduli software e hardware di* Giano

Come si può vedere in Fig.2, la comunicazione con i device avviene attraverso diversi canali: ethernet, linea seriale RS232 e *bus* USB.

Il programma di pre-slit controlla il flusso di informazioni da e verso i diversi sottosistemi (rete, USB o seriale) in modo uniforme per mezzo di descrittori di file, analizzati con soluzioni di *I/O multiplexing*.

2 I sotto-sistemi dell'unità di pre-slit

Il sistema di pre-slit è composto da diversi sotto-sistemi distinti. Oltre alle ottiche fisse che non devono essere controllate, l'unità include i seguenti

device fisici:

- due lampade di calibrazione che possono essere accese o spente e inserite o disinserite lungo il cammino del fascio ottico
- due filtri grigi e uno shutter collegati a dei sistemi meccanici del tipo on/off, che possono essere inseriti o disinseriti
- una ruota con i dicroici e specchi
- la camera di autoguida Flicam.
- lo stage lineare motorizzato ad alta risoluzione della Physics Instrument M111.1DG per il posizionamento di precisione della fibra ottica.

3 La PowerBoard

La PowerBoard, controllata tramite linea seriale RS232 dedicata, gestisce l'alimentazione del sistema di pre-slit e comanda, su decisione dell'osservatore, i sistemi che accendono/spengono e/o inseriscono/disinseriscono i sotto-sistemi della pre-slit, tramite rele' e solenoidi.

La PowerBoard è stata sviluppata presso l'*Osservatorio di Arcetri* [7], intorno ad un microcontrollore PIC, per usi generali, ed il *firmware* che ne controlla il funzionamento è stato appositamente adattato alla configurazione di Giano.

Il protocollo di comunicazione implementato è molto semplice ed è descritto più avanti al § 7.2

4 Il sistema Thorlabs per il posizionamento delle ottiche di pre-slit

Il sistema di posizionamento dei dicroici e degli specchi si appoggia al modulo di rotazione motorizzato della Thorlabs CR1-Z7. Il rotatore fornisce un sistema di posizionamento accurato sull'intero intervallo di movimento.

Questo *device* è gestito dal controller TDC001 T-Cube della Thorlabs connesso tramite bus USB alla macchina di controllo del sistema di pre-slit, xill. L'alimentazione del controller è gestita direttamente dalla PowerBoard ed è comandata dal programma di interfaccia.

Il controller della Thorlabs usa un chip della FTDI (Future Technology Devices International) tipo FT232BM USB per comunicare con il PC di controllo. Questo chipset è un *device* USB UART che consente di vedere la periferica USB come un *device* RS232: il software puó accedere al *device* USB tramite le stesse routines utilizzate per una porta seriale standard.

La FTDI distribuisce un *driver* standard per Linux (*ftdio_sio*) che molte distribuzioni Linux includono direttamente come modulo di sistema e che

risulta comunque scaricabile dal sito Internet della ditta. In questo modo il device risulta riconosciuto ed inizializzato dalla maggior parte delle distribuzioni.

Nel nostro caso, volta connesso il controller `Thorlabs` alla macchina di controllo, questo è automaticamente associato a una delle porte seriali `ttyUSB*`.

Riportiamo, per riferimento, lo output del programma *dmesg*:

```
------------------------------------------------------------
[10246.215373] usb 4-4: new full speed USB {\it device} using
                        ohci_hcd and address 3
[10246.215373] usb 4-4: configuration #1 chosen from 1 choice
[10246.223308] ftdi_sio 4-4:1.0: FTDI USB Serial Device
                        converter detected
[10246.223370] usb 4-4: Detected FT232BM
[10246.223376] usb 4-4: Number of endpoints 2
[10246.223381] usb 4-4: Endpoint 1 MaxPacketSize 64
[10246.223386] usb 4-4: Endpoint 2 MaxPacketSize 64
[10246.223391] usb 4-4: Setting MaxPacketSize 64
[10246.227071] usb 4-4: FTDI USB Serial Device converter now
                        attached to ttyUSB0
------------------------------------------------------------
```

4.1 Lo stage lineare M111.1DG

La posizione in altezza della fibra ottica viene controllata dallo stage lineare ad alta precisione M111.1DG della Physik Instrumente, la cui movimentazione è gestita da un controller Mercury C-863.

L'interfacciamento tra il device Mercury ed il PC di controllo avviene tramite ethernet per mezzo di un modulo Xport connesso al bus seriale RS-232 del controller.

Il controller `Mercury` è accessibile attraverso la porta `10023` del terminal server della Lantronix usato per fornire la connettività di rete a diversi sottosistemi dell'elettronica di Giano.

Con il controller C-863 tutti i movimenti dei motori e degli stage connessi sono controllati dal software. Il sistema della PI supporta un insieme di comandi adatti alle operazioni di posizionamento e indipendenti dal tipo di motore o stage ad esso collegato.

Per maggiori dettagli rimandiamo al Manuale Utente del controller [3].

5 Il sistema embedded di Xill: hardware e sistema operativo

Il programma di controllo dell'unità di pre-slit Xill, insieme a quello di autoguida, sono eseguiti su un sistema *embedded*, `xill`, distinto da quello in uso per l'acquisizione.

