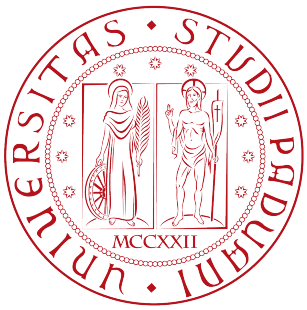




Università degli studi di Padova

Facoltà di Ingegneria

Tesi di Laurea in

Ingegneria Elettronica

# Scheda di acquisizione dati DAQ NI-6008USB: analisi e sviluppo di applicazioni di misura in LabVIEW

Relatore

Prof. Giada Giorgi

Candidato

Marco La Grassa

Anno Accademico 2009/2010

# Indice

# Introduzione

La seguente tesi è parte integrante di un progetto che riguarda la realizzazione di un completo sistema di misura, ovvero un servomeccanismo sul quale è stata montata una piattaforma in grado sia di traslare in senso orizzontale sia di ruotare su sé stessa compiendo un giro completo. Tale sistema di misura viene utilizzato per le misure di radiazioni elettromagnetiche: la struttura in legno del servomeccanismo associata alla possibilità di effettuare spostamenti molto precisi consente di effettuare un notevole numero di misure e di ridurre al minimo tutti quei fenomeni di riflessione associati alla propagazione delle onde. Lo schema a blocchi dell'intero progetto è mostrato in Fig. 1.

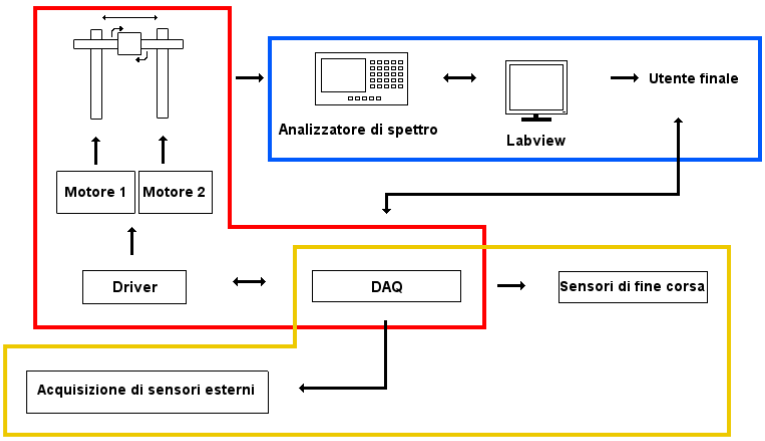



**Fig. 1:** Schema del progetto

Come è possibile notare dalla figura, il progetto è stato suddiviso in tre sottoprogetti: il riquadro di colore azzurro comprende la parte che effettua le misure attraverso un analizzatore di spettro collegato ad un'antenna posizionata sulla base rotante. Successivamente i risultati di queste misure vengono rielaborati e mostrati all'utente mediante uno strumento virtuale realizzato in LabView. Il rettangolo di colore giallo, invece, comprende quella parte

del progetto che ha come fulcro il dispositivo di controllo e il suo azionamento. In pratica si tratta di una scheda di acquisizione e di elaborazione dati, in grado di generare opportuni segnali di azionamento e al contempo di acquisire segnali sia digitali che analogici provenienti da sensori esterni. In realtà, i sensori esterni e di fine corsa indicati in figura non sono stati implementati fisicamente in questa versione del progetto, ma si è scelto di lasciarli comunque indicati nello schema in quanto rappresentano un possibile *upgrade* del sistema. Il rettangolo di colore rosso comprende infine la parte del progetto relativa al controllo della piattaforma, attraverso l'azionamento di due motori di tipo passo-passo unipolari.
In questa tesi ci si è occupati della parte relativa al sistema di controllo (rettangolo giallo), che è stato implementato mediante la scheda di acquisizione e di elaborazione dati NI DAQ-6008. Nonostante il lavoro sia stato svolto nell'ottica di voler realizzare il sistema di controllo per il servomeccanismo, la tesi ha lo scopo di proporsi come punto di partenza per tutti coloro che necessitano di:

- Sviluppare applicazioni in ambiente LabView sfruttando la libreria DAQmx.
- Utilizzare la scheda di acquisizione e di elaborazione dati NI DAQ-6008.

A tal fine, nel primo capitolo verranno descritte le caratteristiche hardware della scheda NI DAQ-6008, con particolare attenzione alla circuiteria di ingresso e di uscita dei canali analogici e digitali, nonchè delle modalità di collegamento della scheda con carichi esterni. Nel secondo capitolo si vedrà invece come sfruttare la libreria DAQmx per realizzare applicazioni LabView in modo efficiente. Il funzionamento interno della libreria risulta infatti assai più complesso di quanto inizialmente appaia, e solo dopo averlo compreso è possibile realizzare applicazioni in modo davvero efficiente, senza incorrere in errori legati all'inesperienza che causano comportamenti non previsti nei programmi. Nel terzo capitolo, infine, verranno discussi due esempi di utilizzo della scheda NI DAQ-6008, tra cui il programma che realizza il controllo dei motori passo-passo del servomeccanismo.

# Capitolo 1

# Scheda DAQ NI-6008USB

La scheda DAQ NI-6008USB della National Instruments è un sistema di acquisizione e condizionamento di segnali molto versatile e a basso costo. In particolare sono disponibili:

- 8 canali di ingresso analogici (AI)
- 2 canali di uscita analogici (AO)
- 12 canali digitali di ingresso/uscita (DIO)
- 1 contatore a 32 bit

Sebbene i canali di ingresso permettano di acquisire segnali provenienti da qualunque sorgente esterna, è necessario tenere presente che il massimo sampling rate è di 10kS/s e quindi lo strumento è adatto ad acquisire esclusivamente segnali a basse frequenze, come ad esempio quelli provenienti da sensori. Eventualmente è possibile acquistare un dispositivo di classe superiore, ad esempio il DAQ NI-6009USB, che presenta un sampling rate massimo di 48kS/s. In entrambi i casi, il grande vantaggio nell'utilizzare questi dispositivi è la possibilità di impostarne la configurazione e quindi di gestire l'acquisizione o generazione dati tramite un computer grazie al software LabVIEW.
In Fig. 1.1 è mostrata una fotografia della scheda; sono visibili in particolare i morsetti per collegare eventuali cavi esterni e l'ingresso per il cavo di alimentazione USB. A fianco di quest'ultimo è presente anche un led di stato verde, che lampeggia quando la scheda risulta alimentata. Le dimensioni del dispositivo invece sono riportate in Fig. 1.2 e come si può dedurre dalla fotografia in Fig. 1.1 risultano abbastanza ridotte, rendendo lo strumento poco ingombrante.


**Fig. 1.1:** Vista frontale del DAQ NI-6008USB

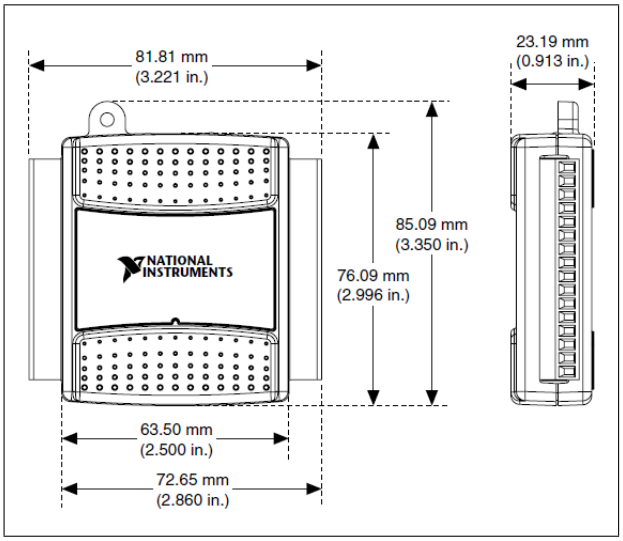



**Fig. 1.2:** Dimensioni del dispositivo DAQ

## 1.1 Schema a blocchi della scheda

In Fig. 1.3 è riportato lo schema a blocchi della scheda DAQ. Come si nota

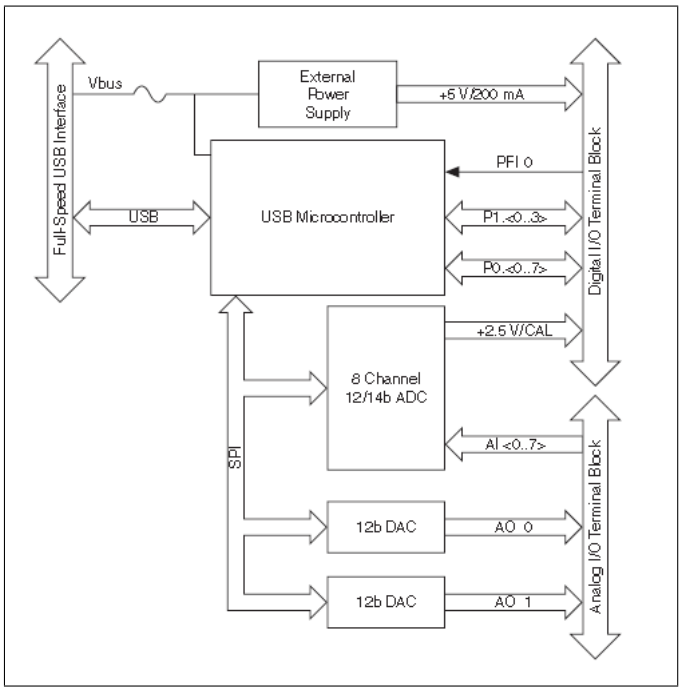



**Fig. 1.3:** Schema a blocchi della scheda DAQ NI-6008USB

dal disegno la scheda viene alimentata tramite un'interfaccia USB; nel caso di interfacciamento ad un computer, quindi, è quest'ultimo a fornire l'alimentazione necessaria al funzionamento del dispositivo.
La scheda mette a disposizione dell'utilizzatore due riferimenti di tensione ricavati dall'alimentazione USB, uno a 2.5V e l'altro a 5V; il primo riferimento può essere utilizzato come segnale per eseguire dei test sul dispositivo, mentre il riferimento a 5V è utile nel caso fosse necessario alimentare componenti esterni (massima corrente erogabile: 200mA).
Il microcontrollore, utilizzato per l'elaborazione dei segnali, può essere programmato mediante il software LabVIEW dopo aver installato i driver del dispositivo. Sono presenti anche due convertitori digitale-analogico (DAC) per i due canali analogici di uscita ed un convertitore analogico-digitale (ADC) ad approssimazioni successive per i canali analogici di ingresso.

## 1.2 Ingressi analogici

La scheda DAQ NI-6008USB dispone di 8 canali di ingresso analogici, che possono essere configurati in modo da effettuare sia misure differenziali sia

misure riferite a massa (RSE). In ogni caso la massima tensione applicabile al singolo canale rispetto alla massa del dispositivo è di ± 10V, che porta ad una tensione massima misurabile (in modalità differenziale) pari a ± 20V.

### 1.2.1 Circuiteria di ingresso per segnali analogici

La Fig. 1.4 rappresenta la circuiteria di ingresso per i canali analogici della scheda DAQ. Al primo stadio è presente un circuito di condizionamento

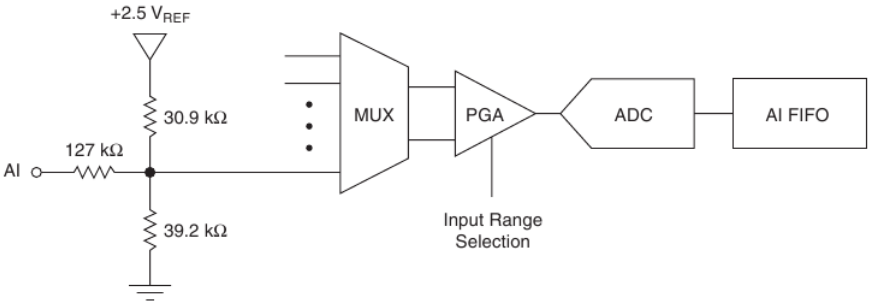



**Fig. 1.4:** Circuiteria di ingresso per segnali analogici

composto da tre resistenze, che purtroppo non viene minimamente motivato nel datasheet dello strumento. Dopo varie analisi, comunque, l'ipotesi più probabile è che il convertitore analogico-digitale (ADC) integrato nella scheda sia di tipo unipolare, mentre i canali analogici di ingresso del dispositivo DAQ sono in grado di funzionare con tensioni nel range [-10, +10]V rispetto alla massa della scheda. Risulta quindi necessario adattare il segnale proveniente da sorgenti esterne prima di inviarlo all'ingresso dell'ADC. Analizzando il circuito di condizionamento di Fig. 1.4 si ricava che con $AI = +10V$ la tensione vista dal blocco successivo (MUX) è di circa $3.6V$, mentre con $AI = -10V$ si ottengono circa $0.03V$. In altre parole, il circuito permette di convertire un segnale di tipo bipolare in un segnale unipolare, che può essere dunque analizzato dal convertitore analogico-digitale integrato nel dispositivo.

**MUX**. Al secondo stadio è presente un blocco di selezione (MUX) a cui sono collegati i canali di ingresso, che determina di volta in volta quale di essi trasferire al blocco successivo. Il concetto di *canale di ingresso* varia a seconda del tipo di misura. Se si effettua una misura riferita a massa, il canale è composto dal segnale di ingresso vero e proprio e dal riferimento a GND. Nel caso di misure differenziali, invece, il canale è composto dai terminali positivo e negativo ai quali sono collegati gli ingressi fisici veri e propri, mentre non è presente nessun riferimento alla massa del sistema. In emtrambi i casi, comunque, dal blocco di selezione MUX escono sempre due riferimenti,

che rappresentano l'input dello stadio successivo. L'utilità del MUX risulta evidente se si effettuano rilevazioni su più canali contemporaneamente. Infatti il sistema dispone di un solo convertitore ADC condiviso da tutti i canali analogici, per cui risulta indispensabile prevedere una qualche tecnica di selezione temporale degli input da digitalizzare. Il MUX risulterebbe indispensabile anche se le misure non avvenissero "contemporaneamente", in quanto bisognerebbe comunque selezionare un canale diverso per ogni misura.

**PGA**. Il terzo stadio è costituito da un amplificatore a guadagno programmabile (Programmable Gain Amplifier) che viene configurato via software in modo del tutto trasparente all'utente utilizzatore. Il fattore di amplificazione viene determinato automaticamente in base al range di variazione del segnale di ingresso specificato nell'applicazione, e varia a seconda del tipo di misura richiesta. Nel caso di misure differenziali i valori possibili di guadagno sono 1, 2, 4, 5, 8, 10, 16, 20, mentre per misure riferite a massa (RSE) il guadagno è sempre unitario.

**ADC**. Il convertitore analogico-digitale digitalizza il segnale analogico proveniente dal canale selezionato dal MUX; se la misura effettuata è di tipo differenziale la parola digitale in uscita è a 12 bits, mentre per misure riferite a massa la codifica è a 11 bits.
Il funzionamento dell'ADC si basa sul metodo della approssimazioni successive, uno tra i più diffusi in quanto consente di ottenere un buon compromesso tra velocità di conversione e risoluzione.

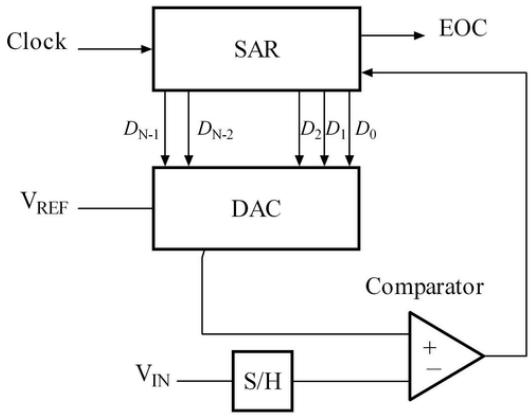



**Fig. 1.5:** Schema a blocchi di un ADC ad approssimazioni successive

In Fig. 1.5 è mostrata una tipica realizzazione circuitale per un convertitore di questo tipo; in particolare sono presenti quattro blocchi principali:

1. Un circuito di sample/hold per campionare (e mantenere) correttamente il segnale di input $V_{IN}$.

2. Un comparatore di tensione che compara il segnale in ingresso $V_{IN}$ con quello generato internamente dal convertitore digitale-analogico (DAC).

3. Un registro ad approssimazioni successive (SAR), progettato per fornire al DAC una appropriata rappresentazione digitale della tensione $V_{IN}$ in ingresso.

4. Un DAC che converte la parola digitale in uscita dal SAR in un segnale analogico da fornire al comparatore. $V_{REF}$ indica la tensione di riferimento da utilizzare per la conversione, e rappresenta quindi la massima tensione convertibile dal DAC.

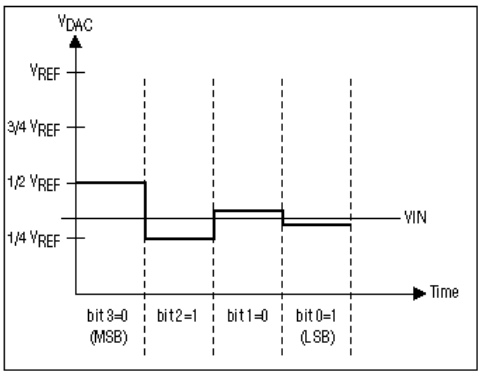



**Fig. 1.6:** Esempio di funzionamento di un convertitore ad approssimazioni successive a 4 bits

Il funzionamento del convertitore ad approssimazioni successive è abbastanza semplice e non è altro che un'implementazione dell'algoritmo di ricerca binaria. Un esempio di conversione per un ADC a 4 bit è mostrato in Fig. 1.6. All'inizio il registro SAR viene inizializzato in modo tale che la parola digitale in uscita abbia tutti i bit a 0, tranne quello più significativo (MSB) che invece è posto ad 1. Il codice così formato viene dato in ingresso al DAC, che lo converte nella rispettiva tensione (in questo caso $V_{REF}/2$) e lo invia in ingresso al comparatore. A questo punto se la tensione generata dal DAC risulta maggiore del segnale in ingresso $V_{IN}$, il bit viene posto a 0 dal SAR, altrimenti viene lasciato a 1. Successivamente viene impostato a livello logico alto il bit successivo e vengono ripetuti gli stessi passaggi, fino a quando tutti i bit del SAR sono stati testati. Quando la conversione termina il segnale EOC (End Of Conversion) viene posto ad 1, per informare l'eventuale circuiteria che segue che la codifica digitale della tensione di ingresso è disponibile in uscita all'ADC.

**AI FIFO**. Quest'ultimo componente funge da buffer con una politica first-

in-first-out (FIFO) e memorizza i dati durante l'acquisizione per evitare che possano essere persi.

### 1.2.2 Tipologie di misure per segnali analogici

Prima di poter effettuare una misura con il DAQ è necessario specificare via software il *tipo* di misura da effettuare; questa scelta viene eseguita impostando opportunamente un parametro nello schema a blocchi di LabView, come verrà ampiamente discusso più avanti. Le misure disponibili sono di due tipi:

- Differenziali: si misura la differenza di potenziale tra due terminali.
- RSE (Referenced Single-Ended): viene misurato il potenziale di un terminale rispetto alla massa

#### Misure differenziali

Quando si configura il sistema per eseguire una misura differenziale, gli 8 ingressi analogici vengono raggruppati in 4 canali differenziali ben precisi, come riportato nello schema in Fig. 1.7, che rappresenta la vista laterale della scheda DAQ dalla parte degli ingressi/uscite analogici.