La scelta è ricaduta su un hardware Koala microPC contenete una scheda mini-itx, equipaggiata da un processore VIA C7 ULV a 1,2 GHz a basso consumo, un controller video VIA CX700/VX700, un controller SATA, due interfaccie di rete, una porta seriale RS232, due USB 2.0 e il supporto per moduli di memoria Type I CompactFlash.

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

Per uniformità la scelta del Sistema Operativo è ricaduta inizialmente sulla distribuzione *Linux Slax*, installando l'ultima versione disponibile, la 6.1.2 con kernel linux 2.6.27.27.

Durante le prove di funzionamento del controller `Thorlabs` sono stati riscontrati una serie di problemi che sembravano strettamente legati a un supporto incompleto del chipset USB del PC o del driver usb-seriale *ftdi* da parte del *S.O*: alla disconnessione dell'alimentazione del device, il driver *ftdi* andava in crash.

É stato giudicato indispensabile un aggiornamento del kernel e del driver usb-seriale, nella speranza che il problema incontrato, probabilmente legato ad alcune incompatibilità dei chipset, fosse stato risolto nelle versioni piú recenti del kernel e del driver.

L'operazione di aggiornamento del kernel su una distribuzione *Slax* risulta un pó laboriosa per cui dopo qualche ricerca ci siamo indirizzati verso una nuova distribuzione, `Porteus`, evoluzione della *Slax*, frutto del lavoro di un gruppo di appassionati di informatica, nello spirito degli *Open Projects*[1].

Oltre alle classiche funzionalità di *Slax*, `Porteus` ne fornisce di nuove, ha un migliore supporto dei driver, un *kernel* aggiornato ed è costantemente manutenuta.

Le nuove prove eseguite sul sistema `Porteus` non hanno manifestato i problemi precedenti, avvallando l'ipotesi di un supporto incompleto dei chipset nelle versioni piú datate del kernel.

6 Le applicazioni *event driven*: il ***server*** Xill

L'applicazione Xill ha una struttura analoga a quella dei moduli software da cui prende vita il sistema di controllo di medio e basso livello di Giano.

Xill è un programma *event-driven*.

[1] La release originale prendeva il nome di *Slax Remix*.

Nel modello di programmazione *event driven* la struttura di un'applicazione è divisibile grosso modo in due gruppi: gli eventi e i gestori.

Con il termine 'evento' si intende comunemente un fatto di un certo rilievo. Anche in programmazione possiamo riferirci a questo termine con tale accezione: un evento è un avvenimento che ricopre un interesse per il sistema in esame. Nella maggior parte dei casi gli eventi sono generati esternamente al programma (ad esempio da device di input, da elementi di una GUI, da interrupt, da attività su *socket* etc.), alcuni possono essere originati da parti stesse del programma.

Il *gestore* rappresenta il modo con cui il sistema reagisce all'evento.

Il compito di un programma *event driven* consiste nel controllare in modo continuo e frequente il verificarsi degli eventi a cui è sensibile e, se rilevati, procedere a mandare in esecuzione il servizio associato. Quest'ultimo, per un corretto funzionamento del sistema, deve essere eseguito velocemente per garantire un rapido ritorno alla parte di codice che esegue il rilevamento degli eventi. Per soddisfare a tale richieste il gestore di evento non puó rimanere in uno stato di attesa per tempi lunghi, pena un sistema insensibile e sordo agli stimoli esterni o interni: il codice scritto delle applicazioni *event driven* deve essere necessariamente *non bloccante*.

Un'applicazione *event driven* deve essere dunque in grado di gestire dinamicamente il susseguirsi degli eventi. Questi possono essere processati man mano che vengono rilevati, oppure inseriti in una coda e gestiti in base alla loro priorità. Eventi non critici per il funzionamento del sistema possono subire, in questo modo, un certo ritardo nella loro esecuzione.

Qualora un gestore di evento richieda del tempo per portare a termine il suo lavoro, l'approccio migliore consiste nell'associare all'evento una routine di servizio che avvia la procedura e aggiungere la rilevazione della condizione di fine-procedura. Spesso questa serie di operazione è sotto il controllo di un timer, avviato alla partenza della procedura.

Il corpo principale di un programma *event driven* consiste di una parte di inizializzazione con la registrazione degli eventi e le relative routine di servizio (*callback handler*), seguito da un loop infinito, detto anche loop degli eventi o *main loop* che controlla il manifestarsi degli eventi e manda in esecuzione le funzioni associate.

6.1 Avvio di Xill

Xill è un'applicazione *event driven* che lavora da *server* di rete: processa i comandi ricevuti dalla GUI attraverso il modulo intermedio Balor.

Come tutte le applicazioni di questo tipo, all'avvio Xill apre un *socket* di ascolto in corrispondenza di una determinata porta di servizio restando in attesa di una richiesta di connessione: il server Xill è configurato per accettare una sola connessione per volta. La porta di servizio di Xill è la `8117`.

Una volta avventuta la connessione il programma procede a connettersi ai sotto-sistemi hardware presenti: la `PowerBoard`, il controller `Thorlabs` e il controller PI `Mercury`. I device fisici sono visti dal sistema operativo *Linux* attraverso i device file a loro associati dai rispettivi driver: nel caso specifico `/dev/ttyS0` per la `PowerBoard`, `/dev/ttyUSB0` per il controller della `Thorlabs` e un socket Internet per il controller della PI.