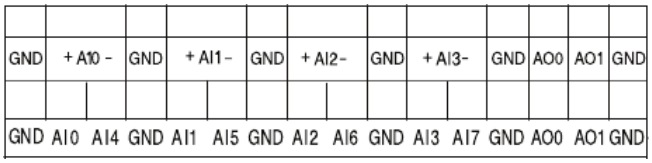



**Fig. 1.7:** Disposizione dei terminali degli ingressi analogici

Come mostra la figura, ognuno dei canali differenziali AI0, AI1, AI2, AI3 è composto da due canali di ingresso analogici. Ad esempio il canale differenziale AI0 è formato dai canali analogici AI0 e AI4, mentre il canale differenziale AI1 è composto dagli ingressi AI1 e AI5. Sono riportati anche dei segni "+" e "-" per indicare quale canale funge da terminale positivo e quale da terminale negativo. L'amplificatore differenziale a guadagno programmabile (blocco PGA) amplifica poi la differenza di potenziale tra i terminali positivo e negativo del canale differenziale su cui avviene la misura, come mostrato in Fig. 1.8, al fine di adattare il segnale per la conversione analogico-digitale che avviene nel blocco successivo. Da notare che nella figura è presente anche il segnale AIGND, che rappresenta la massa del circuito che genera il segnale da acquisire.
Come già accennato in precedenza, la massima tensione misurabile in modalità differenziale è $\pm 20$V (ovvero pari a 20V picco-picco); tuttavia la massima tensione applicabile ad un singolo canale analogico non può superare $\pm 10$V

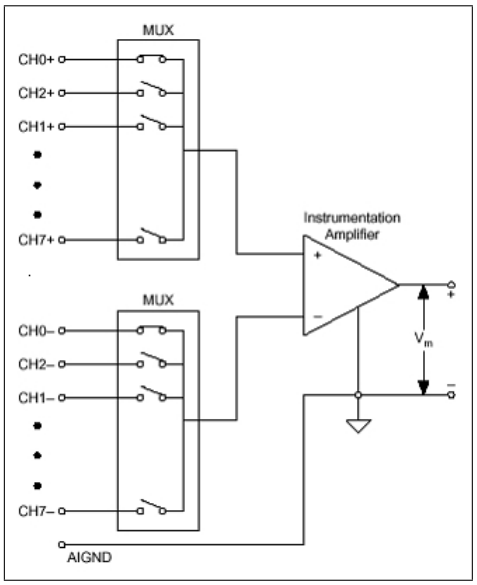



**Fig. 1.8:** Sistema di misura differenziale

rispetto alla massa del sistema. Se non si rispettano questi limiti il segnale di uscita risulta tagliato laddove $|V_{AI^+} - V_{AI^-}| > 20V$; le Fig. 1.9 e Fig. 1.10 mostrano rispettivamente un esempio di misura corretta (la differenza di potenziale tra i terminali è inferiore a 20V) ed uno di misura errata.

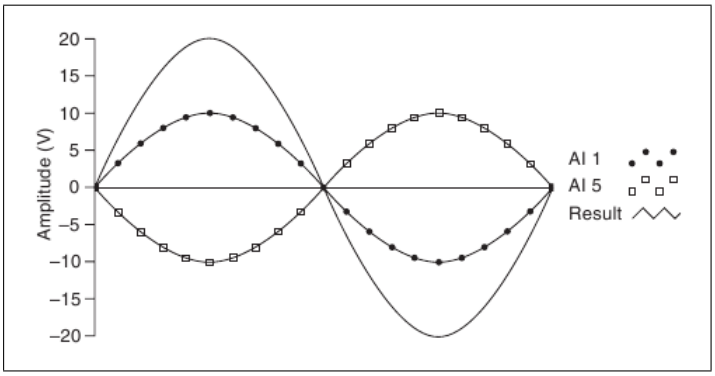



**Fig. 1.9:** Esempio di misura differenziale corretta

### Misure Referenced Single-Ended (RSE)

In una misura di tipo RSE viene misurata la differenza di potenziale tra un segnale collegato ad un canale di ingresso e la massa del sistema. A differenza della modalità differenziale, dunque, un canale è composto da un solo terminale di ingresso, in quanto il secondo terminale viene condiviso

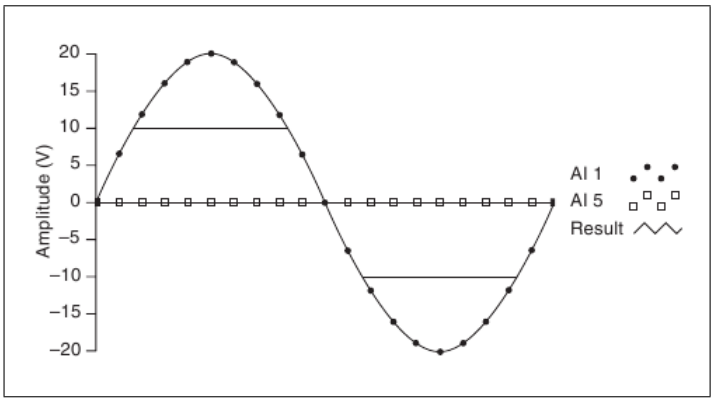



**Fig. 1.10:** Esempio di misura differenziale errata; il segnale viene tagliato qualora la differenza di potenziale ai terminali superi in modulo il valore massimo di 20V

da tutti gli altri canali ed è collegato alla massa del sistema, intesa come potenziale comune della scheda e della circuiteria esterna che genera il segnale da acquisire e misurare. In figura Fig. 1.11 è mostrato un esempio di configurazione per una misura riferita a massa. Questa configurazione

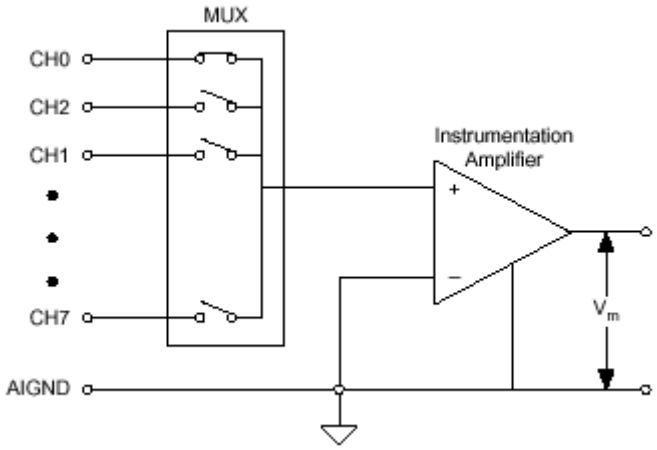



**Fig. 1.11:** Esempio di misura referenced single-ended

ha il vantaggio di raddoppiare il numero di canali disponibili per le misurazioni rispetto alla modalità differenziale. Tuttavia una misura differenziale normalmente restituisce dei valori più accurati, in quanto permette all'amplificatore di eliminare eventuali tensioni di modo comune o altri disturbi presenti in entrambi i terminali. La tensione di modo comune massima che il dispositivo è in grado di eliminare può essere calcolata con la seguente

formula:

$$V_{cm(max)} = \frac{MVW - \left[(V_{diff(max)} \times (Amplification)\right]}{2} \qquad [1]$$

dove:

- MVW è la massima tensione applicabile ai terminali del dispositivo
- $V_{diff(max)}$ è la massima tensione prevista ai capi dei terminali dell'amplificatore differenziale
- Amplification è il guadagno dell'amplificatore

Dalla formula risulta evidente che all'aumentare del guadagno differenziale migliora la capacità dell'amplificatore di eliminare la tensione di modo comune. Poichè quando il dispositivo è configurato per misure RSE il guadagno è sempre unitario, mentre per misure differenziali è in genere maggiore di uno, si ricava che in quest'ultima modalità eventuali tensioni di modo comune influenzano meno l'esito della misura, che quindi risulta più accurata. Un'ultima considerazione va fatta riguardo alla circuiteria di ingresso per segnali analogici nel caso in cui il dispositivo lavori in configurazione RSE. Nella Fig. 1.4 compaiono infatti tre resistenze in ingresso, una delle quali è direttamente connessa al riferimento di tensione a 2.5V. Questo comporta che un terminale di ingresso a cui non è collegato nessun segnale viene portato ad un potenziale di circa 1.4V a causa del partitore resistivo interno. Il partitore è influente solo se il terminale è lasciato flottante, mentre non influenza in alcun modo la misura se si collega un segnale esterno.

## 1.3 Uscite analogiche

La scheda DAQ NI-6008USB dispone di due canali di uscita analogici indipendenti tra loro, denominati AO0 e AO1 in Fig. 1.7; ogni canale è in grado di generare segnali nel range [0, 5]V e di erogare correnti fino ad un massimo di 5mA.

### 1.3.1 Circuiteria di uscita per segnali analogici

In Fig. 1.12 è mostrata la circuiteria di un canale di uscita analogico.

**DAC**. Al primo stadio si trova un convertitore digitale-analogico (DAC) che ha la funzione di convertire i segnali digitali provenienti dal microcontrollore in segnali analogici. Come si vede dalla Fig. 1.12 il DAC può generare solo segnali di tensione compresi tra 0V e 5V in riferimento alla massa del sistema.

**Output buffer**. Questo blocco funge da inseguitore di tensione e serve

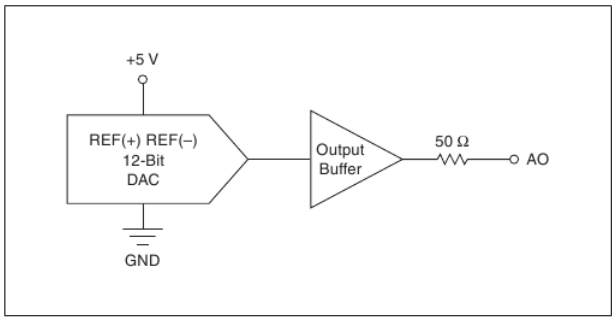



**Fig. 1.12:** Circuiteria dei canali di uscita analogici

per evitare che la tensione di uscita risulti distorta nel caso di impedenze di carico relativamente piccole. Idealmente dovrebbe presentare impedenza di ingresso infinita così da non influenzare il comportamento del DAC e impedenza di uscita nulla, così da evitare eventuali partitori resistivi con il carico. In realtà l'impedenza di uscita è di circa 50Ω, valore standard nella strumentazione elettronica.

Nel caso delle uscite analogiche è stata dunque fatta una scelta progettuale diversa rispetto agli ingressi. Infatti, mentre questi ultimi condividono il convertitore analogico-digitale, ogni canale di uscita possiede un proprio convertitore digitale-analogico. Con questa scelta non è necessaria la logica di selezione del canale tramite MUX, come avviene invece per gli ingressi; tuttavia comporta anche un maggior costo in quanto ogni canale deve prevedere un convertitore, motivo per cui probabilmente è stato preferito il MUX nel caso degli ingressi, che sono ben otto, mentre per le due uscite si è optato per l'ADC dedicato.

### 1.3.2 Glitches nei segnali di uscita

Quando si configura il DAQ per generare una forma d'onda in uscita, possono verificarsi picchi di tensione non previsti (glitches) nel segnale. Questo comportamento è normale ed è legato alla struttura interna del DAQ; si verifica in particolare quando variano i bit in ingresso al convertitore digitale-analogico, a causa delle cariche che vengono rilasciate da quest'ultimo. Il fenomeno è tanto più evidente quanto significativi sono i bit che variano e di conseguenza il massimo picco si ha al variare del bit più significativo (MSB) del DAC. Non è possibile risolvere direttamente questo problema agendo sul DAQ, ma bisogna progettare un apposito filtro passa-basso, ottimizzato in base alla frequenza e alla natura del segnale di uscita desiderato.

## 1.4 Rappresentazione a doppio bipolo per segnali analogici

Ora che sono stati analizzati i canali di ingresso e di uscita per i canali analogici, è possibile dare una rappresentazione a doppio bipolo del dispositivo, dal punto di vista dei segnali analogici. In particolare dalla Fig. 1.4 si ricava che l'impedenza di ingresso vale circa 144kΩ, nell'ipotesi che sia collegato un segnale esterno al terminale e che quindi il riferimento a 2.5V non influenzi la misura. In uscita invece il buffer permette di ottenere una impedenza di circa 50Ω. Complessivamente dunque, per segnali analogici, la scheda DAQ può essere rappresentata con il doppio bipolo resistivo di Fig. 1.13, dove $R_{in} = 144k\Omega$ e $R_{out} = 50\Omega$.

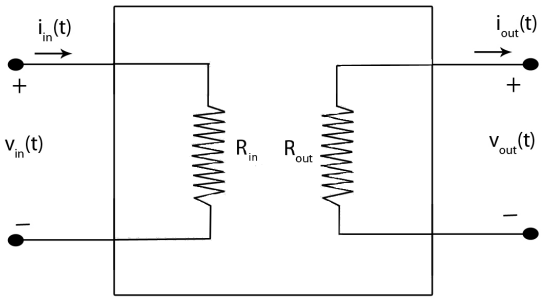



**Fig. 1.13:** Rappresentazione a doppio bipolo della scheda DAQ

## 1.5 Ingressi/Uscite digitali

La scheda DAQ NI-6008 mette a disposizione 12 linee digitali con logica TTL (0 - 5 V) che possono essere separatamente configurate sia come ingressi sia come uscite. La Fig. 1.14 mostra i pin riservati ai segnali digitali della scheda

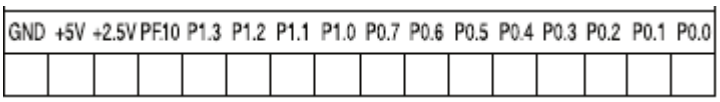



**Fig. 1.14:** Disposizione dei terminali per le linee digitali

DAQ; si osservano in particolare le 12 linee di ingresso/uscita P0.<0..7> e P1.<0..3>, la linea di massa GND e i due riferimenti di tensione a 2.5V e a 5V. Il pin PFI0 non ha la funzione di linea di input/output, ma può essere configurato per realizzare un contatore a 32 bits oppure per fungere da trigger, come descritto nel paragrafo 1.6.

### 1.5.1 Collegamento di un carico esterno

Quando si collega un carico esterno ad una linea digitale bisogna tenere presente che le porte sono di tipo open-drain e che quindi nella maggior parte dei casi è necessario prevedere un circuito di pull-up aggiuntivo. In realtà è presente una resistenza di pull-up interna alla scheda del valore di $4.7K\Omega$, collegata internamente al riferimento da 5V, che protegge la scheda da eventuali corto-circuiti e permette inoltre di alimentare direttamente un eventuale carico anche in assenza del circuito di pull-up esterno. In questa configurazione però la massima corrente erogabile è di circa $1.06mA$, calcolata nel caso limite in cui il carico ha resistenza nulla. Se il carico presenta un assorbimento di corrente maggiore di $1.06mA$ è necessario ricorrere ad uno schema di collegamento come quello riportato in Fig. 1.15, dove in aggiunta al pull-up interno alla scheda è stata inserita una resistenza di pull-up esterna per aumentare la corrente nel carico. In questa configurazione è im-

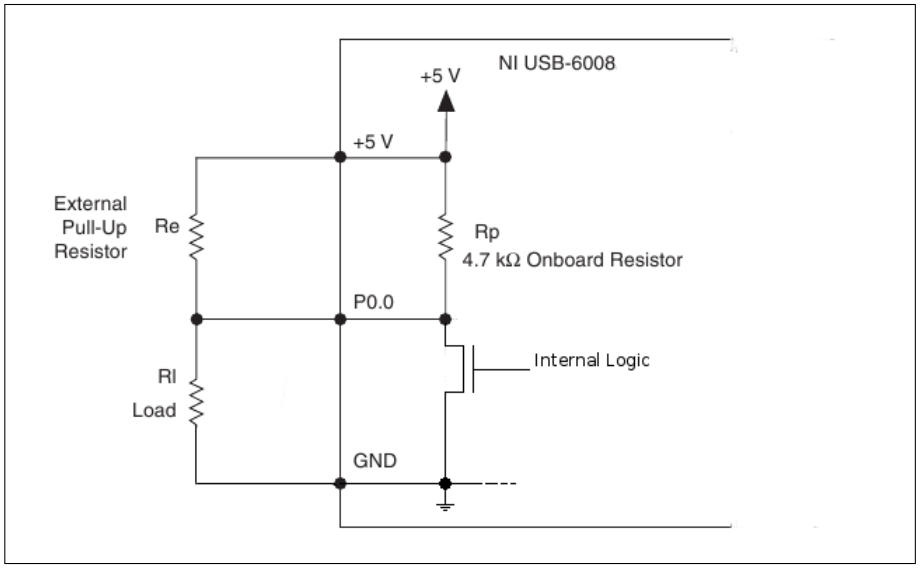



**Fig. 1.15:** Schema di collegamento di un carico esterno ad una linea digitale

portante scegliere il valore di $R_e$ in modo tale che la massima corrente nel carico sia sempre inferiore a $8.5mA$, che rappresenta il limite di corrente che ogni singola linea digitale può sopportare senza danneggiarsi.

### 1.5.2 Accensione e reset del dispositivo

Quando si collega il dispositivo all'alimentazione o quando lo si resetta per qualunque motivo, tutte le linee digitali vengono automaticamente configurate come ingressi ad alta impedenza. In questo stato, tuttavia, è comunque presente il debole circuito di pull-up interno della Fig. 1.15, per cui un eventuale carico verrebbe alimentato con una piccola corrente; collegando ad una

porta un diodo led con in serie una piccola resistenza, ad esempio, si osserverebbe che ad ogni accensione della scheda DAQ si accende anche il led. Questo comportamento può risultare del tutto innocuo in alcune situazioni, mentre può essere dannoso in altre. Un esempio di quest'ultimo caso verrà discusso più avanti e riguarda il controllo di un motore passo-passo tramite la scheda DAQ.

## 1.6 Funzioni del pin PFI0

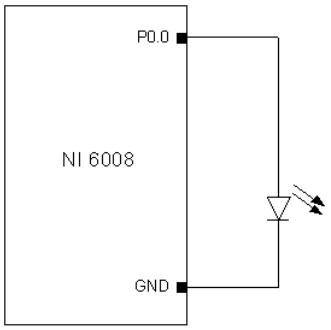
Come detto in precedenza, il pin PFI0 svolge funzioni particolari e non può essere utilizzato come una normale linea di ingresso/uscita digitale. Anch'esso, comunque, prevede segnali in logica TTL come le restanti linee digitali.

**Contatore a 32 bits**
La prima delle due funzioni che può svolgere il pin PFI0 è quella di realizzare un semplice contatore, sfruttando un registro a 32 bits interno alla scheda. Quando si configura il dispositivo per utilizzare il contatore, il registro interno viene inizializzato ad un dato valore, dopodichè vengono contati i fronti di discesa del segnale collegato al pin PFI0 (il software permette anche di sfruttare un clock interno come input del contatore, anche se di fatto non ha molte applicazioni). Un esempio di utilizzo del contatore interno verrà esposto più avanti quando si discuterà della libreria software per LabView "DAQ mx".

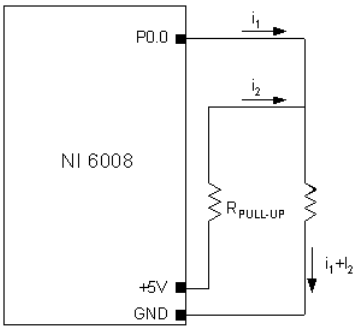
**Trigger digitale**
Quando si utilizza la scheda DAQ per acquisire un segnale analogico, è possibile configurare il pin PFI0 come trigger digitale. In questa modalità il sistema attende che si verifichi un fronte di salita o di discesa, a seconda di come è stato configurato il dispositivo, prima di iniziare l'acquisizione del segnale. Come nel caso del contatore, un esempio di utilizzo del trigger verrà esposto più avanti, discutendo della libreria software. Purtroppo la scheda DAQ 6008 è molto limitata per quanto riguarda la gestione del trigger, in quanto quella appena esposta è l'unica applicazione possibile; altri modelli, più evoluti e costosi, permettono ulteriori funzioni, ad esempio l'utilizzo del trigger per controllare la generazione (e quindi non solo l'acquisizione) dei campioni, sia analogici che digitali.

## 1.7 Esempi di collegamento

Vengono riportati di seguito due esempi di collegamento di un carico esterno alla scheda DAQ. Nell'esempio di Fig. 1.16a viene pilotato un led tramite la scheda DAQ. Notare che non è necessario inserire una resistenza di protezione in serie al led, in quanto è già presente la resistenza di pull-up da




1.16a: Collegamento di un LED




1.16b: Collegamento di un carico generico

**Fig. 1.16:** Esempi di collegamento ad un carico esterno

$4.7k\Omega$ interna alla scheda. Questo tipo di collegamento risulta possibile in quanto la piccola corrente che circola nel circuito (supponendo che la caduta di tensione nella giunzione del led sia di circa 1.8V, la corrente che circola vale $(5V - 1.8V)/4.7k\Omega \approx 0.7mA$) è comunque sufficiente a far accendere il led. In Fig. 1.16b viene invece mostrato come collegare un carico generico; in questo caso è stato inserito un resistore ausiliario di pull-up il cui unico scopo è di aumentare la corrente erogabile al carico. La scelta del resistore ausiliario può avvenire in vari modi; ad esempio, se si conosce a priori la corrente che deve circolare nel carico, è possibile procedere come segue:

1. si inserisce un potenziometro al posto di $R_{PULL-UP}$;
2. si inserisce un amperometro in serie al carico, per misurare il valore di corrente che circola;
3. si varia il valore di resistenza del potenziometro fino ad ottenere la corrente desiderata nel carico;
4. si misura il valore di resistenza del potenziometro: quel valore rappresenta la resistenza ottimale di pull-up.

Un esempio di collegamento errato è rappresentato dalla Fig. 1.17. Lo schema è molto simile a quello di Fig. 1.16a, ma questa volta il led è collegato al riferimento a 5V invece che ad una porta digitale. L'errore consiste nel fatto che in questo caso non è più presente la resistenza di pull-up interna che, nel caso di Fig. 1.16a, limita la circolazione di corrente nel led. Un collegamento di questo tipo, quindi, va assolutamente evitato in quanto si rischia di bruciare il led oltre che di danneggiare la scheda DAQ.

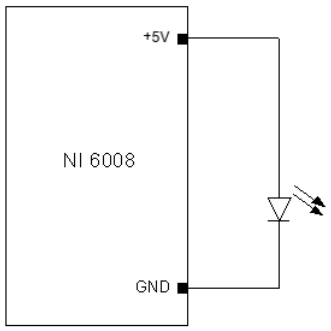



**Fig. 1.17:** Esempio di collegamento errato: manca una resistenza di limitazione della corrente

# Capitolo 2

# Libreria NI-DAQmx

Uno dei principali vantaggi nell'utilizzare un dispositivo di acquisizione dati come il NI DAQ-6008USB è, come già visto, la possibilità di programmarlo con linguaggi ad alto livello. Per farlo la National Instruments mette a disposizione una libreria software chiamata NI-DAQmx, che è un'evoluzione della libreria tradizionale NI-DAQ. Uno dei principali vantaggi della libreria DAQmx è dato dal fatto che le sue API (Application Programming Interface) sono le stesse per tutte le famiglie di dispositivi e per le varie funzionalità che ogni dispositivo offre. Questo significa che tutte le funzionalità di un dispositivo come il DAQ-6008, vengono programmate con lo stesso insieme di funzioni. Ad esempio programmare il dispositivo per acquisire un segnale esterno richiede le stesse funzioni necessarie a generare dei segnali digitali in uscita. Il risvolto pratico più interessante di questa caratteristica è che risulta sufficiente gestire pochissime funzioni di base per sviluppare un vasto numero di applicazioni di misura. Esiste comunque uno svantaggio, non particolarmente rilevante, in questo tipo di approccio. L'ambiente di sviluppo, nella quasi totalità dei casi, non è in grado di stabilire a priori quale dispositivo si sta programmando e di conseguenza la libreria non viene filtrata ma è proposta interamente all'utente. Si corre così il rischio di utilizzare funzioni e proprietà disponibili per alcuni dispositivi ma non per il proprio: il programma di fatto è corretto, ma non può funzionare con il dispositivo in uso e, di conseguenza, vengono generati errori in runtime che ne bloccano l'esecuzione.
La libreria DAQmx è disponibile per diversi linguaggi di programmazione, sia grafici come LabView sia testuali come il C/C++ e Visual Basic: in questa sede viene analizzata solo la versione per LabView, mentre si rimanda alla documentazione online per gli altri linguaggi. A causa del grandissimo numero di funzionalità disponibili, inoltre, è stato necessario eseguire una selezione degli argomenti da esporre. A tal proposito sono state selezionate alcune basilari applicazioni che tuttavia sono anche le più importanti e richieste nel settore delle misure, tra cui l'acquisizione di segnali analogici e

digitali e la generazione di segnali digitali.

## 2.1 Canali fisici, canali virtuali e tasks

Quando si lavora con un dispositivo programmato tramite la libreria DAQmx è indispensabile prendere dimestichezza con tre elementi basilari: i canali fisici, i canali virtuali e i tasks.

### 2.1.1 Canali fisici

Un canale fisico è un terminale o un pin del dispositivo in uso sul quale è possibile eseguire una misura, ad esempio generare un segnale digitale o acquisirne uno analogico. A seconda del tipo di applicazione, un canale fisico può essere composto da uno o più terminali; se per esempio si effettua una misura differenziale, il canale si compone di due terminali, mentre se si genera una parola digitale da 8 bits il canale è composto da 8 linee digitali. Ogni canale fisico, che nel caso più generale rappresenta un pin del dispositivo, possiede un nome univoco che rispetta la *convenzione dei nomi per i canali fisici* della libreria NI DAQmx. Secondo tale convenzione, il nome di un canale fisico consiste nell'*identificatore del dispositivo* e da uno slash (/), seguiti dall'*identificatore del canale*. L'identificatore del dispositivo viene generato in automatico dal sistema e di default è DevN, dove N è un numero progressivo assegnato in base all'ordine con cui i vari dispositivi vengono installati, ad esempio Dev0 e Dev1. L'identificatore del canale combina la tipologia del canale, come "analog input" (ai), analog output (ao) e contatore (ctr) con un numero intero, ad esempio ai2, ao0, ctr0. Per i canali digitali può essere specificata una porta, per esempio port0, che comprende tutte le linee disponibili, oppure si può specificare una linea in particolare all'interno di una porta con port0/line1. Nel caso in cui fosse necessario, è possibile anche specificare un range di canali tramite il carattere ":", per esempio Dev0/ai0:4, Dev0/port0/line0:3 o Dev0/port0:1.
Per agevolare l'utente nella programmazione, nei dispositivi NI DAQmx compatibili in generale accanto ai vari pin è scritto anche il nome del terminale secondo la convenzione appena esposta; ad ogni modo è possibile anche assegnare dei nomi arbitrari sia ai dispositivi, sia ai canali fisici.

### 2.1.2 Canali virtuali

Come suggerisce la dicitura, un canale virtuale è un oggetto software che incapsula un canale fisico, insieme ad altre proprietà che sono caratteristiche dell'operazione che si vuole effettuare, come il range della tensione in ingresso e la configurazione dei terminali. Queste informazioni, sebbene non siano strettamente necessarie da un punto di vista fisico per eseguire una misura, risultano molto utili in quanto consentono al sistema non solo di formattare

correttamente i dati ottenuti, ma anche di ottimizzare l'hardware interno per l'operazione che si vuole eseguire. Per esempio, il guadagno dell'amplificatore a guadagno programmabile (PGA) interno alla scheda DAQ-6008 viene calcolato proprio in base al range della tensione in ingresso e alla configurazione dei terminali scelta.

### 2.1.3 Tasks

Un task rappresenta concettualmente un'operazione che si vuole effettuare, sia essa una misura o la generazione di un segnale. Dal punto di vista software, un task è una collezione di uno o più canali virtuali con l'aggiunta di alcune proprietà, ad esempio la temporizzazione o il triggering dei segnali; tutti i canali virtuali appartenenti ad uno stesso task devono essere della stessa tipologia, cioè o di ingresso o di uscita. La gestione dei tasks in molte applicazioni avviene in maniera quasi automatica ed invisibile al programmatore, in quanto il software è in grado di creare e distruggere un task in modo del tutto indipendente. Ad esempio, quando si crea un canale virtuale senza specificare un task di appartenenza, viene creato automaticamente un nuovo task senza ulteriori notifiche all'utente. Il task appena creato verrà poi distrutto nel momento in cui il sistema riterrà opportuno farlo, in genere quando viene completata un'operazione di acquisizione o generazione di un segnale. Questo meccanismo ha tuttavia dei limiti, che riducono notevolmente l'efficienza di un'applicazione in alcuni casi particolari. Inoltre può risultare necessario intervenire sulla gestione di un task manualmente, ad esempio per obbligare il software a validare i parametri impostati dall'utente tramite un eventuale pannello frontale. Per soddisfare questo genere di esigenze, la libreria mette a disposizione delle particolari funzioni avanzate, che permettono di modificare lo stato di un task e quindi di controllarne l'esecuzione. La descrizione di queste funzioni verrà svolta più avanti, ma prima di tutto è necessario comprendere come il sistema gestisce i tasks.

#### Il modello a stati dei tasks

Al fine di rendere più efficiente la gestione dei tasks da parte del sistema e di semplificarne l'utilizzo, è stato introdotto nella libreria DAQmx un modello a stati. Un task, durante il suo ciclo di vita, può trovarsi in un solo stato alla volta e per cambiare stato deve effettuare una cosiddetta transizione. Gli stati previsti dal modello sono cinque, *unverified*, *verified*, *reserved*, *committed* e *running*; in generale un task, dal momento in cui viene creato a quello in cui viene distrutto, passa per tutti e cinque gli stati, che si comportano dunque come anelli di una catena. Per spiegare quali sono le varie transizioni ammesse tra gli stati è utile fare riferimento alla Fig. 2.1. Come si vede, è possibile passare da uno stato di livello inferiore, cioè più a sinistra nella figura, ad uno di livello superiore ad esso adiacente; non è invece possibile

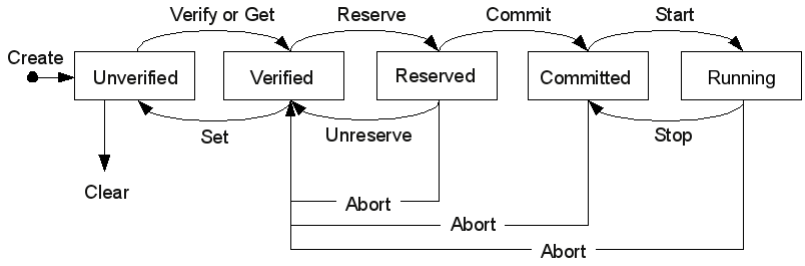



**Fig. 2.1:** Modello a stati dei tasks

"saltare" uno stato, passando ad esempio dallo stato "unverified" a quello "reserved" senza transitare per "verified". Questa limitazione è necessaria in quanto ogni stato della catena effettua determinate operazioni sul task, ognuna delle quali è indispensabile per la corretta esecuzione del task stesso. Se il task viene abortito (ad esempio premendo il tasto "Abort Execution" del pannello di LabView) passa automaticamente allo stato "verified", per poi venire eventualmente distrutto. Le transizioni tra i vari stati possono avvenire sia implicitamente, cioè è il sistema che si occupa di gestire il task, sia esplicitamente, quando l'utente chiama un'apposita funzione per modificare lo stato del task. In LabView, la funzione si richiama utilizzando il "Control Task function/VI" che, come descritto più avanti, ha tra i vari parametri lo stato in cui si vuole portare il task.
Come già accennato, in ogni stato vengono effettuate diverse operazioni fondamentali, necessarie per poter infine eseguire il task. Vediamo dunque più in dettaglio che cosa effettivamente accade ad un task quando si trova in uno dei cinque stati possibili.

**Stato "unverified"**
Quando un task viene creato o caricato, sia implicitamente che esplicitamente, si trova nello stato "unverified", dove è possibile configurare le proprietà del task come il timing, il triggering e gli attributi dei canali virtuali. Un task non può essere portato manualmente in questo stato, ma ci transita solo quando viene creato e quando viene distrutto.

**Stato "verified"**
Le proprietà impostate allo stato precedente vengono verificate durante la transizione allo stato "verified". Infatti, sebbene determinati valori non validi vengano individuati immediatamente quando l'utente li imposta, ce ne sono alcuni che non possono essere validati all'istante, in quanto dipendono da altri attributi e proprietà, nonchè dal dispositivo che si sta usando. In questo caso la verifica della correttezza avviene durante la trasizione "verify" ed eventuali errori o incompatibilità vengono riportate in quel momento. Se non si verificano errori, il task risulta verificato correttamente, e quindi passa allo stato "verified"; in caso contrario, rimane nello stato "unverified".

In alcuni casi, la libreria DAQmx può forzare i valori di alcuni attributi e proprietà durante il processo di verifica del task, invece che generare un errore. Questo avviene quando i valori impostati sui suddetti attributi e proprietà non riescono a passare la validazione così come sono stati specificati, e la forzatura ha un piccolo impatto sulla funzionalità del task.

**Stato "reserved"**
Tutte le risorse fisiche necessarie per il corretto funzionamento del task, ad esempio i canali fisici del dispositivo in uso, vengono acquisite solamente durante la transizione del task allo stato "reserved". La libreria cerca in pratica di assicurarsi l'accesso esclusivo alle risorse hardware che servono al task, in modo da evitare che altri tasks possano utilizzare le stesse risorse causando interferenze e malfunzionamenti. La transizione a questo stato fallisce nel caso in cui le risorse richieste siano già utilizzate, e quindi riservate, da un altro task. Se ciò avviene, il task rimane nello stato "verified", in caso contrario viene portato allo stato "reserved" e la transizione termina con successo.

**Stato "committed"**
In questo stato l'hardware viene programmato in base ad alcune impostazioni specificate precedentemente, come ad esempio la frequenza di un clock o i limiti delle tensioni di input per un canale analogico. Non tutte le impostazioni vengono trasferite all'hardware in questo momento; alcune infatti vanno programmate ogni volta che il task viene eseguito. Se l'operazione di "commit" ha successo, il task viene portato allo stato "committed", altrimenti rimane in "reserved".

**Stato "running"**
Quando il task inizia ad eseguire le operazioni per cui è stato programmato, passa dallo stato "committed" allo stato "running". Questa operazione generalmente non dovrebbe mai fallire, in quanto i parametri del task sono già stati validati e le risorse riservate negli stati precedenti; un fallimento, quindi, è dovuto ad una condizione eccezionale non prevista, che dovrebbe capitare molto raramente. La transizione allo stato "running", inoltre, non implica necessariamente l'inizio di un'acquisizione o generazione di campioni, ma provoca solo l'esecuzione delle operazioni previste per quel particolare task. Un tipico esempio di tale comportamento è quello di un'acquisizione di un segnale analogico subordinata ad un trigger digitale: il sistema non inizia ad acquisire i campioni finchè non si verifica l'evento specificato come trigger, sebbene il task sia già nello stato "running".
Quando il task esaurisce le operazioni previste, o quando viene esplicitamente interrotto, torna allo stato "committed". Anche in questo caso l'operazione non dovrebbe mai fallire; se si verifica una situazione eccezionale che causa

ne causa il fallimento, il task viene riportato allo stato "reserved", in quanto con tutta probabilità c'è stato un problema all'hardware del dispositivo e, quindi, è necessario eseguire nuovamente la transizione "commit".

**Transizioni di stato esplicite: quando effettuarle**

Dopo aver introdotto il modello a stati dei tasks, viene naturale chiedersi quando è opportuno alterare manualmente lo stato di un task e quando, invece, conviene lasciare che sia la libreria a gestirli autonomamente. Purtroppo non esiste una risposta precisa per questa domanda, in quanto dipende essenzialmente dall'applicazione che si sta sviluppando. In linea di principio, comunque, è possibile individuare alcuni casi tipici in cui conviene intervenire manualmente sui tasks; la lista seguente espone, per ogni stato disponbile, alcuni di questi scenari.

- **Verify**. Come già anticipato, se l'applicazione prevede che l'utente possa configurare un task impostando i parametri per il timing, per il triggering e per i canali virtuali, è possibile verificare manualmente il task per informare l'utente nel caso in cui i parametri inseriti non siano validi.

- **Reserve**. Si supponga di sviluppare un'applicazione in cui ci sono molti tasks che utilizzano lo stesso set di risorse fisiche, ad esempio le stesse linee di ingresso analogiche. Se uno di questi tasks effettua ripetutamente le stesse operazioni, è possibile che tra un'iterazione e l'altra le risorse hardware vengano acquisite da uno degli altri tasks. Effettuando una transizione esplicita allo stato "reserve" prima che il task inizi la sequenza di operazioni, ci si assicura che nessun altro task possa acquisire le risorse fisiche prima che la sequenza di operazioni sia stata completata.

- **Commit**. Quando l'applicazione che si sviluppa deve effettuare più acquisizioni o generazioni di campioni, eseguendo e bloccando ripetutamente un task, conviene effettuare una transizione esplicita allo stato "commit" prima che il task inizi la serie di operazioni. Quando il task si porta allo stato "commit", viene programmato l'hardware del dispositivo utilizzato in base alle impostazioni specificate nei vari blocchi. Forzando la transizione allo stato "commit" prima che il task inizi la serie di operazioni previste, assicura che l'hardware venga programmato un'unica volta, e non ad ogni esecuzione del task, il che si riflette in un considerabile aumento delle prestazioni. Ad esempio, se l'applicazione effettua ripetutamente l'acquisizione di un numero finito di campioni, con una temporizzazione fornita dall'hardware del dispositivo, il tempo richiesto per eseguire il task (ovvero per portarlo allo stato "running") diminuisce drasticamente se si effettua una transizione esplicita allo stato "commit" prima che il task inizi le acquisizioni.

- **Start**. Quando l'applicazione esegue ripetutamente operazioni di lettura e scrittura di campioni, ad esempio all'interno di un ciclo, è opportuno eseguire manualmente il task, tramite il blocco "Start Task function/VI". Entrambi i blocchi di lettura e scrittura, infatti, portano automaticamente il task dallo stato in cui si trova allo stato "running", passando per tutti gli stati intermedi; quando il blocco termina di leggere o generare i campioni, il task torna infine nello stato originario, seguendo il percorso inverso. Se non si interviene nella gestione del task questi automatismi, descritti più in dettaglio nei prossimi due paragrafi, vengono eseguiti ad ogni iterazione del ciclo, causando un notevole peggioramento delle prestazioni del dispositivo. Portando manualmente il task allo stato "running" tramite il blocco "Start Task", invece, ci si assicura che il task rimanga in tale stato fino a quando il ciclo non completa la propria esecuzione.

### Transizioni di stato implicite

Generalmente applicazioni non particolarmente complesse non richiedono di gestire manualmente le transizioni dei tasks tra i vari stati. Anche qualora l'utente decida di effettuare alcune transizioni esplicitamente, è improbabile che necessiti di un controllo completo sul task. Ad esempio potrebbe essere necessario, come verrà mostrato in seguito, eseguire esplicitamente il "commit" di un task che in quel momento si trova nello stato "verified". In questo caso, chiamando il "Control Task function/VI" di LabView ed impostando come azione "commit", per quanto è stato detto durante l'introduzione al modello a stati, il task verrà fatto implicitamente transitare per lo stato "reserved" prima di passare a "committed". Questa funzionalità è molto comoda perchè evita al programmatore di dover esplicitare tutte le transizioni intermedie tra i due stati, risparmiando così tempo e codice.
Un'altra situazione tipica in cui vengono eseguite delle transizioni implicite di stato è quando viene chiamata una funzione che richiede che il task sia in un particolare stato, mentre in quel momento non lo è. Se ciò accade, il task viene fatto transitare implicitamente nello stato richiesto dalla funzione, passando eventualmente per gli stati intermedi, come nel caso precedente. Alcuni esempi di operazioni che comportano questo tipo di transizioni sono:

- L'impostazione del valore di un attributo o proprietà, che comporta implicitamente la verifica del task.