Abbiamo ritenuto non necessario dare all'applicazione la flessibilità necessaria a riconoscere descrittori variabil (del tipo `/dev/ttyUSBX`) perchè il disegno del sottosistema di pre-slit non prevede estensioni.

6.2 Gli eventi di Xill

Nei sistemi operativi *Unix-like* come *Linux* un descrittore di file (o *file descriptor*) è un numero intero non negativo che rappresenta un *file*, un *device* fisico, una *pipe* o un *socket* aperto da un processo e sul quale il processo può effettuare operazioni di lettura/scrittura.

Ognuna delle connessioni che Xill stabilisce è caratterizzata da un *file descriptor* e le attività di lettura/scrittura (I/O) che avvengono sui questi descrittori costituiscono l'insieme degli eventi che il sistema deve gestire.

A questi si aggiungono una serie di eventi *interni* al programma stesso, generati da alcuni timer[2].

A ciascuna sorgente di evento Xill associa un livello di priorità che determina l'ordine con cui questi eventi vengono processati dal loop degli eventi: quelli a priorità maggiore (valore negativo) hanno la precedenza sugli altri la cui esecuzione viene ritardata perchè non critica per il sistema

Nel codice del programma ogni sorgente di evento è rappresentata da una struttura[3] che riassume le caratteristiche della sorgente in questione, come il numero identificativo, il tipo, la priorità, e il puntatore alla routine di gestione dell'evento.

Il programma tiene traccia delle sorgenti registrate aggiungendole ad una lista che mantiene aggiornata: ogni nuovo evento registrato corrisponde ad un nuovo elemento della lista e in modo analogo, ogni evento rimosso, perché non piú *interessante* per il programma, porta alla cancellazione del corrispondente nodo della lista.

L'operazione di apertura e connessione a un nuovo device da parte di Xill corrisponde dunque alla registrazione di un nuovo evento e del suo gestore nella lista degli eventi analizzata dal loop principale.

[2]Le di callback temporizzate sono di aiuto in operazioni che devono essere ripetute periodicamente (lettura del sensore di temperatura), oppure che richiedono tempi di attesa lunghi rispetto al tempo di reazione propria di un programma *event driven* (attesa del fine-movimento delle ottiche o della fibra

[3]Struttura del tipo `Source_t`.

6.3 Il controllo degli eventi

La parte di rilevazione degli eventi di I/O sui descrittori è eseguita dal sistema operativo tramite la chiamata di sistema *poll*.

Il funzionamento di questa chiamata di sistema dipende da un insieme di descrittori di file che, organizzati in un array di strutture, individuano i canali di cui occorre seguire lo stato, in attesa di un loro cambiamento.

Ogni struttura raccoglie le informazioni di ciascun descrittore insieme al tipo di evento (lettura, scrittura o errore) che l'applicazione deve monitorare.

Questa chiamata di sistema lavora in *multiplexing*, tenendo sotto controllo piú dispositivi in contemporanea. Appena una delle richieste in esame risulta soddisfatta la funzione *poll* accede ai risultati, oppure blocca il processo se i descrittori non risultano pronti.

La tecnica del *multiplexing* unisce a un'elevata scalabilità il vantaggio di risolvere problemi di sincronizzazione, serializzando gli eventi.

6.4 I gestori degli eventi

Il sistema di controllo accede ad ognuno dei sotto-sistemi dell'unità di pre-slit come se fosse un file. Questo fatto consente di interagire con essi con le stesse funzioni usate per accedere ai normali file di dati: le chiamate di sistema *read* e *write*. È possibile pertanto scrivere allo stesso modo sulla porta seriale, sull' USB oppure su un *socket*, con l'accortezza che non tutte le operazioni sono disponibili per tutti questi diversi tipi di descrittore. Quello che differisce è il tipo di protocollo di comunicazione implementato dai diversi device. Per maggiori dettagli su questa parte, rimandiamo al § 7.

Le routine di servizio leggono dal rispettivo *file descriptor* e tenendo conto dei *task* di origine e destinazione del comando, invocano la funzione di decodifica incaricata della traduzione tra i diversi protocolli e scrivono la richiesta, o la risposta, sul device file di destinazione.

Le operazioni di lettura e scrittura di basso livello, che si appoggiano direttamente alle chiamate di sistema di I/O, sono raggruppate nella libreria *Irsock* ([8]) da noi sviluppata per raccogliere questa parte di codice usata in modo diffuso dalle nostre applicazioni.

7 I protocolli di comunicazione

Il software di controllo Xill comunica con tre processi distinti: il software *middle-ware* Balor, il firmware della `PowerBoard` e il firmware `Thorlabs`.

La comunicazione si appoggia su tre protocolli distinti che andiamo a illustrare nei prossimi paragrafi.

7.1 Protocollo di comunicazione Xill-Balor

I due processi comunicano per mezzo di pacchetti di dimensione nota, caratterizzati da un'intestazione di lunghezza fissa (16-byte) con le informazioni necessarie per decodificare l'eventuale area dati (*payload*).

Questo protocollo viene usato nella comunicazione tra tutti i moduli software del progetto Giano e descrive la lista dei comandi/messaggi che queste applicazioni si scambiano per comunicare.

Per maggiori dettagli sulle specifiche di questo protocollo e il set di comandi relativi al funzionamento della pre-slit, rimandiamo ai documenti [2] e [4].