- La chiamata alla funzione LabView "Read function/VI", il cui comportamento è descritto più avanti, che implicitamente esegue un'operazione di "commit" sul task. Se inoltre il parametro "Auto Start" della funzione è impostato a "True", il task viene anche implicitamente eseguito, portandolo allo stato "running".

- La chiamata alla funzione LabView "Write function/VI", che comporta una transizione allo stato "committed". Anche in questo caso la proprietà "Auto Start" impostata a "True" fa sì che il task venga anche implicitamente eseguito.

Quando viene chiamata una funzione che richiede una transizione di stato, come quelle appena descritte, c'è ovviamente la possibilità che il task sia già nello stato richiesto dalla funzione. Uno scenario simile avviene anche quando il programmatore richiede esplicitamente una transizione ad uno stato che corrisponde allo stato attuale del task in esame, tramite la funzione "Control Task function/VI". In entrambi i casi il sistema non esegue nuovamente l'operazione richiesta e non genera nessun errore. Ad esempio se il task è nello stato "reserved", una chiamata al "Control Task function/VI" con azione impostata a "reserve" non causerà una nuova acquisizione delle risorse, in quanto queste ultime sono già state acquisite e riservate al task.

### Transizioni a stati inferiori

Nel precedente paragrafo è stato visto come un task possa eseguire una o più transizioni implicite verso stati superiori, quando viene invocata una funzione che richiede che il task si trovi in un certo stato. Questo comportamento, comodo da un lato perchè evita al programmatore di portare manualmente il task nello stato richiesto dalla funzione, genera tuttavia un problema nel momento in cui tale funzione esaurisce le operazioni sul task. Si supponga, ad esempio, che il task sia nello stato "verified" e che venga chiamata la funzione "Read function/VI". Questa operazione genera una serie di transizioni implicite agli stati "reserved", "committed" ed infine "running". Quando il task viene interrotto, o esaurisce le operazioni richieste, viene automaticamente fatto transitare indietro, portandosi allo stato "committed". Si otterrebbe così che prima della chiamata alla funzione "Read function/VI" il task è nello stato "verified", quindi senza nessuna risorsa hardware riservata, mentre dopo la chiamata è nello stato "committed", cioè con tutte le risorse riservate. Per evitare questo comportamento non desiderato, quando avvengono transizioni implicite a stati inferiori, il sistema riporta il task nello stato in cui era *prima* della chiamata alla funzione, ripercorrendo a ritroso le operazioni eseguite durante la prima transizione implicita. Nell'esempio precedente, quindi, il task verrebbe riportato prima a "committed", poi a "reserved", dove vengono rilasciate le risorse occupate, quindi nuovamente a "verified". Questo meccanismo è completamente automatizzato e trasparente al programmatore, che quindi non deve minimamente preoccuparsene, così come non deve preoccuparsi delle transizioni implicite verso stati superiori.

## 2.2 Implementazione in LabView della libreria

Ora che sono stati introdotti i concetti chiave che caratterizzano la libreria DAQmx, è possibile analizzare i blocchi logici in ambiente LabView che ne realizzano le funzioni principali. Come è stato già spiegato all'inizio del capitolo una delle caratteristiche più accattivanti di questa libreria è la possibilità di gestire un grande numero di funzionalità di vari dispositivi con lo stesso set di funzioni. In LabView questo è possibile grazie ad una caratteristica nota come "polimorfismo", un concetto tipico di tutti i linguaggi di programmazione orientati agli oggetti. Con il polimorfismo, in sintesi, è possibile riunire sotto uno stesso nome tante funzioni che si distinguono quindi dal tipo e dal numero di parametri in ingresso che accettano. Quando la funzione viene invocata, in base ai parametri in ingresso verrà di volta in volta scelta l'implementazione consona al caso specifico. In Fig. 2.2 sono rappresentate le icone dei blocchi in ambiente LabView che verranno analizzati. Con questi blocchi è possibile, grazie al polimorfismo, realizzare un gran-

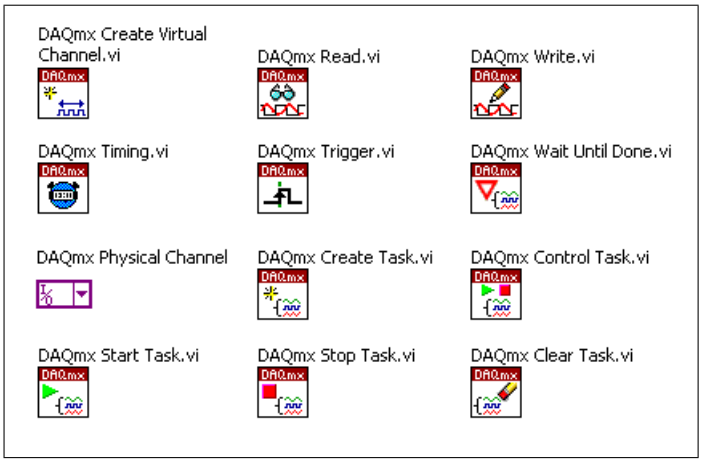



**Fig. 2.2:** Principali blocchi funzionali della libreria DAQmx in ambiente LabView

de numero di applicazioni, compatibili con tutti i dispositivi programmabili mediante la libreria DAQmx che supportano le funzionalità richieste di volta in volta dal programma. Prima di iniziare l'analisi, tuttavia, è necessario precisare che non è possibile fornire la descrizione di tutti i parametri di ingresso/uscita di ogni blocco, in quanto questi dipendono in parte dall'implementazione che viene scelta per quel particolare blocco polimorfico. Ad esempio il "Create Virtual Channel function/VI" accetterà parametri diversi a seconda che si selezioni, come tipologia di canale, "analog input" piuttosto che "digital input". In questa sede ci si limiterà dunque ad analizzare ogni blocco in alcune implementazioni particolari e, per ognuna, verrà fornita la descrizione dei parametri di ingresso e di uscita del blocco in esame.

### 2.2.1 La gestione degli errori

La libreria DAQmx gestisce le situazioni di errore mediante un apposito segnale, che normalmente viene fatto propagare da un blocco all'altro del programma. A tale proposito, tutti i blocchi che fanno parte della libreria DAQmx possiedono un ingresso e un'uscita in comune, chiamate rispettivamente "error in" e "error out". Collegando l'uscita "error out" di un blocco all'ingresso "error in" del blocco successivo, si ottiene il meccanismo di propagazione a catena appena citato. Il segnale di errore è composto da tre campi:

- **status**: è impostato a TRUE se si è verificato un errore o a FALSE per indicare o un warning o l'assenza di errori.
- **code**: indica il codice dell'errore o del warning. Se non ce ne sono, viene impostato a 0.
- **source**: identifica la sorgente e la causa dell'errore. In particolare indica il blocco da cui si è generato l'errore, quali ingressi sono coinvolti e come risolvere il problema.

Quando si verifica un errore in un blocco, questi campi vengono valorizzati di conseguenza, e l'errore si propaga inalterato fino all'ultimo blocco della libreria DAQmx presente nel programma.
Nei prossimi paragrafi verranno di proposito trascurati l'ingresso "error in" e l'uscita "error out" nella descrizione dei vari blocchi perchè, per quanto appena detto, svolgono sempre la stessa funzione.

### 2.2.2 VI "Create Virtual Channel"

Questo blocco è sicuramente uno tra i più utilizzati quando si programma con la libreria DAQmx, in quanto serve a creare, come suggerisce il nome, un nuovo canale virtuale da assegnare ad un task.

**Acquisizione di un segnale analogico**
Quando si vuole acquisire un generico segnale di tensione analogico, si sceglie l'opzione "AI Voltage" nel selettore polimorfico del blocco. In questa configurazione i parametri di ingresso sono:

- **input terminal configuration**: indica il tipo di misura che si vuole effettuare. I valori selezionabili se si utilizza il dispositivo DAQ 6008 sono "Differential" e "RSE", per eseguire misure rispettivamente differenziali e riferite a massa; sono presenti anche altre opzioni che tuttavia non sono compatibili con il dispositivo.
- **minimum value**, **maximum value**: indicano il minimo e massimo valore di tensione che ci si aspetta di misurare. Questi valori servono,

nel caso di misura differenziale, a determinare il fattore di amplificazione dell'amplificatore a guadagno programmabile (PGA) interno, e non influenzano in nessun modo i limiti fisici di tensione applicabili al dispositivo.

- **task in**: specifica il task al quale si vuole aggiungere questo canale virtuale. Se non viene specificato alcun task, la libreria ne crea uno in automatico, assegnandogli il canale virtuale.

- **physical channels**: indica i canali fisici da utilizzare in questo canale virtuale. È possibile collegare una stringa contenente la lista, o eventualmente il range, dei canali fisici desiderati; in alternativa creando una costante per l'attributo viene generato un oggetto di tipo "DAQmx physical channel constant" che permette di selezionare i canali fisici disponibili per il dispositivo in uso.

- **name to assign**: rappresenta il nome da assegnare al canale virtuale. Se non si specifica alcun nome, la libreria ne assegna uno in automatico, a partire dal canale fisico.

- **units**: indica l'unità di misura da utilizzare quando vengono ritornati i dati della misura. Nella maggior parte delle applicazioni è possibile ignorare questo attributo, che per default è impostato su "Volts". In alternativa è possibile utilizzare un'unità di misura personalizzata impostando l'attributo su "From Custom Scale".

- **custom scale name**: specifica il nome della scala personalizzata da utilizzare qualora "units" sia stato impostato su "From Custom Scale".

In uscita al blocco, oltre al già citato "error out", è presente il segnale "**task out**", ovvero il riferimento al task al quale appartiene il canale virtuale.

**Acquisizione di un segnale digitale**
Per acquisire un generico segnale digitale, occorre specificare "Digital Input" nel selettore polimorfico. I parametri di ingresso previsti in questa configurazione sono i seguenti:

- **task in**: vedi "Acquisizione di un segnale analogico".

- **lines**: specifica il nome delle linee o delle porte digitali da utilizzare per creare il canale virtuale.

- **name to assign**: vedi "Acquisizione di un segnale analogico".

- **line grouping**: se impostato su "one channel for all lines", tutte le linee digitali specificate su "lines" vengono raggruppate in un unico canale virtuale. Se si seleziona "one channel for each line", invece, viene

creato un canale virtuale per ogni linea digitale specificata. Tuttavia, se su "lines" sono stati inseriti i nomi di una o più porte digitali, quest'ultima opzione non è selezionabile.

In uscita al blocco è presente il segnale "**task out**", ovvero il riferimento al task al quale appartiene il canale virtuale.

**Generazione di un segnale analogico**
Per generare un segnale analogico in tensione, occorre specificare "AO Voltage" nel selettore polimorfico. In questa configurazione i parametri di ingresso previsti sono i seguenti:

- **output terminal configuration**: definisce la configurazione dei terminali di ingresso. La libreria mostra diverse scelte disponibili per questo ingresso, ma si deve scegliere necessariamente "RSE".
- **minimum value**, **maximum value**: analogamente al caso dell'acquisizione di un segnale analogico, questi ingressi indicano rispettivamente la minima e la massima tensione che ci si aspetta di generare.
- **task in**: vedi "Acquisizione di un segnale analogico".
- **physical channels**: vedi "Acquisizione di un segnale analogico".
- **name to assign**: vedi "Acquisizione di un segnale analogico".
- **units**: vedi "Acquisizione di un segnale analogico".
- **custom scale name**: vedi "Acquisizione di un segnale analogico".

In uscita al blocco è presente il segnale "**task out**", ovvero il riferimento al task al quale appartiene il canale virtuale.

**Generazione di un segnale digitale**
Per generare un segnale digitale, occorre specificare "Digital Output" nel selettore polimorfico. I parametri di ingresso e di uscita in questa configurazione sono gli stessi descritti nel caso "Acquisizione di un segnale digitale", a cui si rimanda.

### 2.2.3 VI "Read"

Il blocco "Read VI", come suggerisce il nome, serve per leggere dei campioni da un task o da uno o più canali virtuali. L'istanza del selettore polimorfico che viene di volta in volta selezionata permette di scegliere il formato dei dati da ritornare, ad esempio un array piuttosto che una "LabView Waveform", quanti campioni acquisire alla volta, e se leggere da uno o più canali.

**Lettura di una forma d'onda da un canale analogico**

Per leggere una forma d'onda da un canale analogico, occorre specificare nel selettore polimorfico "Analog Wfm 1Chan NSamp". La dicitura "1Chan" indica che la lettura viene effettuata da un unico canale virtuale, mentre "NSamp" significa che vengono acquisiti almeno due campioni. I parametri di ingresso previsti in questa configurazione sono i seguenti:

- **task/channels in**: specifica il nome del task o una lista di canali virtuali sui quali effettuare la lettura. In quest'ultimo caso, comunque, viene creato automaticamente anche un task al quale associare i canali virtuali specificati.

- **number of samples per channel**: specifica il numero di campioni da leggere. Se si lascia questo ingresso non connesso, o lo si imposta al valore "-1", la libreria DAQmx determina quanti campioni leggere in base alla modalità di campionamento, ovvero se vengono acquisiti campioni in modo continuo oppure in numero finito. Nel primo caso, vengono letti tutti i campioni disponibili in quel dato momento nel buffer. Nel secondo, invece, il VI attende che vengano acquisiti tutti i campioni previsti per poi leggerli. Le diverse modalità di campionamento, così come altre proprietà di temporizzazione del task, sono specificate mediante il blocco "Timing VI".

- **timeout**: specifica il tempo massimo, in secondi, da attendere affinchè i campioni diventino disponibili per essere letti. Se dopo questo tempo l'acquisizione dei campioni non è ancora terminata, il VI genera un errore e vengono ritornati i campioni letti prima dello scadere del "timeout". Impostando come valore -1, il VI attende indefinitamente, mentre con un timeout pari a 0 il VI cerca di leggere i campioni un'unica volta, ritornando un errore se in quell'istante non sono disponibili. In molti casi, comunque, è possibile lasciare scollegato questo ingresso, che utilizza come valore di default 10 secondi.

Le uscite del blocco sono "task out", ovvero il riferimento al task di appartenenza, e "data", che contiene i campioni letti nel formato "LabView waveform". I dati sono espressi secondo l'unità di misura specificata nel canale virtuale coinvolto nell'operazione di lettura.

**Lettura di una parola digitale**

Per leggere una parola digitale, occorre specificare "Digital 1D Bool 1Chan 1Samp" nel selettore polimorfico. La parola digitale da leggere è formata da tanti bits quante sono le linee digitali specificate nell'ingresso "lines" del canale virtuale da cui si vuole leggere. La dicitura "1D Bool", invece, indica che viene ritornata una matrice unidimensionale, cioè un array, di valori booleani. La parola digitale letta, dunque, viene rappresentata come un array in cui ogni elemento vale "TRUE" se la linea digitale corrispondente è a

livello logico alto, e "FALSE" in caso contrario. I parametri di ingresso in questa configurazione sono i seguenti:

- **task/channels in**: vedi "Lettura di una forma d'onda da un canale analogico".
- **timeout**: vedi "Lettura di una forma d'onda da un canale analogico".

Le uscite del blocco sono "task out" e "data", che contiene i campioni letti formattati come array di booleani.

### 2.2.4 VI "Write"

Il blocco "Write VI" viene utilizzato per generare dei campioni dalla scheda DAQ. Come al solito è possibile specificare varie opzioni tramite il selettore polimorfico, ad esempio se generare uno o più campioni, e se formattarli come "LabView waveform" piuttosto che come semplice array.

**Scrittura di campioni analogici**
Per generare dei semplici campioni analogici, occorre specificare "Analog 1D DBL 1Chan NSamp" nel selettore polimorfico. I parametri di ingresso del blocco "Write" in questa configurazione sono i seguenti:

- **task/channels in**: specifica il nome del task o una lista di canali virtuali coinvolti nell'operazione. In quest'ultimo caso, la libreria DAQmx crea in automatico un nuovo task a cui aggiunge i canali virtuali specificati.
- **data**: contiene i campioni da scrivere, formattati come un array di valori "floating point".
- **timeout**: specifica il numero di secondi da attendere affinchè i campioni possano essere scritti. La libreria esegue il controllo sul timeout solo se il "Write VI" deve aspettare un qualche evento prima di poter eseguire la scrittura dei campioni, altrimenti ignora il campo. Il valore di default è di 10 secondi; impostando come valore "-1" si permette al blocco di attendere indefinitamente. Infine, se il timeout è 0, il VI cerca di scrivere immediatamente tutti i campioni e, in caso di fallimento, viene ritornato un errore insieme al numero di campioni che sono stati scritti; quest'ultimo dato è reso disponibile all'uscita "number of samples written per channel" del blocco.
- **auto start**: se impostato a TRUE, indica che il blocco può implicitamente impostare il task allo stato "running" ed iniziare dunque la scrittura dei campioni automaticamente. Se viene specificato FALSE, invece, per iniziare la generazione dei campioni è necessario far transitare esplicitamente il task nello stato "running", mediante il VI "DAQmx Start Task"

Le uscite del blocco sono "task out" e "number of samples written per channel", che indica il numero di campioni correttamente scritti dal VI.
L'utilizzo delle uscite analogiche presenta varie problematiche, tra cui ad esempio il fatto che la scheda DAQ 6008 non dispone di un circuito di temporizzazione interno, per cui tutti i campioni, presenti nell'array "data", che vengono generati hanno una durata pari a 1ms. Questo valore è stato ricavato sperimentalmente, e rappresenta un limite fisico del dispositivo. Per modificare la durata dei campioni, il metodo più semplice è quello di inserire dei campioni aggiuntivi nell'array "data": ad esempio, se dopo ogni campione della sequenza originale se ne inserisce uno identico, si ottiene che ogni campione in uscita ha una durata doppia rispetto a prima. Un'altra limitazione nell'utilizzo delle uscite analogiche della scheda DAQ 6008, è che la corrente massima erogabile risulta pari a 5mA. Questo limite fisico esclude di fatto la possibilità di pilotare dispositivi esterni generici, rendendo indispensabile la presenza di uno stadio di potenza aggiuntivo, come avviene per le linee digitali.

**Scrittura di una parola digitale**
Per scrivere una parola digitale, occorre specificare "Digital 1D Bool 1Chan 1Samp" nel selettore polimorfico. La parola da scrivere è formata da tanti bits quante sono le linee digitali specificate nel canale virtuale. Anche in questo caso bisogna tenere presente che non è possibile pilotare direttamente grossi carichi mediante le uscite digitali, in quanto la massima corrente erogabile è di 8.5mA per ogni linea. Un'altra questione da prendere in considerazione, quando si realizza un software che sfrutta le uscite digitali della scheda DAQ 6008, è l'inizializzazione del dispositivo; a tal proposito si faccia riferimento al paragrafo 1.5.2.
I parametri del blocco in questa configurazione sono gli stessi visti nel caso della scrittura di campioni analogici. La sola differenza risiede nel fatto che questa volta l'ingresso "data" contiene un array di valori booleani e non più floating-point.

### 2.2.5 VI "Timing"

Il blocco "Timing VI" serve per realizzare un segnale di clock al fine di temporizzare l'acquisizione dei segnali analogici. La libreria mette a disposizione molte funzionalità per questo blocco, ma con la scheda DAQ 6008 è possibile impostare solo la modalità "Sample Clock". Inoltre, a causa del fatto che la scheda non possiede un circuito di temporizzazione interno per le uscite, nel caso di generazione di segnali questo blocco è inutilizzabile e bisogna necessariamente ricorrere ad una temporizzazione realizzata via software. Infine, sia nel caso dell'acquisizione che in quello della generazione di campioni, non inserendo il blocco "Timing VI" si ottiene il funzionamento "on demand", in

cui i campioni vengono acquisiti o generati immediatamente e senza alcuna temporizzazione, ad esclusione dei limiti fisici del dispositivo come, ad esempio, la massima frequenza di campionamento.