7.2 Il protocollo di comunicazione host-PowerBoard

La comunicazione con il *firmware* della PowerBoard avviene seguendo il protocollo specificato da quest'ultima che accetta da seriale comandi costituiti da un solo carattere, **senza alcun carattere di terminazione.**

Le risposte sono stringhe di caratteri ASCII terminate dal carattere *line feed*.

La lista dei comandi viene pubblicata dal *firmware* della PowerBoard come risposta al comando ?.

```
 ------------ Help Comandi -------------
  1 =Reserve  2 =Reserve  3 =Reserve  4 =Reserve
  5 =Reserve
  6 =Reserve
  7 = Read  x y z Accelerometer
------------Controlli ------------------------
  A / a = On/Off Power_Lamp Calibrator
  B / b = On/Off Power_Halogen_Lamp
  C / c = HalogenLamp Inline/OffLine
  D / d = Filter1 InLine /Filter1 Offline
  E / e = Filter2 InLine /Filter2 Offline
  F / f = Light Stop Inline/Light Stop Offline
  G / g = On/Off Power_Torlab
  H / h = Intensity Lamp 100/10
  M / m = Read Temperature
  S     = System Status
  T / t = On/Off Flicam
  /= Reset_Cpu
  ?= Help
   ---------------------------------------
```

La ricezione di un comando che non compare nella lista precedente produce una risposta del tipo :

`bad command` / `Not implemented`

Nella tabella 1 riportiamo l'elenco delle risposte generate dal *firmware* in corrispondenza di ogni comando valido.

7.3 Il protocollo di comunicazione host-controllers Thorlabs APT

La documentazione allegata allo strumento descrive in modo dettagliato ed esauriente il protocollo di basso livello e i comandi implementati per la comunicazione tra il PC e le unità della famiglia di controller APT (Advanced Positioning Technology) della `Thorlabs`.

Il protocollo di comunicazione usato dai controllers della `Thorlabs` si basa su messaggi con una precisa struttura. Questi iniziano sempre con un'intestazione (*header*) di 6 bytes, talvolta seguita da un pacchetto di dati di dimensione variabile.

Per comandi elementari sono spesso sufficienti le sole informazioni codificate all'interno dell'intestazione, mentre ció non è piú vero per i comandi piú complessi che, ad esempio, devono specificare i parametri di movimento. Per questi ultimi lo *header* è seguito da un pacchetto di dati la cui presenza e dimensione in *byte* viene definita all'interno della stessa intestazione.

In questo modo il processo ricevente è in grado di determinare l'inizio e la fine di un nuovo messaggio.

Lo header ha la seguente struttura:

	byte0	byte1	byte2	byte3	byte4	byte5
Meaning if no data to follow	MESSAGE ID		param1	param2	source	dest
Meaning if data packet to follow	MESSAGE ID		datapacket length		source |0x80	dest

Figura 3: *Struttura dell'header del protocollo APT Thorlabs*

Il significato di alcuni campi dello *header* dipende strettamente dalla eventuale presenza di ulteriori informazioni.

Nel caso in cui il messaggio sia costituito dal solo *header*, il significato dei campi è il seguente:

message ID : rappresenta il codice del comando e corrisponde all'azione da intraprendere

param1 e param2 : sono rispettivamente il primo e il secondo parametro del comando, se questo li prevede, altrimenti valgono `0`.

source : è il modulo sorgente del messaggio

dest : rappresenta il modulo di destinazione

comando	**Risposta**
7	X = 32956; Y= 32189 ; Z = 45400;
a/A	Off → Power_Lamp Calibrator On → Power_Lamp Calibrator
b/B	Off → Power_Halogen_Lamp On → Power_Halogen_Lamp
c/C	→ Halogen_Lamp Offline → Halogen_Lamp Inline
d/D	→ Filter1 Offline → Filter1 Inline
e/E	→ Filter2 Offline → Filter2 Inline
f/F	→ Light Stop Offline → Light Stop Inline
g/G	→Off→ Power_Torlab →On→ Power_Torlab
h/H	→Intensity Lamp 10 →Intensity Lamp 100
t/T	→Flicam Off →Flicam On
M	→ Temperature :23.46
S	System Status: Power_Lamp Calibrator = Off Power_Halogen_Lamp = Off HalogenLamp = Offline Filter1 = OffLine Filter2 = OffLine Light Stop = Offline Power_Torlab = Off Intensity Lamp = 10
\	Reset_Cpu !!! Attendi !!! — Test LampCalibrator - Com1— Mauro Sozzi @ Maggio 2011 Versione 1.0 — Per Help Comandi premere ?—

Tabella 1: `Protocollo PowerBoard`

Se il bit piú significativo del `byte4` è posto a `1`, allora all'intestazione farà seguito un pacchetto con ulteriori dati e lo *header* sarà interpretato nel modo seguente:

message ID : rappresenta il codice del comando e corrisponde all'azione da intraprendere

datapacket length : numero di bytes che seguono lo *header*

source|0x80 : è il modulo sorgente del messaggio con il bit piú significato posto a `1`

dest : rappresenta il modulo di destinazione

Il campo sorgente e destinazione, nel caso di un sistema come il nostro in cui ogni modulo è rappresentato da un nodo USB separato[4], ha valori prefissati.

Quando il PC invia un messaggio al modulo USB (il controller), usa il codice identificativo `0x01` (*host*) per il campo `source` e `0x50` (*unità USB generica*) per il byte di destinazione: nei messaggi di risposta generati dal modulo, i bytes sorgente e destinazione risultano ovviamente invertiti.

Per ulteriori dettagli sul protocollo e le strutture dati, rimandiamo al documento [1].

7.4 Il protocollo di comunicazione host-controller PI Mercury

I sistemi della Physike Instrumente forniscono un set di comandi (GCS) indipendente dall'hardware. I comandi sono usati per configurare la modalità operativa, inizializzare il movimento dell'asse e chiedere i valori dei parametri di sistema e di movimento.

Eccetto alcuni comandi speciali, un comando GCS consiste di 3 caratteri, ad esempio `CMD`. La richiesta corrispondente viene formulata aggiungendo alla medesima sequenza il carattere '?'; ad es. `CMD?`.

Gli eventuali argomenti di un comando (numero identificativo dell'asse, del canale e parametri) devono essere separati da uno spazio. Inoltre la linea di comando termina con il carattere di *fine riga* `LF`.

$$\text{CMD}[< SP >< argument >]<\text{LF}>$$

Ad esempio:

$$\text{MOV}< SP >1< SP >10.0< LF >$$

I comandi speciali[5], cioè quelli usati per il polling, sono composti da un solo carattere ASCII e **non sono terminati** dal carattere `LF`.

Quando tutti gli argomenti sono opzionali e sono omessi, il comando è eseguito per tutti gli argomenti.

[4] Il valore del modulo sorgente e destinazione dipende dal tipo di sistema, cioè se è del tipo *card-slot* oppure no.

[5] #4, #5, #7, #8, #24 indicano rispettivamente il quarto, quinto, etc. carattere ascii

A esempio:

$RPA < LF >$

resetta tutti i parametri della memoria volatile del sistema ai valori di default.

La risposta ad un comando GCS è del tipo:

$[< argument > [SP < argument >]'' =''] < value >< LF >$

La risposta puó essere formattata su piú linee quando il comando richiede piú di un argomento oppure, come nell'esempio precedende, non specifica gli argomenti opzionali.

8 Funzionamento di Xill

8.1 Inizializzazione dell'unità di pre-slit

L'inizializzazione dei sotto-sistemi fisici comporta l'apertura[6] e connessione ai *device file* associati e, se questa operazione ha successo, la chiamata alla routine di inizializzazione dell'hardware.

Durante l'avvio di Xill un eventuale fallimento nel rilevamento di uno dei sottosistemi non porta alla terminazione del server. L'applicazione si limita a segnalare l'errore incontrato durante questa fase e sarà compito dell'operatore controllarne la causa: al successivo comando ricevuto dalla GUI con destinazione uno dei sottosistemi, il processo Xill ripete la procedura di connessione/inizializzazione dei device.

Non è dunque necessario che l'hardware della `PowerBoard`, della `Thorlabs` oppure del controller `Mercury` sia *on-line* al momento della connessione: il sistema è in grado di re-inizializzarsi se l'hardware non è stato correttamente rilevato in precedenza.

8.1.1 Inizializzazione della `PowerBoard`

La procedura di inizializzazione della `PowerBoard` consiste nell'invio del comando di stato e nella successiva lettura della risposta, in quanto il processore PIC in essa contenuta provvede all'inizializzazione locale. Se la `PowerBoard` funziona correttamente restituisce, per ogni linea di output, lo stato degli interruttori, la temperatura e le coordinate dell'accelerometro.

L'applicazione Xill considera la `PowerBoard` inizializzata solo quando entrambe le operazioni, quella di scrittura e quella successiva di lettura, sono state portate a termine con successo.

[6] I parametri di configurazione delle porte sono i seguenti: `9600 8 N 1` per la porta seriale e `115200 8 N 1` per la porta seriale su USB.

Uno dei problemi che si presenta nell'accesso alla `PowerBoard` è che il suo spegnimento oppure la disconnessione del cavo seriale sono eventi che non vengono rilevati dal sistema.

Si deve tener conto che la scrittura su un device seriale ha sempre successo tranne quando i suoi attributi di accesso in scrittura sono disabilitati. L'unico modo per 'intercettare' queste situazioni particolari è accertarsi che a ogni invio di un comando corrisponda *sempre* la ricezione di una risposta, come stabilito dal protocollo di comunicazione della `PowerBoard`.

Per ogni scrittura sul device seriale il programma Xill registra un timer a cui è associato una routine di servizio eseguita alla sua scadenza. Se entro il tempo programmato dal timer l'applicazione *server* non riceve alcuna risposta dalla seriale, la `PowerBoard` è considerata non operativa[7] e la *callback* di timeout va in esecuzione: la porta seriale viene chiusa e l'evento di I/O ad essa associato rimosso dalla lista.

8.1.2 Inizializzazione del sistema `Thorlabs`

Dall'inizializzazione della `PowerBoard` dipende quella del sistema di posizionamento delle ottiche: l'alimentazione del controller `Thorlabs` è pilotato da uno degli switch della `PowerBoard`.

La procedura di inizializzazione del sistema `Thorlabs` è relativamente articolata e dipende strettamente dall'ultima configurazione in cui è stato lasciato il device.