**Realizzare un segnale di clock per l'acquisizione**
Per implementare un segnale di clock al fine di regolare l'acquisizione di campioni analogici, occorre specificare "Sample Clock" nel selettore polimorfico. I parametri di ingresso in questa configurazione sono i seguenti:

- **task/channels in**: specifica il nome del task o la lista dei canali virtuali a cui applicare la temporizzazione. Specificando una lista di canali virtuali, inoltre, viene creato automaticamente un task.
- **rate**: indica la frequenza di campionamento (sampling rate), espressa come numero di campioni da acquisire al secondo.
- **source**: specifica il terminale da utilizzare come sorgente del clock, nel caso si voglia utilizzare un clock esterno. Nel caso della scheda DAQ 6008, però, è possibile utilizzare esclusivamente il clock interno al dispositivo, per cui questo terminale va lasciato scollegato.
- **active edge**: indica se campionare durante i fronti di salita oppure di discesa del clock. La scheda DAQ 6008 supporta solamente l'opzione "Rising", che esegue il campionamento durante i fronti di salita.
- **sample mode**: indica la modalità di campionamento da eseguire. Con la scheda DAQ 6008 è possibile specificare "Continuous Samples", in cui vengono acquisiti campioni fino a quando il task si trova nello stato "running", e "Finite Samples", in cui vengono acquisiti un numero finito di campioni.
- **samples per channel**: specifica il numero di campioni da acquisire (o generare, ma non con la scheda DAQ 6008) per ogni canale del task nel caso di campionamento di tipo "Finite Samples". Se invece si esegue un campionamento continuo, questo ingresso determina la dimensione del buffer riservato ai campioni acquisiti.

In uscita al blocco si trova, come di consueto, il riferimento al task corrente "task out". Infine, per quanto riguarda l'acquisizione di campioni digitali, è possibile effettuare solo una temporizzazione di tipo "on demand", per cui il blocco "Timing" non è utilizzabile.

### 2.2.6 VI "Trigger"

Il blocco "Trigger VI" serve per configurare le proprietà di *triggering* per il task corrente. Di tutte le funzionalità che mette a disposizione la libreria, utilizzando la scheda DAQ 6008 è possibile impostare esclusivamente "Start

Digital Edge". Inoltre, la funzionalità di trigger è disponibile solo se il canale virtuale è di tipo "Analog Input", mentre per tutti gli altri casi il blocco risulta inutilizzabile.

**Realizzare un trigger digitale**
Per implementare un trigger digitale, che regoli l'inizio dell'acquisizione di campioni analogici, occorre specificare "Start Digital Edge" nel selettore polimorfico. Inoltre, è necessario impostare una temporizzazione per il task che non sia di tipo "on demand"; nel caso della scheda DAQ 6008, quindi, bisogna necessariamente utilizzare un clock di tipo "Sample Clock".
I parametri di ingresso sono i seguenti:

- **task/channels in**: specifica il nome del task o la lista dei canali virtuali a cui collegare questo blocco. Specificando una lista di canali virtuali, inoltre, viene creato automaticamente un task.
- **source**: specifica il nome del terminale dove è presente un segnale digitale da usare come sorgente di trigger. Con la scheda DAQ 6008 è possibile impostare solo "PFI0".
- **edge**: indica se l'acquisizione dei campioni analogici deve iniziare al verificarsi di un fronte di salita oppure di discesa del segnale di trigger.

In uscita al blocco è presente il riferimento al task corrente "task out".

### 2.2.7 VI "Wait Until Done"

Il blocco "Wait Until Done", quando inserito in un task, fa sì che il sistema attenda fino a quando la generazione o l'acquisizione dei campioni termina, prima di compiere ulteriori operazioni sul task. Lo scopo principale di questo blocco è quello di assicurare al programmatore che il dispositivo abbia effettivamente completato la lettura o la scrittura dei campioni, altrimenti si rischia di bloccare l'esecuzione del task in modo prematuro, con conseguente perdita dei campioni. Ad esempio, se si utilizza un "Sample Clock" per acquisire 1000 campioni ad un rate di 100 campioni al secondo, ci si aspetta che il sistema impieghi 10 secondi prima che i campioni siano resi disponibili. Tuttavia, se dopo al "Sample Clock" si utilizza un blocco "Read", specificando di leggere, ad esempio, 200 campioni, quello che accade è che dopo soli due secondi la lettura dei campioni termina. Questo fatto si spiega in quanto il dispositivo inizia ad acquisire i campioni solo quando viene chiamato il blocco "Read", in quanto la chiamata fa transitare il task allo stato "running". Dopo due secondi il dispositivo avrà già acquisito i 200 campioni richiesti, pertanto il blocco "Read" termina la sua esecuzione e, di conseguenza, il task viene retrocesso ad uno stato inferiore, bloccando dunque l'acquisizione degli 800 campioni rimanenti. Un comportamento di questo tipo può risultare accettabile o meno in base all'applicazione che

si vuole sviluppare: qualora non lo sia, è possibile inserire il blocco "Wait Until Done" subito dopo il blocco "Read". Così facendo si forza il task a rimanere nello stato "running" fino all'effettiva conclusione dell'acquisizione dei campioni. Ovviamente gli stessi ragionamenti valgono nel caso in cui sia necessario generare dei campioni, piuttosto che acquisirli.
I parametri di ingresso del blocco "Wait Until Done" sono i seguenti:

- **task/channels in**: specifica il nome del task o una lista di canali virtuali a cui applicare il blocco. Specificando una lista di canali virtuali, viene creato automaticamente un task.

- **timeout**: indica il tempo massimo, in secondi, che il sistema può attendere affinchè l'acquisizione o la generazione dei campioni termini correttamente. Se viene superato questo termine, il VI ritorna un errore e non attende ulteriormente. Il valore di default è di 10 secondi; impostando -1 il sistema attende indefinitamente, mentre un timeout impostato a 0 fa sì che il VI controlli un'unica volta se la lettura o la generazione dei campioni è terminata, ritornando un errore qualora non lo sia.

In uscita al blocco è presente il riferimento al task corrente "task out".

### 2.2.8 "Physical Channel"

Questo blocco della libreria DAQmx permette di selezionare le linee di ingresso/uscita, sia analogiche che digitali, da collegare ai vari VI. Grazie al polimorfismo, il blocco è in grado di filtrare le linee che vengono visualizzate in base alla tipologia del terminale a cui è connesso. Ad esempio, se si collega al blocco "Create Virtual Channel" con selettore polimorfico impostato su "Analog Input", vengono mostrati solo gli ingressi analogici del dispositivo in uso; nel caso della scheda DAQ 6008, dunque, la lista è formata dalle linee AI<0-7>. Questo comportamento, attivato di default, è comunque modificabile tramite le opzioni del blocco, così come ad esempio la possibilità di selezionare più linee contemporaneamente.

### 2.2.9 VI "Create Task"

Il blocco "Create Task" permette di creare manualmente un task, assegnandogli eventualmente dei canali virtuali globali. Questi ultimi non sono trattati in questo documento, pertanto si rinvia alla documentazione ufficiale della libreria DAQmx per maggiori informazioni. Sebbene la creazione esplicita dei tasks possa essere utile in alcuni casi particolari, in molte applicazioni è possibile evitarla in quanto se ne occupa automaticamente la libreria, come spiegato nei precedenti paragrafi.
I parametri di ingresso del blocco sono i seguenti:

- **task to copy**: permette di specificare il nome di un task da "copiare". In questo caso, il nuovo task erediterà la configurazione di quello specificato, ad esclusione degli eventuali canali virtuali globali aggiuntivi.

- **global virtual channels**: specifica un canale virtuale globale o una lista di canali virtuali globali da assegnare al task.

- **new task name**: specifica il nome da assegnare al nuovo task. Se si utilizza il blocco in un loop e si specifica un nome per il task, è necessario distruggere il task alla fine di ogni iterazione, tramite il blocco "Clear Task". Se ciò non avviene, il sistema tenterebbe di creare più tasks con lo stesso nome e, di conseguenza, verrebbe generato un errore. In realtà, conviene distruggere il task anche quando non si specifica esplicitamente un nome, perchè altrimenti i tasks creati ad ogni iterazione rimarrebbero in memoria, causando inutili sprechi ed inefficienze.

- **auto cleanup**: specifica se distruggere automaticamente il task quando l'applicazione termina la sua esecuzione. Se impostato a FALSE, fa sì che il task venga distrutto solo quando si chiude l'ambiente LabView.

In uscita al blocco è presente il riferimento al task appena creato, "task out".

### 2.2.10 VI "Control Task"

Il blocco "Control Task" è stato già introdotto nel paragrafo 2.1.3, in riferimento alle transizioni di stato dei tasks. In effetti questo blocco permette di forzare una transizione di stato su un task, alterando dunque quella che è la gestione automatica dei tasks da parte della libreria DAQmx.
I parametri di ingresso del blocco sono i seguenti:

- **task/channels in**: specifica il nome del task o una lista di canali virtuali a cui applicare il blocco. Specificando una lista di canali virtuali, viene creato automaticamente un task.

- **action**: specifica l'azione da eseguire sul task; ogni azione comporta la transizione del task in un determinato stato. Le scelte possibili per questo attributo sono "abort", "commit", "reserve", "unreserve" e "verify"; in Fig. 2.1 a pagina 20 si può osservare la corrispondenza tra le azioni e i relativi stati a cui vengono portati i tasks.

In uscita al blocco è presente il riferimento al task corrente "task out".

### 2.2.11 VI "Start Task"

Il blocco "Start Task", già introdotto nel paragrafo 2.1.3, permette di portare manualmente un task nello stato "running", alterando dunque la gestione automatica dei tasks della libreria DAQmx. I parametri di questo blocco sono identici a quelli del blocco "Control Task", ad eccezione dell'ingresso "action" che in questo caso è mancante.

### 2.2.12 VI "Stop Task"

Questo blocco, come suggerisce il nome, blocca l'esecuzione del task, eseguendo dunque l'operazione inversa rispetto al blocco "Start Task". Le operazioni eseguite sul task dopo che è stato bloccato dipendono dallo stato in cui era prima di essere eseguito, come descritto in dettaglio nel paragrafo "Transizioni a stati inferiori" a pagina 24. I parametri di questo blocco sono identici a quelli del blocco "Control Task", ad eccezione dell'ingresso "action" che in questo caso è mancante.

### 2.2.13 VI "Clear Task"

Il blocco "Clear Task" è l'ultimo dei quattro blocchi fondamentali per la manipolazione dei tasks. Con questo blocco è possibile distruggere un task, così da rilasciare tutte le risorse fisiche e virtuali da esso occupate. Oltre all'ingresso "error in" e all'uscita "error out", l'unico altro ingresso del blocco è "task in". A differenza di quanto avviene negli altri VI, in questo caso è possibile collegare a "task in" esclusivamente un task, e non più anche una lista di canali virtuali. Quando un task viene distrutto tramite questo blocco, le operazioni effettuate su di esso variano a seconda dello stato in cui si trova, analogamente a come descritto nel paragrafo "Transizioni a stati inferiori" a pagina 24.

### 2.2.14 I Property Nodes

A seconda del valore che si imposta nel selettore polimorfico di un blocco, i parametri di ingresso e di uscita di quest'ultimo possono, in generale, variare. Ci sono determinati parametri, che risultano opzionali o comunque di secondaria importanza rispetto a quelli esaminati, che non vengono proposti direttamente come terminali del blocco, semplicemente perchè sono troppi ed è impossibile visualizzarli tutti. Per poter impostare questi attributi si ricorre dunque a dei blocchi particolari, detti "property nodes", con i quali è possibile selezionare di volta in volta il parametro da modificare, collegando il valore desiderato. Esistono property nodes per molti dei blocchi esposti precedentemente, ad esempio il "Channel Property Node" per configurare i canali virtuali, o il "Read Property Node" per configurare i blocchi "Read".

I property nodes sono una caratteristica dell'ambiente LabView e prescindono dalla libreria DAQmx; si rimanda alla documentazione fornita dalla National Instruments per ulteriori informazioni.

## 2.3 Esempi basilari di utilizzo della libreria

I concetti fin'ora esposti sono sufficienti a sviluppare un grande numero di applicazioni di misura, sfruttando la libreria DAQmx e un dispositivo compatibile. Qui di seguito vengono riportati alcuni esempi di applicazioni basilari realizzate mediante la libreria e la scheda DAQ 6008; per ogni esempio è presente un'immagine dello schema a blocchi in LabView, oltre ad una spiegazione completa delle operazioni svolte dall'applicazione

### 2.3.1 Acquisizione di un segnale analogico

In questo esempio viene mostrata una possibile realizzazione di un'applicazione LabView che acquisisce una forma d'onda analogica. In figura Fig. 2.3 è mostrato lo schema a blocchi complessivo del VI; nella figura sono stati inseriti dei riquadri colorati e numerati, ognuno contenente un insieme di blocchi LabView, che verranno analizzati separatamente in quanto svolgono funzioni logiche distinte.
Il secondo riquadro, riportato in Fig. 2.5 e indicato con il numero "2" in Fig. 2.3, contiene la logica di acquisizione del segnale analogico. Innanzi tutto viene creato un canale virtuale di tipo "AI Voltage", che viene configurato tramite degli appositi controlli nel pannello frontale del VI, e che nello schema a blocchi sono "Configurazione dei terminali di ingresso", "Minimo valore del segnale", "Massimo valore del segnale". In questo esempio non è possibile specificare, dal pannello frontale, i terminali a cui viene collegata la sorgente di segnale; le linee da usare sono specificate invece tramite una costante "DAQmx Physical Channel". Un altro aspetto da notare è che il canale virtuale ha il terminale "task in" scollegato. Di conseguenza, la libreria DAQmx crea, al momento dell'esecuzione del VI, un nuovo task a cui il canale virtuale viene automaticamente assegnato. Il blocco successivo ha la funzione di impostare le regole di temporizzazione del task. In particolare vengono acquisiti un numero finito di campioni, specificato tramite il controllo "Numero di campioni da acquisire", alla frequenza di campionamento impostata dall'utente. Importante, in questo caso, è impostare su "Rising" l'ingresso del blocco "Timing" relativo al fronte del clock su cui avviene il campionamento, perchè il dispositivo DAQ 6008 non supporta l'opzione "Falling". Il blocco successivo è di tipo "Read", con il selettore polimorfico impostato su "Analog Wfm 1Chan NSamp" in quanto bisogna acquisire un segnale analogico da un unico canale. Il parametro "number of samples per channel" è stato impostato a "-1", perchè così vengono acquisiti tutti i campioni disponibili nel buffer quando l'operazione di campionamento termina.

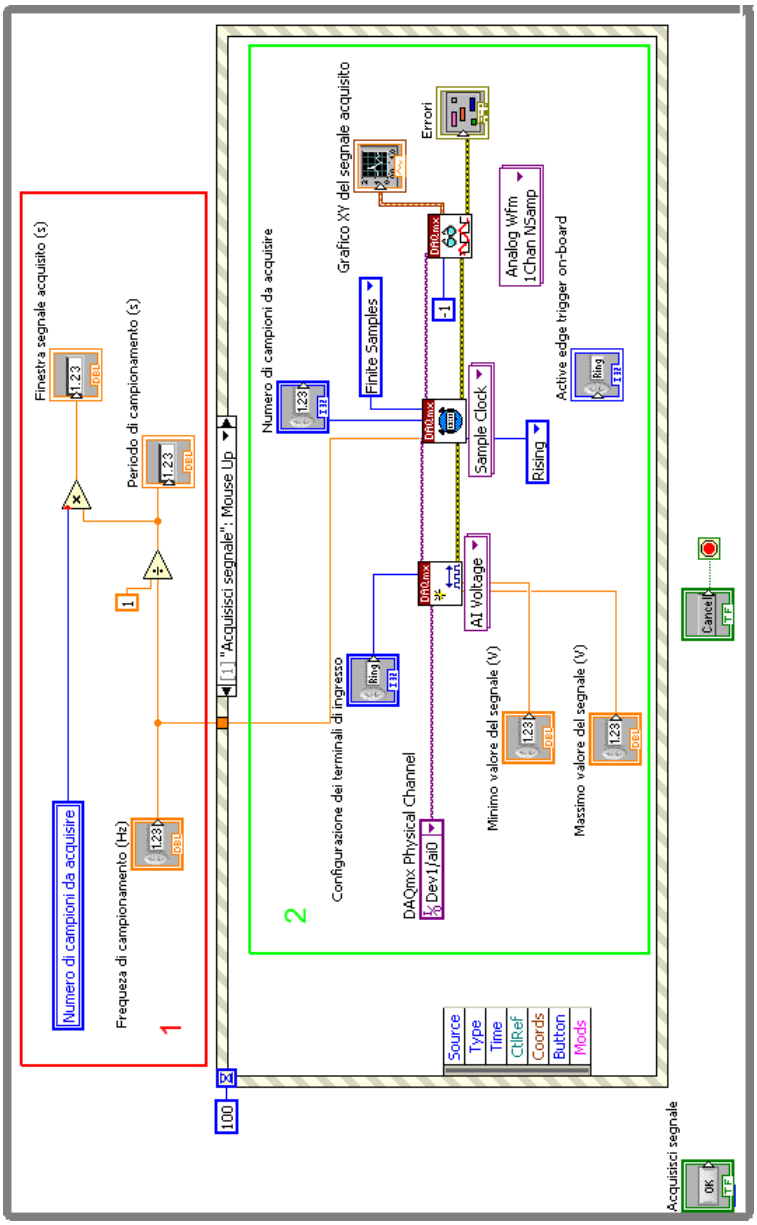



**Fig. 2.3:** Schema a blocchi del VI

Come si nota, manca l'ingresso "timeout", che assume dunque il valore di default di 10 secondi. Se si imposta dal pannello frontale un numero di campioni da acquisire pari a 2000 ed una frequenza di campionamento di 100Hz, la durata dell'acquisizione diventa di 20 secondi, un valore superiore al timeout di default. Con simili impostazioni, di conseguenza, viene generato un errore in fase di esecuzione da parte del blocco "Read", che non riesce ad acquisire i campioni necessari prima dello scadere del timeout. Questo

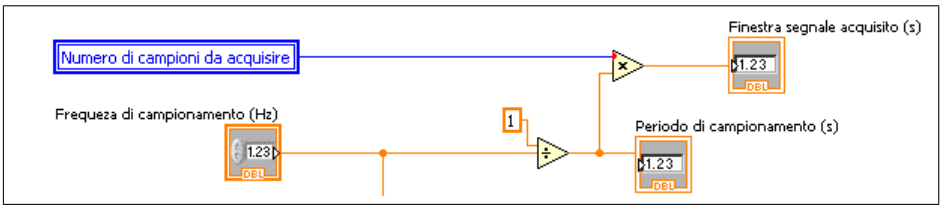



**Fig. 2.4:** Ingrandimento del riquadro "1" di Fig. 2.3

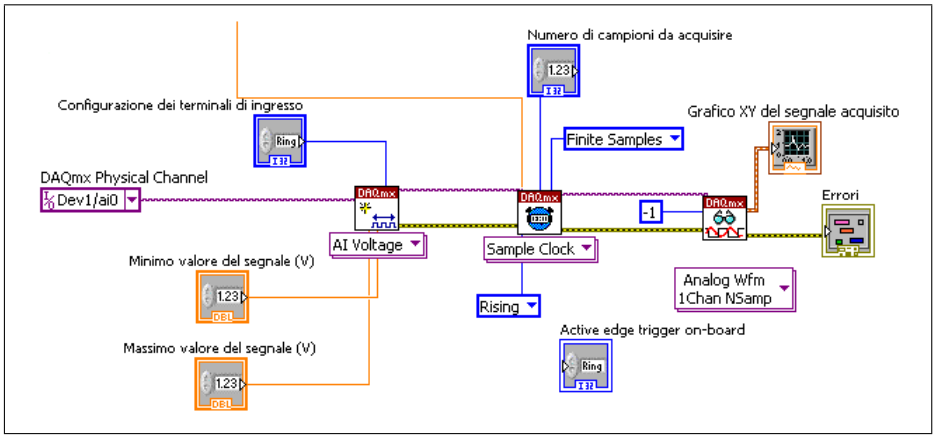



**Fig. 2.5:** Ingrandimento del riquadro "2" di Fig. 2.3

problema, ovviamente, è di semplice soluzione; è sufficiente infatti impostare manualmente l'attributo "timeout" al valore "-1", così da obbligare il blocco "Read" ad attendere indefinitamente senza generare alcun errore. Il segnale analogico acquisito viene infine visualizzato sfruttando il blocco "Waveform Graph" di LabView.
Tutta la logica di acquisizione del segnale appena descritta, come si osserva dalla Fig. 2.3, è contenuta all'interno di un blocco "Event" di LabView, sensibile alla pressione del pulsante "Acquisisci segnale". L'intera applicazione, invece, è contenuta all'interno di un ciclo "while", che termina solo alla pressione del pulsante "Termina VI" del pannello frontale. Grazie a questa configurazione, è possibile eseguire più acquisizioni senza dover eseguire il VI ogni volta.
Sebbene l'applicazione possa teoricamente acquisire forme d'onda analogiche di qualunque frequenza, bisogna ricordare che il dispositivo DAQ 6008 ha un sampling-rate massimo pari a 10kS/s. Di conseguenza, per il teorema di Nyquist-Shannon, è possibile campionare esclusivamente segnali di frequenza inferiore a 5kHz, pena la presenza di effetti di aliasing in frequenza. La massima frequenza dei segnali da acquisire diminuisce ulteriormente se si vuole ottenere una buona ricostruzione del loro andamento nel dominio del tempo senza la necessità di dover applicare algoritmi di interpolazione

dei campioni acquisiti.
In Fig. 2.6 è riportato il pannello frontale dell'esempio appena descritto; in questo caso è stato acquisito un segnale analogico con andamento sinusoidale, avente ampiezza picco-picco di 2V e frequenza di 10Hz.