La routine che inizializza lo *hardware* `Thorlabs` invia una serie di richieste di protocollo per ottenere i valori delle variabili interne tra cui, in particolare, la posizione in micro-step della piattaforma di rotazione.

Il device `Thorlabs` è equipaggiato con un encoder incrementale che, ad ogni spengimento, perde le informazioni sulla sua posizione. Per ottenere un punto di riferimento valido, è indispensabile passare attraverso l'operazione di ricerca della *home*.

Questa procedura impiega piú di un minuto e la sua terminazione è segnalata da un messaggio di protocollo che induce la rotazione del motore alla posizione assoluta 2000. In questo modo il valore dell'encoder uguale a 0[8] è sempre interpretato dal software come equivalente a una sequenza di spegnimento-accensione.

Il sistema è ritenuto operativo al termine della procedura di *homing* oppure, per quanto detto sopra, quando l'inizializzazione restituisce una posizione diversa da 0.

[7] Questa condizione puó corrispondere alla `PowerBoard` non alimentata, oppure al cavo seriale scollegato.

[8] Vista la presenza di gioco sulla vite senza fine, è necessario procedere ad un'operazione di reset anche quando la posizione restituita differisce da 0 per meno del numero di micro-step di tolleranza, definito uguale a 10.

8.1.3 Inizializzazione del device `Mercury`

Il controller C-863 può immagazzinare i valori dei parametri nella memoria non volatile e può essere fornito con un set di valori predefiniti, specialmente quando è venduto con un stage di serie.

L'inizializzazione del controller PI `Mercury` dipende dalle caratteristiche dell'unità a esso connesso. Le specifiche hardware dello stage connesso e il setting del controllo a loop chiuso si riflettono in una serie di parametri del controller.

I parametri di funzionamento dell'unità sono caricati nella memoria volatile del controller, mentre quelli predefiniti sono registrati nella memoria non volatile. Entrambi possono essere visualizzati attraverso i comando `SPA?` e `SPE?`. Con questi comandi abbiamo ricavato il set di parametri caratteristici dello stage lineare MD111.1DG una volta che questo era stato inizializzato con il programma fornito per Windows.

Abbiamo poi confrontato i valori ottenuti con quelli presenti nella memoria non volatile generando una la lista con solo i parametri, e i loro valori, da riprogrammare nella memoria volatile per adeguare il controller al modello dello stage connesso. La lista ottenuta comprende 49 parametri che riportiamo in Appendice `B`.

Il protocollo di comunicazione non prevede alcuna conferma del comando eseguito. L'unico modo per sapere se un comando non di richiesta, è stato processato correttamente, consiste nel richiedere il codice di errore: se è 0 il comando è stato eseguito con successo, altrimenti il valore identifica il tipo di fallimento.

Durante la scrittura in memoria dei parametri di configurazione, è necessario sapere se la loro programmazione ha avuto successo, altrimenti il controller non processa i successivi ordini di movimento destinati allo stage: ad ogni scrittura di un comando ha sempre seguito una richiesta di errore.

La procedura di inizializzazione del `Mercury`, con la programmazione e il controllo di 49 comandi, impiega generalemente un tempo non inferiore a 30 sec. È possibile ridurre questo intervallo se risulta che lo stage lineare ha già eseguito un'operazione di riferimento dell'asse[9], informazione ottenuta come risposta al comando `FRF?`. La lettura del valore `1` implica che l'asse è 'referenziato', cioè il controller non è stato spento e i parametri relativi allo stage collegato sono ancora attivi nella memoria non volatile.

La procedura di *homing* richiede che il controller sia stato precedentemente abilitato per lavorare in modalità *closed-loop* (`SVO 1`) e di riferimento (*referencing*) (`RON 1`)[10].

[9] Questa operazione è analoga alla procedura di ricerca della *home*

[10] La modalità di riferimento posta uguale a 1 implica che per il riferimento dell'asse possono essere eseguiti solo i comandi FRF, FPL, FNL e l'uso di posizioni assolute con il comando POS non sono accettate.

L'operazione di riferimento dell'asse durante l'inizializzazione è indispensabile solo se il comando di `FRF?` restituisce 0.

8.2 Operatività di Xill

I pacchetti di protocollo provenienti dalla GUI, contenenti solo istruzioni di tipo COMANDO, sono processati dalla routine di servizio associata al *socket* di comunicazione Xill-Balor, analizzati dalla funzione di decodifica e tradotti nel *linguaggio* del processo di destinazione.

I pacchetti non sono rimossi dalla coda del *socket* finchè non sono stati processati. In questo modo un comando non viene mandato in esecuzione fino a quando il sistema non è pronto a gestirlo nel modo appropriato. Questo implica, d'altro canto, che la loro gestione deve avvenire in tempi rapidi, pena la scarsa responsività dell'applicazione.

I tipi di comando che giungono alla preslit sono di tre tipi: richiesta di stato (XSTATUS), selezione (SWITCH) e movimento (WHEEL, FIBER_MOVE)

8.2.1 Richiesta dello stato del sistema

La richiesta di stato del sistema viene sempre processata anche quando uno o entrambi i sotto-sistemi non risultano correttamente inizializzati: il programma GUI deve conoscere l'effettivo stato dell'unità di pre-slit.

8.2.2 Selezione dei device della `PowerBoard`

La selezione dei sotto-sistemi attivati dalla `PowerBoard` è un'operazione molto semplice: il codice traduce la richiesta nel corrispondente comando di protocollo (vedi § 7.2) che viene scritto direttamente sul device seriale e l'operazione è processata immediatamente dal *firmware* della `PowerBoard`.

L' operazione di scrittura sulla seriale rimane disabilitata fino a quando non viene ricevuta la risposta al comando eseguito. In questo modo il *firmware* riesce a processare tutti i caratteri-comando ricevuti.

Il pacchetto di protocollo corrispondente al comando in esecuzione, non viene rimosso dalla coda del *socket* finchè la seriale non viene riabilitata per la scrittura, ma il sistema non perde in responsività visto il breve tempo impiegato dalla `PowerBoard` a processare un comando.

8.2.3 Comando di movimento

Le richieste di movimento sono quelle che richiedono un tempo più lungo di esecuzione e il modo corretto di gestirle è attraverso la registrazione di timer e le funzioni di *callback* associate.

All'arrivo di un pacchetto di tipo comando, se il sistema è pronto per processarlo, viene invocata la routine di movimento associata. Questa invia la richiesta al device di destinazione, installa una funzione di timeout per

il controllo del movimento e ritorna alla funzione chiamante che rimuove il pacchetto dalla coda del *socket* e restituisce il controllo al loop degli eventi. In questo modo la terminazione del movimento è completamente gestita dalla funzione di callback registrata dal timer e il ciclo degli eventi è sempre pronto a gestire le nuove richieste di protocollo che giungono dalla GUI.

Posizionamento dello stage rotazionale della `Thorlabs`

Il posizionamento dello stage rotazionale è la procedura che impiega piú tempo per essere portata a termine. Il movimento dello stadio di rotazione alla posizione di destinazione avviene sempre per spostamenti assoluti e viene realizzato in due passaggi consecutivi per garantire un corretto recupero dei giochi della vite senza fine del motore.

Il primo movimento porta la piattaforma ad una posizione assoluta inferiore di 2000 micro-step[11] rispetto a quella richiesta; il secondo esegue il posizionamento definitivo. L'ultimo movimento, pari appunto a 2000 step, avviene sempre nella stessa direzione.

La necessità di usare movimenti assoluti e non relativi, che impiegano generalmente un tempo inferiore perchè tengono conto della direzione di minor spostamento, è dovuto al disegno della meccanica dell'oggetto in questione. Lo *stage* rotazionale in uso lavora, in realtà, come un sistema lineare. L'unico *limit switch* presente è stato configurato come *home* ma funziona anche da fine corsa. Se il movimento del motore avviene nella stessa direzione di ricerca della *home*, il raggiungimento del *limit switch* provoca la fine della rotazione.

Per ovviare a questo problema vengono usati solo spostamenti assoluti.

In corrispondenza dell'inizio di ogni movimento, il *software* di controllo registra un timer per gestire la condizione di *timeout* di movimento.

La conclusione di ogni spostamento, sia avvenuta in modo regolare, sia causata da una esplicita richiesta di stop, è comunicata dal controller `Thorlabs` al sistema *host* tramite un messaggio specifico di fine-movimento.

Il programma di controllo rimane in attesa della sua ricezione e nel frattempo processa solo quei comandi provenienti dalla GUI che non coinvolgono il device `Thorlabs`.

Se la fine del movimento non avviene entro il tempo registrato dal timeout, la sua scadenza manda in esecuzione il relativo gestore che invia al controller `Thorlabs` il comando di *stop* del movimento.

Il sistema di movimentazione `Thorlabs` è descritto da un insieme di stati e dalle transizioni tra questi. Ad esempio lo stato *init* indica l'inizializzazione dell'hardware, *idle* corrisponde al motore fermo in attesa di un comando, *move* indica invece il motore in movimento, lo stato *homing* corrisponde sempre ad un'operazione di movimento ma finalizzata alla ricerca della *home*.

[11] Il valore 2000 è stato scelto perchè corrisponde ad una rotazione di poco piú di 1^o.

In un qualunque istante il sistema si trova in uno solo di questi possibili stati e la transizione tra questi avviene in risposta ad uno specifico evento: inizializzazione, inizio movimento, fine movimento etc.

Per tradurre in software il comportamento dinamico del sistema, il codice si appoggia ad una variabile globale di stato.

Questa è costruita come una maschera a otto bit, in cui ogni bit descrive la condizione del motore a quell'istante.

La variabile passa dal valore *idle* a *move* all'inizio di un movimento a *end* quando questo è terminato. I bit restanti identificano delle condizioni particolari del sistema come quella di ricerca della *home*, di inizializzazione, di fine movimento, oppure di recupero dei giochi.

Posizionamento dello stage lineare della PI

Il controller non è in grado di conoscere la posizione assoluta dell'asse alla sua accensione perchè l'encoder di posizione del sistema usato fornisce solamente informazioni sul movimento relative.

Lo stage lineare può dunque essere posizionato solo dopo che la procedura di riferimento dell'asse è terminata, portando lo stage in corrispondenza del limit-switch negativo.

All'avvio del movimento Xill installa due *callback* richiamate periodicamente: la prima interroga il controller sulla posizione corrente (`POS?`), la seconda sullo stato dell'asse (#5). Quest'ultima viene rimossa dal ciclo degli eventi al termine del movimento, mentre la prima rimane sempre attiva per monitorare le variazioni in posizione della fibra[12].