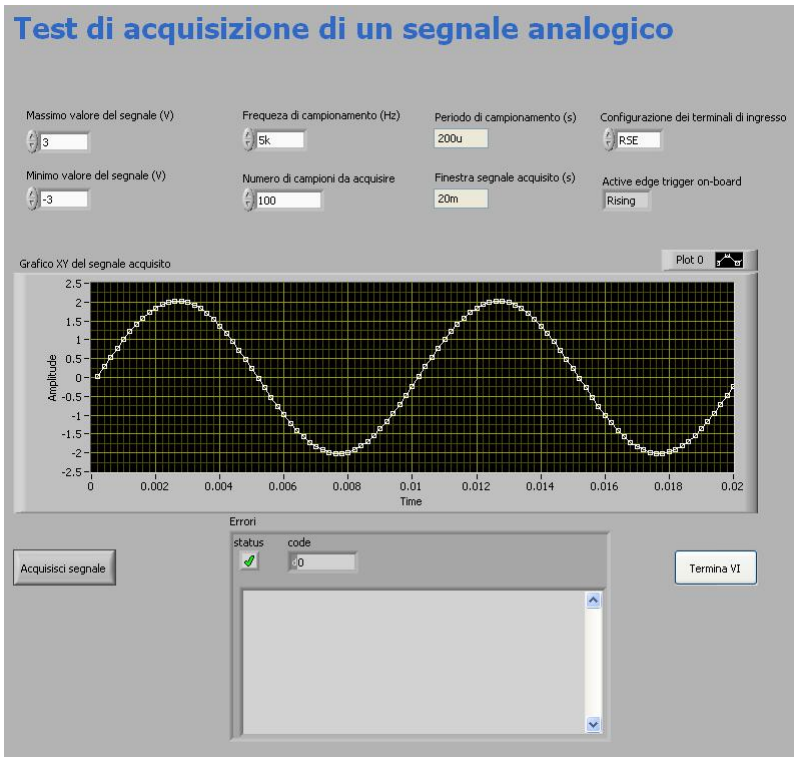



**Fig. 2.6:** Pannello frontale dell'applicazione, che mostra l'acquisizione di un segnale analogico sinusoidale

### 2.3.2 Acquisizione di un segnale analogico con trigger digitale

Quando è stato descritto il blocco "Trigger" della libreria DAQmx, si è visto come il dispositivo DAQ 6008 sia molto limitato per quanto riguarda le funzionalità di triggering. L'unica funzione che è possibile realizzare, infatti, è quella di un trigger digitale da utilizzare al fine di determinare l'istante in cui inizia l'acquisizione di campioni analogici. In altre parole, il sistema rimane in attesa fino a quando non si verifica un evento che fa "scattare" il trigger, e solo in quel momento il dispositivo inizia ad acquisire i campioni. La figura Fig. 2.7 mostra lo schema a blocchi dell'applicazione in esame. Per semplicità è stato riutilizzato il codice dell'esempio precedente, aggiungendo esclusivamente il blocco "Trigger", necessario per implementare il trigger digitale. Il funzionamento dell'applicazione è praticamente invariato rispetto

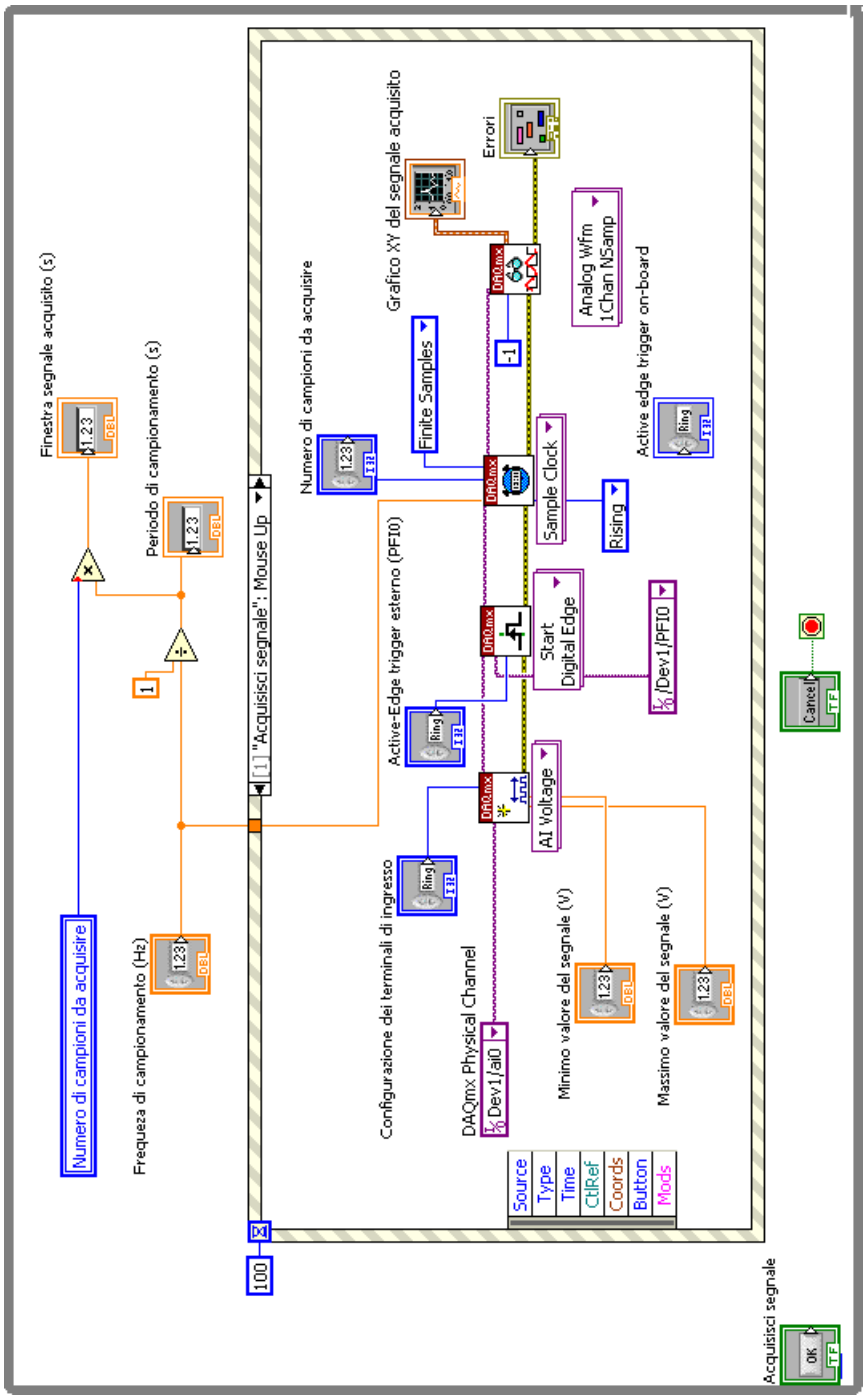



**Fig. 2.7:** Schema a blocchi del VI

all'esempio precedente; l'unica differenza è che, questa volta, l'acquisizione del segnale analogico inizia solo quando si verifica l'evento atteso dal blocco

"Trigger", di cui è riportato un ingrandimento in figura Fig. 2.8. Tramite

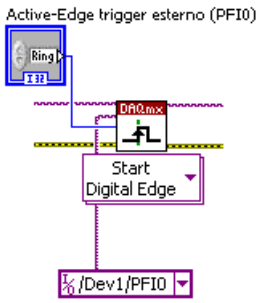



**Fig. 2.8:** Ingrandimento del blocco "Trigger"

il pannello frontale è possibile, mediante uno switch, impostare l'attributo "edge" del blocco, che per quanto detto sul dispositivo DAQ 6008, deve avere il selettore polimorfico impostato su "Start Digital Edge". Non è invece possibile scegliere il terminale fisico al quale collegare il segnale di trigger, che deve necessariamente essere il pin PFI0 e, pertanto, è stato impostato mediante una costante LabView.
Anche in questo caso, valgono tutte le considerazioni, fatte nell'esempio precedente, circa la massima frequenza del segnale da acquisire al fine di evitare fenomeni di aliasing. Per quanto riguarda l'output nel pannello frontale del VI, si faccia riferimento a quello di Fig. 2.6, che è del tutto identico a meno dello switch di selezione dell'attributo "edge" del trigger.

### 2.3.3 Lettura di N linee digitali

Questo esempio mostra come realizzare una semplice applicazione in grado di monitorare il livello logico di un numero variabile di linee digitali. Nell'esempio ci si limita a visualizzare lo stato delle linee con dei led, ma in un'applicazione reale è possibile inserire una logica aggiuntiva che elabori le informazioni sugli ingressi digitali. Lo schema a blocchi dell'applicazione è mostrato in Fig. 2.9. Anche in questo esempio è stata scelta una struttura ad eventi, contenuta a sua volta in un ciclo "while", così da poter eseguire più misure senza dover riavviare ogni volta il VI. Come si osserva dall'immagine, lo schema a blocchi di questo esempio è davvero semplice. Innanzi tutto viene creato un canale virtuale di tipo "Digital Input", in quanto bisogna leggere delle linee digitali. Le linee digitali che si intende monitorare vengono specificate, nel pannello frontale, tramite un apposito selettore; poichè in generale vengono lette più linee, è possibile impostare un range di linee digitali, o eventualmente anche un'intera porta digitale. Il blocco seguente è di tipo "Read", con il selettore polimorfico indicante "Digital 1D Bool 1Chan 1Samp", in modo da leggere da un singolo canale un unico campione

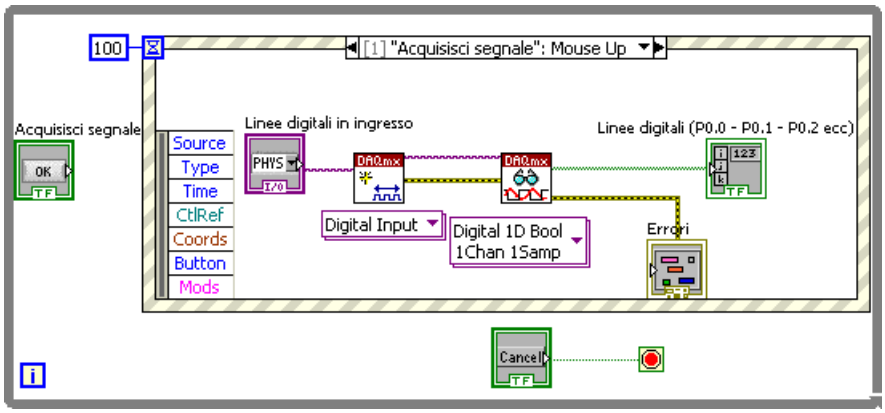



**Fig. 2.9:** Schema a blocchi del VI

digitale, formattato come array di booleani. Con il dispositivo DAQ 6008 non è possibile, come già visto, impostare alcuna proprietà di timing nel caso di lettura di campioni digitali, che avviene dunque in modalità "on demand", cioè appena il task viene eseguito. Inoltre, sempre a causa delle limitazioni fisiche della scheda, non è realizzabile neanche il trigger digitale visto nell'esempio precedente, che può essere implementato solo per la lettura di campioni analogici. Le informazioni lette vengono infine visualizzate nel pannello frontale mediante degli indicatori di tipo "led".
In figura Fig. 2.10 è mostrato un esempio di monitoraggio di quattro linee digitali, in cui è possibile notare la notazione "a range" utilizzata per specificare quali linee osservare.

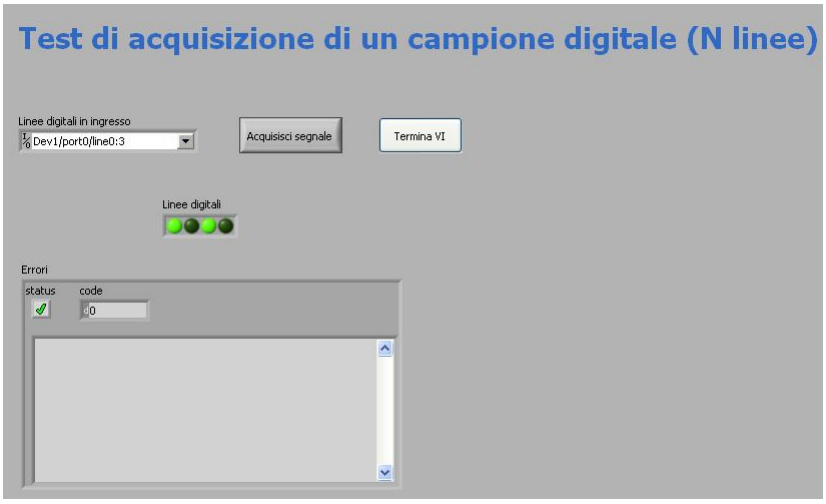



**Fig. 2.10:** Pannello frontale del VI, mentre vengono monitorate quattro linee digitali

### 2.3.4 Lettura di un byte da una porta digitale

Nell'esempio precedenze è stato visto come sia possibile monitorare lo stato di una o più linee digitali della scheda DAQ 6008. Nella pratica un sistema di quel genere può essere utile per monitorare lo stato di vari sensori di un sistema di automazione, nel quale ogni linea svolge una funzione logica indipendente da tutte le altre. Esiste però la possibilità che le varie linee facciano parte di uno stesso bus dati, con lo scopo di trasmettere dati digitali che richiedono più di un bit per essere memorizzati. La scheda DAQ 6008, come già visto quando si è discusso dell'hardware del dispositivo, è fornita di 12 ingressi/uscite digitali raggruppate in due porte, ovvero P0 che contiene 8 linee e P1 che ne contiene 4. Le operazioni di lettura e scrittura possono avvenire su una sola linea o su un gruppo arbitrario di linee, come visto nell'esempio precedente. Un'ulteriore opzione, infine, consente di leggere o scrivere un'intera porta, così da non doversi preoccupare delle singole linee. In questo caso il dato che viene letto non sarà restituito sottoforma di un array di booleani, bensì come intero senza segno, in quanto non si è interessati alle singole linee, ma all'informazione complessiva che esse trasportano. In figura Fig. 2.11 è mostrato lo schema a blocchi di un'applicazione che legge un byte, cioè un intero a 8 bit, dalla porta P0 del dispositivo, visualizzando il dato nel pannello frontale mediante un indicatore numerico. Come si può

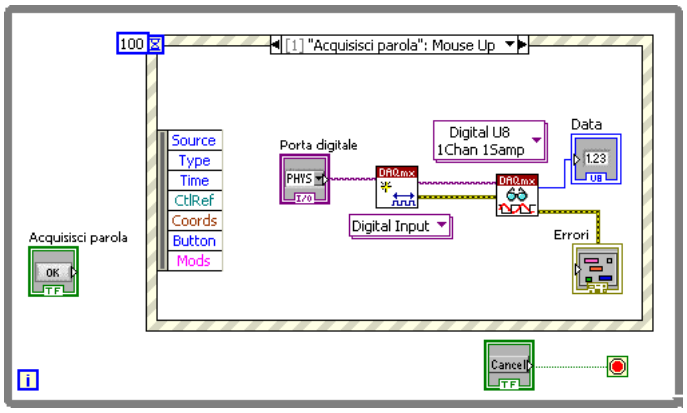



**Fig. 2.11:** Schema a blocchi del VI

notare lo schema a blocchi è molto simile a quello dell'esempio precedente, tranne per il fatto che il blocco "Read" ha il selettore polimorfico impostato su "Digital U8 1Chan 1Samp" e, in uscita, restituisce un intero piuttosto che un array di booleani. Inoltre nel pannello frontale è necessario inserire il nome di una porta digitale del dispositivo, e non più dunque un range di linee digitali su cui leggere i dati. In realtà, usando il DAQ 6008, la scelta della porta è obbligata, in quanto solo la porta P0 possiede le 8 linee necessarie all'applicazione, mentre la P1 ne possiede solo 4. In Fig. 2.12 infine è

mostrato il pannello frontale dell'applicazione, quando tutte le linee digitali della porta P0 sono allo stato logico basso.