[12]La callback periodica di lettura della posizione viene invocata con tempi scala diversi a seconda che lo stage lineare sia in movimento (circa 1 sec), oppure fermo (circa 45 sec.).

A Appendice: Problemi di *memory alignment* per i dati definiti dal protocollo Thorlabs

Il sistema in esame prevede la possibile presenza di molti processori e/o processi che comunicano l'uno con l'altro attraverso messaggi di varia natura. Affinchè lo scambio di messaggio tra due sistemi (hardware e/o software) lavori in modo corretto, il formato dei messaggi non deve presentare ambiguità.

In molti sistemi questo obiettivo viene raggiunto implementando i messaggi attraverso la definizione di strutture C/C++. L'uso di strutture C/C++ è un approccio semplice, ma puó nascondere alcune insidie legate alle differenti combinazioni di processori/compilatori.

La stessa struttura dati puó risultare definita in modo diverso, causando possibili incompatibilità nell'interfaccia di definizione che possono essere connesse a:

- le restrizioni dovute all'allineamento in memoria (*byte alignment*)
- l'ordinamento dei byte (*byte ordering*)

Per motivi di efficienza quasi tutti i compilatori eseguono l'operazione di allineamento di una struttura in memoria, detta *byte padding* che consiste nell'inserire dei byte *tampone* (in inglese *pad*) tra un campo e un altro in modo da "allinearli" opportunamente rispetto al loro indirizzo di memoria.

Se la stessa struttura messaggio viene usata in due diverse combinazioni di compilatore/processore, la quantità di byte tampone inserita dal compilatore puó risultare diversa e le due applicazioni che si appoggiano allo stesso *header file* contenente la definizione della struttura, possono essere incapaci di comunicare l'una con l'altra.

Di solito è buona pratica inserire dei *byte* tampone in tutte le strutture C che devono essere condivise tra sistemi che possono differire sia per il processore, sia per il compilatore. Questa soluzione non è peró adottabile nel caso in cui si debba ricorrere ad una struttura dati già esistente, che non puó essere modificata.

In questo caso possiamo agire solo sui parametri di compilazione, oppure definendo la struttura in questione attraverso le direttive #pragma o analoghe.

Il nostro sistema rientra nello schema precedente. Abbiamo infatti due sistemi diversi che si scambiano messaggi seguendo una definzione pre-definita, a cui il software da noi sviluppato si deve necessariamente adeguare.

Riferendosi ai possibili motivi di incompatibilità descritti in precedenza, possiamo sicuramente scartare il secondo, visto che la documentazione `Thorlabs` specifica chiaramente che per i valori di dimensione maggiore al *byte*, viene usato il formato Intel *little-endian*.

Per quanto riguarda invece i tipi di dati definiti dal protocollo `Thorlabs` queste sono strutture C/C++ che non risultano però allineate in memoria (vedi [1]).

Per garantire compatibilità tra i due sistemi, abbiamo definito le strutture dati C usate nel protocollo `Thorlabs` specificando l'attributo *packed*, attributo che implica l'uso della dimensione minima della memoria per rappresentare il tipo di dato e garantisce così che la struttura abbia la stessa dimensione su tutti i processori.

B Appendice:Lista dei registri del controller Mercury

Lista dei registri del controller `Mercury` e dei rispettivi valori di programmazione per la movimentazione dello stage lineare M111.1DG.

Registro	Valore	Registro	Valore
0x1	140	0x403	120
0x2	180	0x406	0
0x3	120	0x407	0
0x8	0.0137329	0x411	140
0xA	1.4999981	0x412	180
0xB	10.0000034	0x413	120
0xC	10.0000034	0x416	100
0xE	1310720	0x421	140
0xF	9	0x422	180
0x15	14.9999982	0x423	120
0x16	7.5000029	0x426	1000000
0x17	7.5000029	0x427	1000000
0x2F	7.4999960	0x431	140
0x30	0.0000000	0x432	180
0x32	0	0x433	120
0x3F	0.000000	0x436	1000000
0x49	0.7500023	0x437	1000000
0x4A	10.0000034	0x441	140
0x4B	10.0000034	0x442	180
0x4D	0	0x443	120
0x50	0.7500023	0x446	1000000
0x63	0.1999992	0x447	1000000
0x95	1.000	0x7000000	-14745.5999936
0x401	140	0x7000001	14745.5999936
0x402	180		

Riferimenti bibliografici

[1] "**Thorlabs APT Motor Controllers Host-Controller Communications Protocol.Draft**."

[2] "**Progetto Giano**. Protocollo di comunicazione tra Balor e Giano GUI", C.Baffa, E.Giani, Vers 1.39, Arcetri Technical Report 2-2012.

[3] "MS205E User Manual. C-863.11 Mercury",2011, PI

[4] "**Progetto Giano**. Comandi sintetici di stato per le applicazioni server di Giano", Vers.1.12, 2012,E.Giani, C.Baffa, Memo

[5] "Dungeons & Dragons - Monster Manual 3.5", 2003, J. Tweet, M. Cook e S. Williams, TSR.

[6] "Nuova struttura del software di basso livello, Giano Memo, 1/2011, C.Baffa, E.Giani

[7] Comunicazione privata, 1/2011, M.Sozzi

[8] "La libreria Irsock, in preparation, 2011, E.Giani

Indice