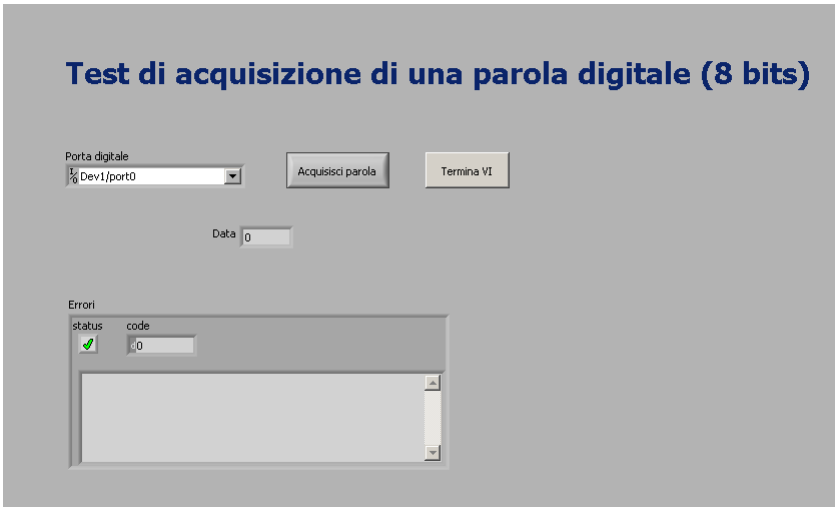



**Fig. 2.12:** Pannello frontale del VI

### 2.3.5 Scrittura di N linee digitali

Questo esempio mostra come realizzare una sempice applicazione che scrive una parola digitale di N bits su altrettante linee digitali della scheda DAQ 6008. Lo schema a blocchi del programma è rappresentato in Fig. 2.13. Come al solito è stata utilizzata una struttura ad eventi, racchiusa in un

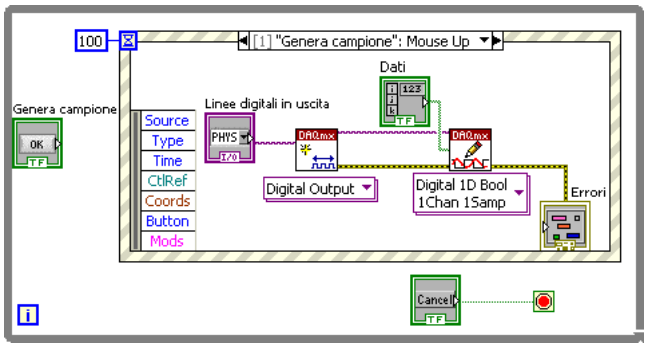



**Fig. 2.13:** Schema a blocchi del VI

ciclo "while", per permettere di generare più campioni senza la necessità di riavviare il VI. All'interno della struttura ad eventi è presente in primo luogo un blocco "Create Virtual Channel", impostato su "Digital Output", in quanto è necessario scrivere delle linee digitali, specificate nel controllo

"Linee digitali in uscita". Di seguito è presente un blocco "Write", con selettore polimorfico impostato su "Digital 1D Bool 1Chan 1Samp", che come già visto permette di scrivere un campione digitale composto da N bits su un canale virtuale. La parola da scrivere viene impostata tramite il pannello frontale, mediante degli interruttori a led. In Fig. 2.14 è rappresentato


**Fig. 2.14:** Pannello frontale del VI

il pannello frontale del VI, mentre viene generata la parola digitale "1010", mentre la Fig. 2.15 riporta l'andamento nel tempo delle linee digitali coinvolte, ottenuto mediante un oscilloscopio digitale. Nella Fig. 2.15 è possibile osservare un limite fisico del dispositivo, dovuto al fatto che le linee digitali vengono pilotate in sequenza e non in parallelo: le linee D0 e D2 non vengono portate allo stato logico alto contemporaneamente, ma c'è un ritardo. Si è verificato sperimentalmente che questo ritardo aumenta all'aumentare del numero di linee che cambiano stato, e che quindi è necessario tenerne conto in un eventuale sistema di misura che riceve in ingresso le linee digitali della scheda DAQ 6008.

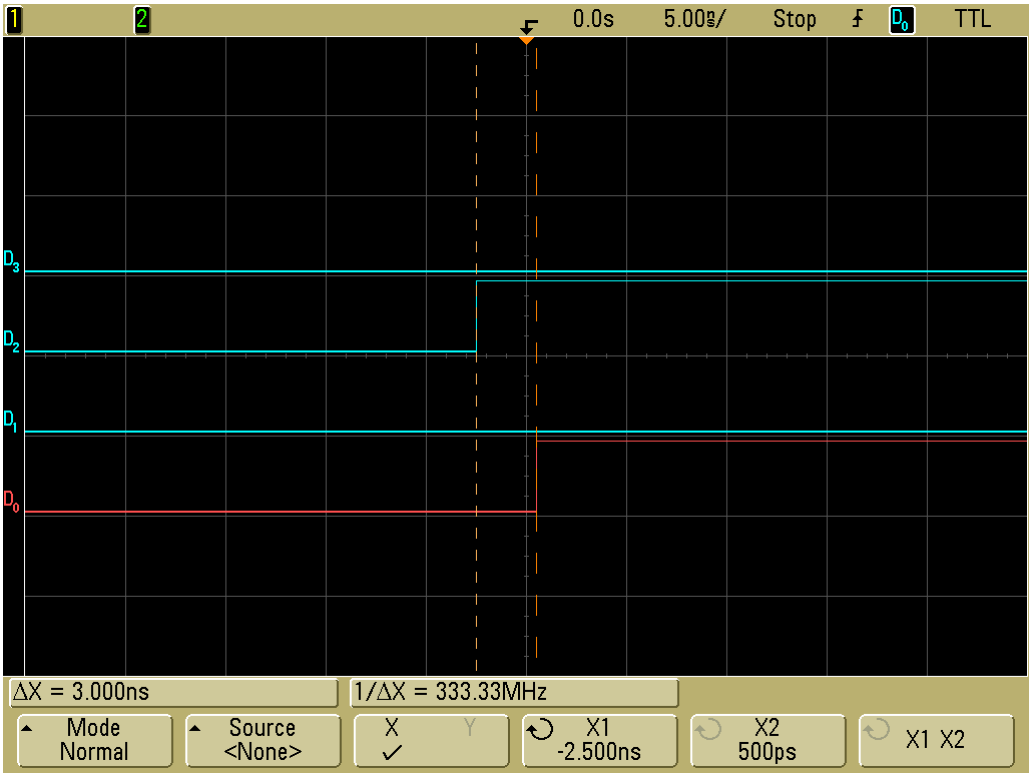



**Fig. 2.15:** Andamento nel tempo delle linee digitali scritte, ottenuto mediante un oscilloscopio digitale. Si nota in particolare il ritardo di 3ns tra le transizioni a livello logico alto delle linee D0 e D2.

# Capitolo 3

# Esempi di applicazioni con la scheda NI DAQ-6008

Nel capitolo precedente sono state descritte le principali funzionalità della libreria DAQmx, con particolare attenzione alle funzioni realizzabili con il dispositivo DAQ 6008. Verranno ora illustrati due esempi di applicazioni realizzate mediante la libreria DAQmx in ambiente LabView e la scheda DAQ 6008, con lo scopo di mostrare le potenzialità e la versatilità di questo dispositivo.

## 3.1 Realizzazione di una forma d'onda digitale

Si supponga di avere un sistema elettronico che riceve in ingresso forme d'onda digitali, per poi eseguire delle particolari elaborazioni sui campioni. Un esempio di tale sistema può essere un semplice convertitore digitale-analogico, che riceve in ingresso delle parole binarie e le converte in un segnale analogico. Per poter verificare il funzionamento di questo semplice dispositivo, è chiaro che bisogna disporre di un qualche strumento in grado di generare segnali digitali di test.
Utilizzando la scheda DAQ 6008 è possibile generare i campioni digitali richiesti, nella condizione di dover testare un DAC con frequenza di campionamento compatibile con quella del DAQ a disposizione. Il programma realizzato per questo esempio, infatti, permette di generare una forma d'onda analogica e successivamente di quantizzarla, il tutto via software; i campioni ottenuti dal processo di quantizzazione, infine, vengono generati fisicamente sfruttando le linee di input/output digitali della scheda DAQ.
La figura Fig. 3.1 mostra lo schema a blocchi del VI dell'applicazione. Come si può notare osservando la figura, il programma si compone essenzialmente di tre blocchi logici fondamentali. Inizialmente viene generata una forma d'onda sinusoidale a partire dai parametri "Frequenza", "Ampiezza" e "Fase" specificati nel pannello frontale. L'ampiezza è collegata anche al terminale

"Offset" del blocco, in modo da ottenere un'onda sempre positiva, in quanto il convertitore analogico-digitale da simulare è di tipo unipolare. I "Parametri di campionamento", specificati nel pannello frontale, indicano quanti campioni acquisire e a quale frequenza, simulando un circuito di campionamento sample/hold. Una volta che i campioni analogici sono stati acquisiti e memorizzati, è necessario simulare il processo di quantizzazione del convertitore. Questo avviene all'interno del ciclo "for" mostrato nello schema a blocchi, in cui vengono eseguite le seguenti operazioni:

1. Si controlla che il valore del campione appartenga al range [0, $V_{SAT}$]V. $V_{SAT}$ indica la tensione di saturazione del convertitore ADC, e vale $V_{SAT} = V_{FS} - \Delta Q$, dove $V_{FS}$ è la tensione di fondo scala dell'ADC, mentre $\Delta Q$ indica il passo di quantizzazione. La tensione di fondo scala può essere impostata direttamente tramite il pannello frontale del VI, mentre il passo di quantizzazione è una grandezza derivata a partire dalla tensione di fondo scala e dal numero N di bits dell'ADC che si vuole simulare, e vale $\Delta Q = \frac{V_{FS}}{2^N}$. Se il campione analogico fuoriesce dall'intervallo ammesso, viene forzato al valore della soglia più vicina; ad esempio se $V_{SAT} = 3.75V$, un campione di $4V$ viene forzato a $3.75V$.

2. Si ricava il codice digitale $q_k$ che rappresenta il campione analogico $a_k$ mediante la formula $q_k = round\left(\frac{a_k}{\Delta Q}\right)$, dove $round(x)$ è una funzione che arrotonda $x$ all'intero più vicino. Il codice intero $q_k$ così ottenuto viene poi convertito in binario, sottoforma di un array di booleani, e infine inviato in uscita al ciclo.

A questo punto il processo di generazione di una forma d'onda analogica e relativa quantizzazione è terminato e, quindi, è possibile procedere alla generazione della forma d'onda digitale. Questo processo avviene all'interno del ciclo "while" mostrato nello schema a blocchi del VI, che riceve in ingresso l'array di parole binarie che corrispondono ai codici digitali che rappresentano i campioni analogici. Per quanto detto riguardo alle transizioni di stato implicite dei tasks, sono stati inseriti i blocchi "Start Task" e "Stop Task" fuori dal ciclo "while" per aumentare l'efficienza dell'applicazione. Infine, è necessaria una osservazione riguardo all'uso del blocco LabView "Wait". Nel VI, infatti, tale funzionalità è stata utilizzata per permettere all'utente di impostare la durata temporale dei singoli campioni digitali. Tale soluzione, però, si è rivelata poco affidabile in quanto la durata degli intervalli risulta variare a seconda del calcolatore utilizzato. Cosa ancora peggiore, risulta variare anche a seconda del numero di processi che il sistema operativo sta gestendo mentre il VI viene eseguito, in modo del tutto imprevedibile. Malgrado ciò, dal punto di vista teorico il programma funziona correttamente, pertanto è stato deciso di lasciare inalterata la struttura a blocchi. Sarebbe

comunque interessante osservare come si comporterebbe l'applicazione qualora venga eseguita in un sistema real time.
La Fig. 3.2 mostra il pannello frontale del programma mentre vengono generate parole digitali a 5 bits, mentre in Fig. 3.3 è riportato l'andamento delle linee digitali della scheda DAQ 6008 con parole a 4 bits. In quest'ultima figura viene anche evidenziato il periodo dell'onda sinusoidale digitale generata, che risulta di 16ms. Questo valore è stato ottenuto impostando il parametro "rate", collegato al blocco "Wait" di LabView, a 0ms, e quindi rappresenta il periodo minimo ottenibile. Non è chiaro, comunque, se si tratta di un limite fisico del dispositivo DAQ 6008 o, piuttosto, di una limitata velocità di esecuzione del VI da parte del calcolatore.

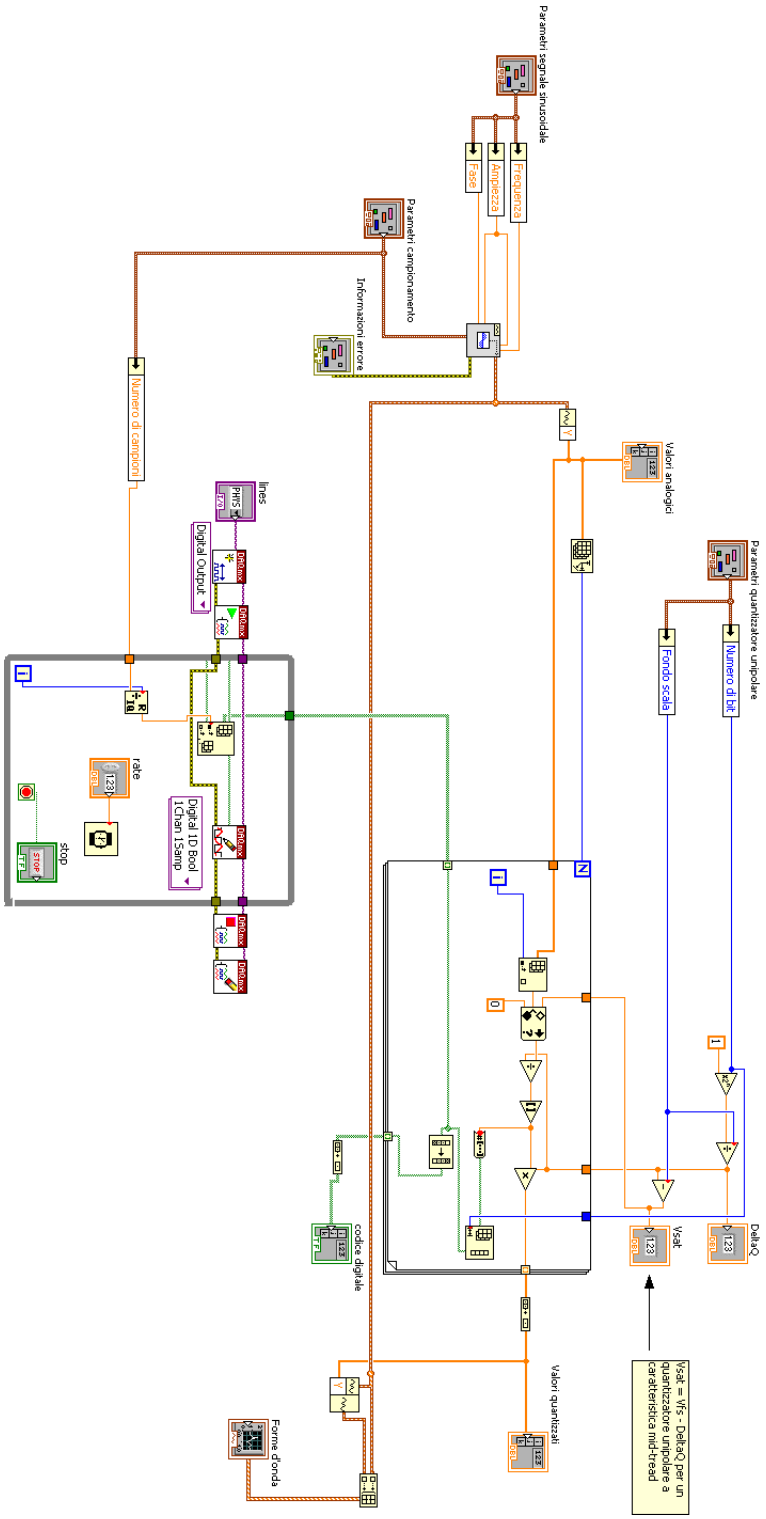



**Fig. 3.1:** Schema a blocchi dell'applicazione

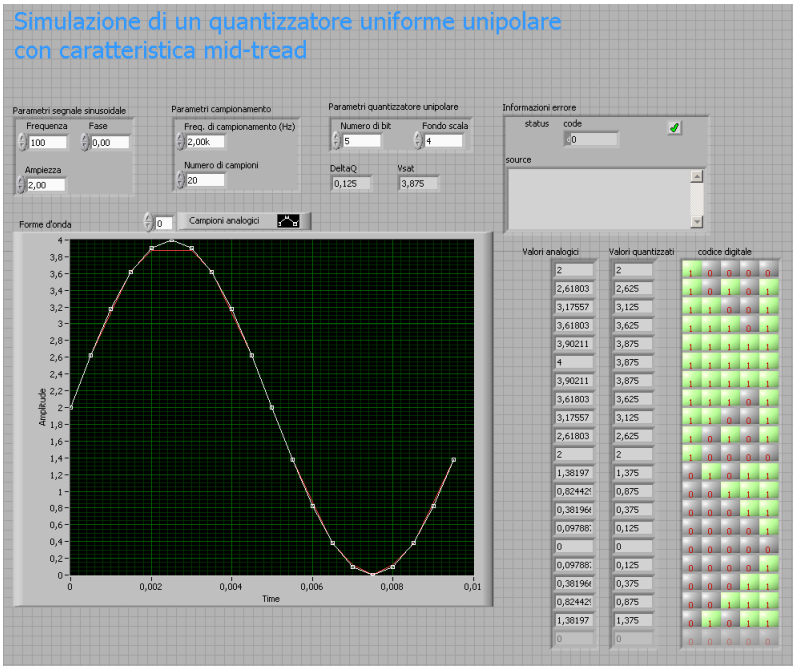



**Fig. 3.2:** Pannello frontale dell'applicazione

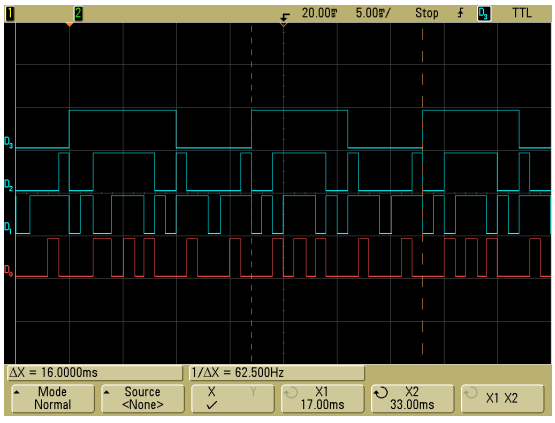



**Fig. 3.3:** Andamento temporale delle uscite digitali della scheda DAQ 6008

## 3.2 Realizzazione di un sistema di automazione

Questo esempio di utilizzo della scheda DAQ 6008 rappresenta in realtà l'obiettivo finale della tesi, il cui scopo era proprio di sviluppare un sistema di automazione controllato in ambiente LabView. Il programma che verrà esposto di seguito permette all'utente di controllare dei motori passo-passo unipolari, generando degli opportuni segnali digitali, in due modalità differenti: "wavemode" (o full-step) e "two phases on". La differenza tra i tipi di alimentazione è che con la prima viene alimentata una sola fase alla volta, mentre nel secondo caso vengono alimentate due fasi contemporaneamente, ottenendo quindi una coppia maggiore. I segnali digitali generati dalla scheda DAQ, ovviamente, non pilotano direttamente i motori, ma controllano un opportuno stadio di potenza, che si occupa infine di fornire energia alle fasi dei motori.

### 3.2.1 Il pannello frontale

Il pannello frontale del *virtual instrument* che controlla la rotazione dei motori è mostrato in Fig. 3.4. Nella figura è stato riportata l'interfaccia che controlla l'azionamento di un solo motore. In realtà il pannello frontale è costituito da due interfacce identiche a quella della Fig. 3.4 perchè deve gestire la rotazione di due motori. Infatti la piattaforma da azionare può sia traslare sia ruotare su sè stessa, per cui necessita di due motori che siano azionati in modo indipendente l'uno dall'altro.
Nella tabella 3.1 sono stati elencati i parametri che l'utente deve inserire tramite il pannello frontale. Una volta impostati tutti i parametri necessari, premendo il pulsante "Start Rotazione" il programma comincia ad elaborare la sequenza di alimentazione del motore. Se alcuni valori inseriti dall'utente risultano errati(ad esempio se il numero di step è impostato a 0), allora l'elaborazione della sequenza di controllo si interrompe e mostra l'errore in un'apposita finestra presente sul pannello frontale; in tali casi l'esecuzione del VI non viene interrotta ma rimane in attesa fino ad una nuova pressione del pulsante di start. Se i parametri inseriti sono corretti, allora il VI comincia a scrivere la sequenza di alimentazione delle fasi sui pin della scheda DAQ. Ogni volta che viene impostata una nuova sequenza, il programma aggiorna un indicatore che mostra all'utente le fasi attive.
Sul pannello frontale è stato introdotto anche un pulsante *Stop VI* che termina l'esecuzione del VI. In pratica, quando viene premuto il pulsante di stop, l'esecuzione del VI viene interrotta; per riavviare l'esecuzione del programma bisogna selezionare il comando di *run* integrato nell'ambiente grafico di LabView. Si noti che il programma è stato realizzato in modo da terminare l'esecuzione dopo aver compiuto la rotazione del motore: se per esempio l'utente preme il pulsante di start e subito dopo termina l'esecuzione del VI, il programma prima esegue il comando di *start rotazione* facendo compiere

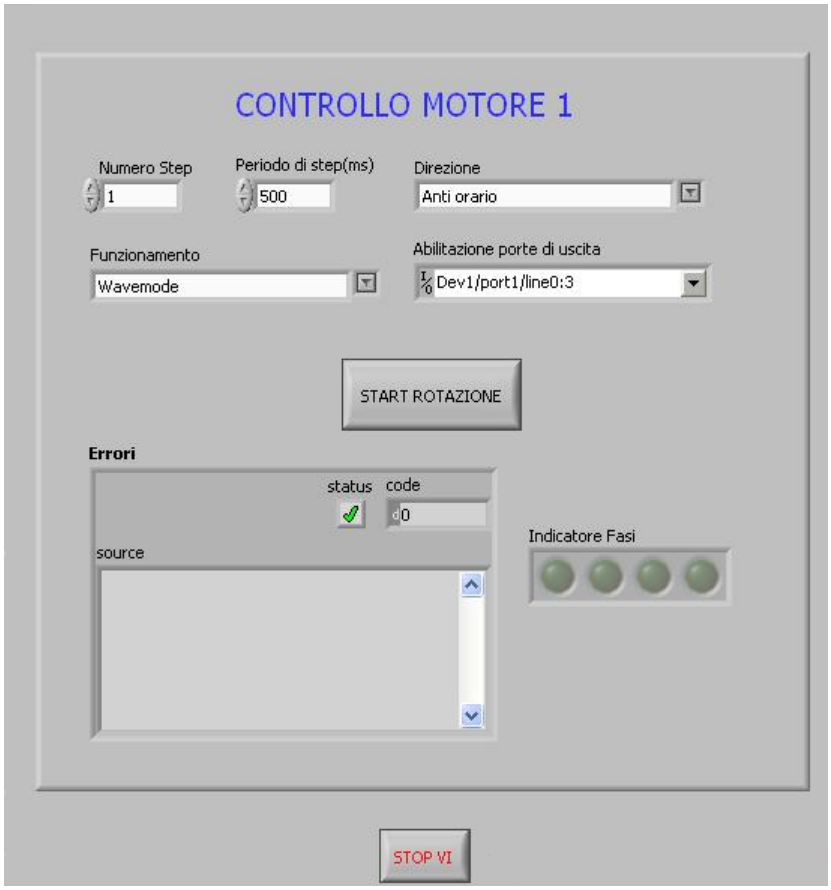



**Fig. 3.4:** Pannello frontale

| Nome del controllo | Descrizione |
|---|---|
| Numero di step | Mediante questo controllo l'utente può inserire il numero di step da far eseguire al motore. Si noti che per *step* si intende l'attivazione di una sola fase del motore. Questo significa che se si vuole eseguire un ciclo completo delle sequenze *wavemode* o *two-phases on* bisogna impostare un valore pari a 4. |
| Periodo di step | Questo comando consente di impostare il tempo che intercorre tra l'accensione di una fase e la successiva. Il valore da specificare è espresso in ms. |
| Direzione di rotazione | Attraverso questo menu a tendina è possibile impostare il verso di rotazione del motore(orario o antiorario). |
| Funzionamento | Questo menu permette di impostare la sequenza di alimentazione delle fasi. Il programma è in grado di generare le sequenze di funzionamento *wavemode* o *two-phases on*. |
| Abilitazione porte di uscita | Questo menu specifica alla scheda DAQ i pin sui quali bisogna generare la sequenza di abilitazione delle fasi. Per il motore 1 sono state impostate di *default* le uscite digitali 0-3 sulla porta 0, mentre per il motore 2 sono state selezionati i pin 0-3 sulla porta 1. |

**Tabella 3.1**

al motore gli *step* specificati dal pannello frontale e poi termina l'esecuzione del VI. Se così non fosse, ogni volta che viene abortita l'esecuzione del programma, i pin di uscita della scheda rimarrebbero impostati sull'ultima sequenza elaborata e quindi una delle fasi del motore rimarrebbe attiva per un tempo indefinito causando il surriscaldamento degli avvolgimenti interni del motore.

### 3.2.2 Struttura interna del VI

In questo paragrafo viene illustrata la struttura e il funzionamento del programma che genera le sequenze di alimentazione delle fasi a partire dai parametri inseriti dall'utente nel pannello frontale. Innanizitutto bisogna

specificare che l'analisi del programma verrà limitata alla parte che aziona il motore 1, in quanto quella che riguarda il motore 2 è del tutto identica.

Nella Fig. 3.7 viene mostrato il diagramma a blocchi del *virtual instrument*. Innanzitutto si può notare che quando inizia l'esecuzione del VI, viene avviato un ciclo *while* la cui condizione di terminazione è collegata al pulsante *Stop VI* del pannello frontale. All'interno del ciclo è presente una struttura ad eventi sensibile alla pressione dei pulsanti indicati nel pannello frontale con il nome di *start rotazione*: dopo la pressione del pulsante l'esecuzione si sposta all'interno della struttura. Il flusso di esecuzione all'interno della struttura ad eventi è mostrato nello schema di Fig. 3.5.

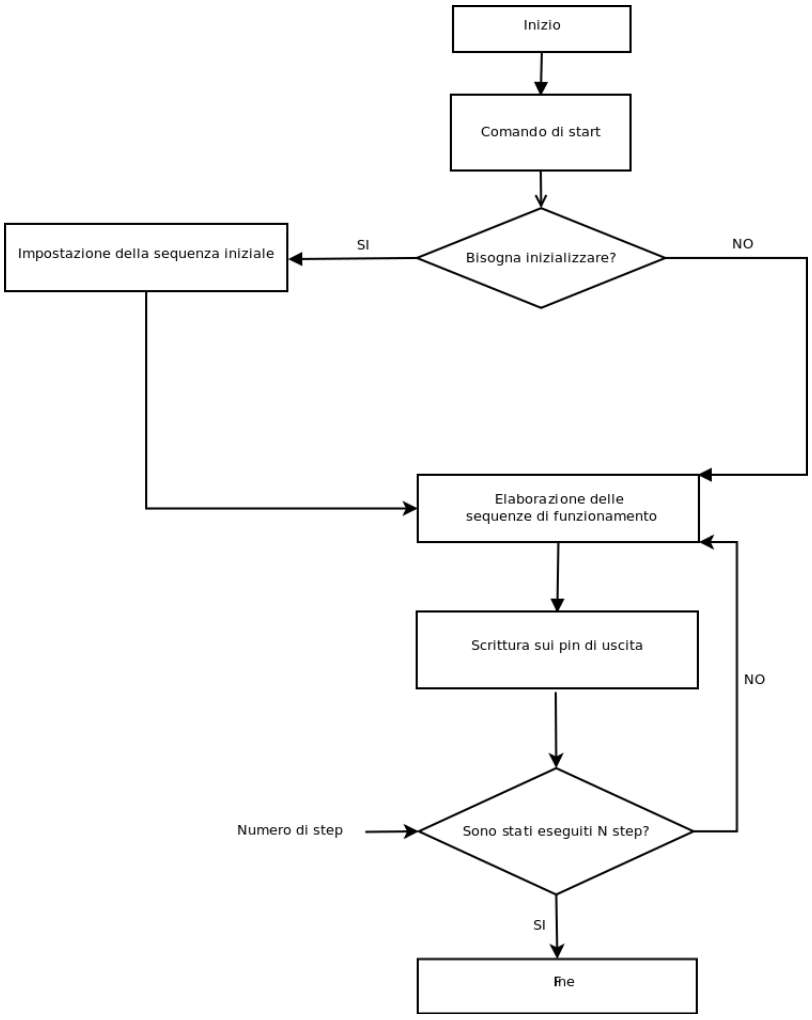



**Fig. 3.5:** Diagramma di flusso

Ogni volta che viene premuto il pulsante di start, il blocco riportato in

Fig. 3.7 ed evidenziato con il riquadro di colore rosso verifica lo stato di inizializzazione del programma. In particolare, si tratta di un breve listato scritto in linguaggio MathScript che verifica se è la prima volta che si aziona uno dei due motori oppure se l'utente ha modificato la modalità di funzionamento del motore. Se si verifica una di queste condizioni, allora il driver, che scrive i dati sui pin di uscita della scheda, ha bisogno di una sequenza iniziale a partire dalla quale vengono elaborate tutte le sequenze che consentono gli *step* successivi al primo.
Se è necessaria l'inizializzazione delle porte di uscita, il *multiplexer* incluso nel riquadro giallo trasferisce in uscita i valori provenienti dal riquadro blu. Questo blocco è indispensabile perchè il VI prevede due modalità di funzionamento (wavemode o two-phases on) che richiedono due diverse sequenze di inizializzazione. Se, invece, il pulsante di start è stato già premuto in precedenza, il VI elabora la sequenza di alimentazione delle fasi a partire dall'ultima sequenza scritta sui pin di uscita della scheda. Il blocco che controlla l'inizializzazione del programma elabora quindi un diverso segnale di selezione in modo tale che il *multiplexer* del riquadro giallo trasferisca in uscita il valore proveniente dalla retroazione che unisce le uscite del blocco di scrittura con gli ingressi del selettore.
Per comprendere a fondo l'importanza del blocco di controllo dell'inizializzazione, si supponga che l'utente voglia effettuare un solo step in modalità wavemode ad ogni pressione del pulsante di start. Quando viene eseguito il primo step, la sequenza scritta sulle uscite della scheda DAQ è la seguente

| Fase 1 | Fase 2 | Fase 3 | Fase 4 |
|---|---|---|---|
| ON | OFF | OFF | OFF |

A questo punto se viene premuto nuovamente il pulsante di start, il programma elabora la nuova sequenza a partire dalla precedente ottenendo

| Fase 1 | Fase 2 | Fase 3 | Fase 4 |
|---|---|---|---|
| OFF | ON | OFF | OFF |

Se non ci fosse il blocco che verifica l'inizializzazione del programma, ogni volta che viene premuto il pulsante di start verrebbe ripetuta la sequenza riportata nella prima tabella causando un malfunzionamento del VI.

Il riquadro verde racchiude il blocco che aggiorna le uscite della scheda DAQ a partire dai valori specificati dai valori precedenti. In pratica si tratta di un *SubVI* che riceve in ingresso la sequenza iniziale oltre ad alcuni parametri specificati dall'utente mediante il pannello frontale. Lo schema degli ingressi e uscite del SubVI è illustrato in Fig. 3.6, mentre il diagramma a blocchi è mostrato nella Fig. 3.8.
La scrittura dei valori sui pin della scheda DAQ avviene mediante la creazione di un canale virtuale che riceve in ingresso l'indicazione delle porte

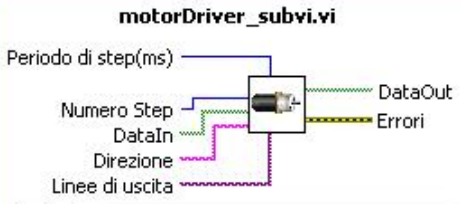



**Fig. 3.6:** Schema degli ingressi e delle uscite

di uscita da aggiornare. Una volta creato il canale, l'esecuzione si sposta all'interno di un ciclo *for* che esegue un numero di iterazioni pari al numero di step specificato dall'utente. All'interno del ciclo sono presenti tre blocchi fondamentali: uno *shifter* (indicato nella Fig. 3.8 col riquadro rosso), un blocco che aggiorna le uscite della scheda DAQ (riquadro giallo) e il *timing* su ciascuna iterazione (riquadro verde). Il primo serve ad elaborare una nuova sequenza a partire da quella in ingresso; siccome la sequenza di alimentazione delle fasi è rappresentata da un array di variabili booleane, ad ogni iterazione il blocco *shifter* ruota il vettore di una posizione, come illustrato nella tabella 3.2. Ovviamente il SubVI non effettua nessun controllo sul dato fornito in ingresso, semplicemente si occupa di modificare il vettore ad ogni iterazione.

| Step | Fase 1 | Fase 2 | Fase 3 | Fase 4 |
|---|---|---|---|---|
| 1 | T | F | F | F |
| 2 | F | T | F | F |
| 3 | F | F | T | F |
| 4 | F | F | F | T |
| 5 | T | F | F | F |

**Tabella 3.2:** Rotazione del vettore nella modalità wavemode

A questo punto l'esecuzione si sposta nel blocco che scrive fisicamente la nuova sequenza sui pin di uscita specificati nel canale virtuale. Risulta evidente che ogni valore impostato a *true* corrisponde alla soglia di tensione associata ad un valore logico alto. Le operazioni di rotazione del vettore e di scrittura sulle uscite della scheda vengono ripetute tante volte quanti sono gli step da eseguire sul motore. Si noti che all'interno del ciclo for è stato fissato un *timing* (riquadro verde), il quale stabilisce un intervallo di tempo prefissato che deve intercorrere tra un'iterazione e la successiva; questa temporizzazione sul ciclo permette di impostare la durata del periodo di ciascuno step.

Quando il motore ha compiuto tutti gli step richiesti, allora l'esecuzione esce dal ciclo for e si sposta in un nuovo blocco di scrittura che porta tutte le

uscite a un valore logico basso. Se non ci fosse quest'ultima fase, una volta terminato il ciclo le uscite della scheda rimarrebbero impostate sull'ultima sequenza elaborata e quindi una delle fasi rimarrebbe attiva fino ad una nuova pressione del pulsante di start.

### 3.2.3 Visualizzazione dell'andamento temporale delle uscite

In questa sezione vengono mostrati i risultati di alcune misure ottenute collegando un oscilloscopio MSO (Mixed Signal Oscilloscope) alle uscite della scheda DAQ. In particolare sono stati utilizzati i canali digitali dell'oscilloscopio perchè si è interessati ad evidenziare l'evoluzione temporale degli stati logici di uscita piuttosto che l'andamento dei fronti di salita e di discesa; i canali di ingresso dell'oscilloscopio, infatti, sono collegati ad un comparatore che confronta il segnale con una soglia di tensione prestabilita mostrando l'evoluzione temporale di due sole soglie di tensione corrispondenti a degli stati logici.

Sul pannello frontale del VI sono stati impostati i parametri riportati nella tabella 3.3 e successivamente è stato analizzato l'andamento temporale dei segnali digitali sulle uscite della scheda.

| Periodo di step | Modalità di funzionamento | Direzione di rotazione |
|---|---|---|
| 500 ms | Wavemode | Anti-oraria |

**Tabella 3.3**

Il diagramma temporale sui quattro pin di uscita della scheda è mostrato in Fig. 3.9.
Si può notare che, essendo stata impostata la modalità di funzionamento *wavemode*, in ciascun istante il diagramma temporale evidenzia un solo stato logico alto. Impostando i cursori dell'oscilloscopio in modo che venga misurato il periodo del primo canale di ingresso e il rispettivo *duty cycle*, si ottengono i risultati mostrati in Fig. 3.10 e riassunti nella tabella 3.4.

| Periodo complessivo | *Duty cycle* |
|---|---|
| 2 s | 25% |

**Tabella 3.4**

Tali risultati sono in accordo con i parametri impostati perchè un periodo di step di 500 ms implica che ciascuna fase venga attivata ogni 2 secondi.

A questo punto è stata cambiata la modalità di funzionamento lasciando invariati tutti gli altri parametri. Con la modalità *two-phases on*, il diagramma temporale dei segnali viene mostrato nella Fig. 3.11, mentre i risultati riguardanti la misura del periodo e del duty-cycle sono mostrati nella tabella

3.5.

| Periodo complessivo | *Duty cycle* |
|---|---|
| 2 s | 50% |

**Tabella 3.5**

Come si può notare dalla figura, in ciascun istante vengono abilitate contemporaneamente ad un valore logico alto due linee di uscita in modo tale che vi siano sempre due fasi attive del motore. Se si analizza il periodo di commutazione di una delle linee, si può notare che questa volta il duty cycle assume un valore pari al 50%, per cui ciascun impulso assume una durata pari alla metà del periodo di commutazione di ciascuna linea.

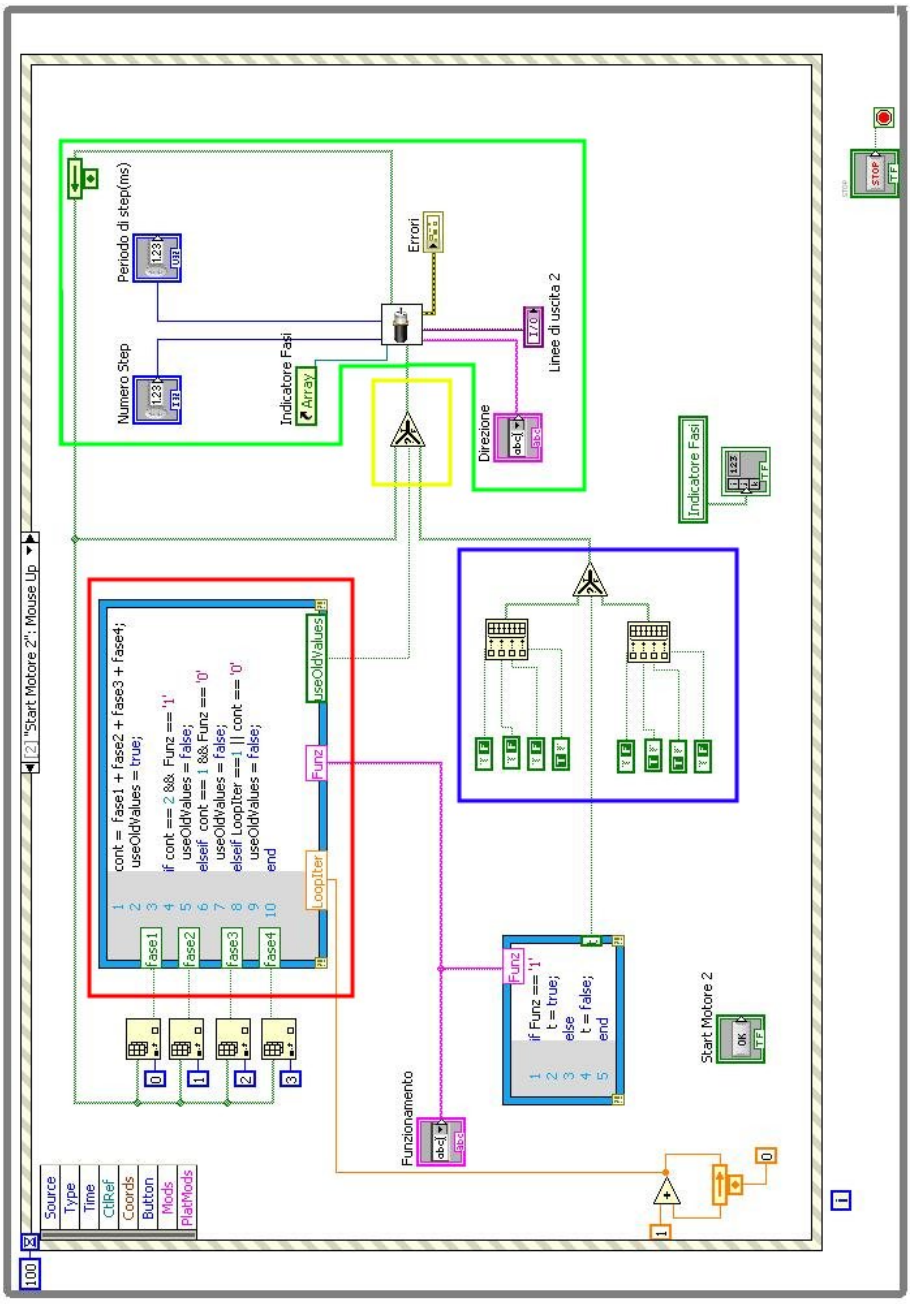



**Fig. 3.7:** Diagramma a blocchi del VI

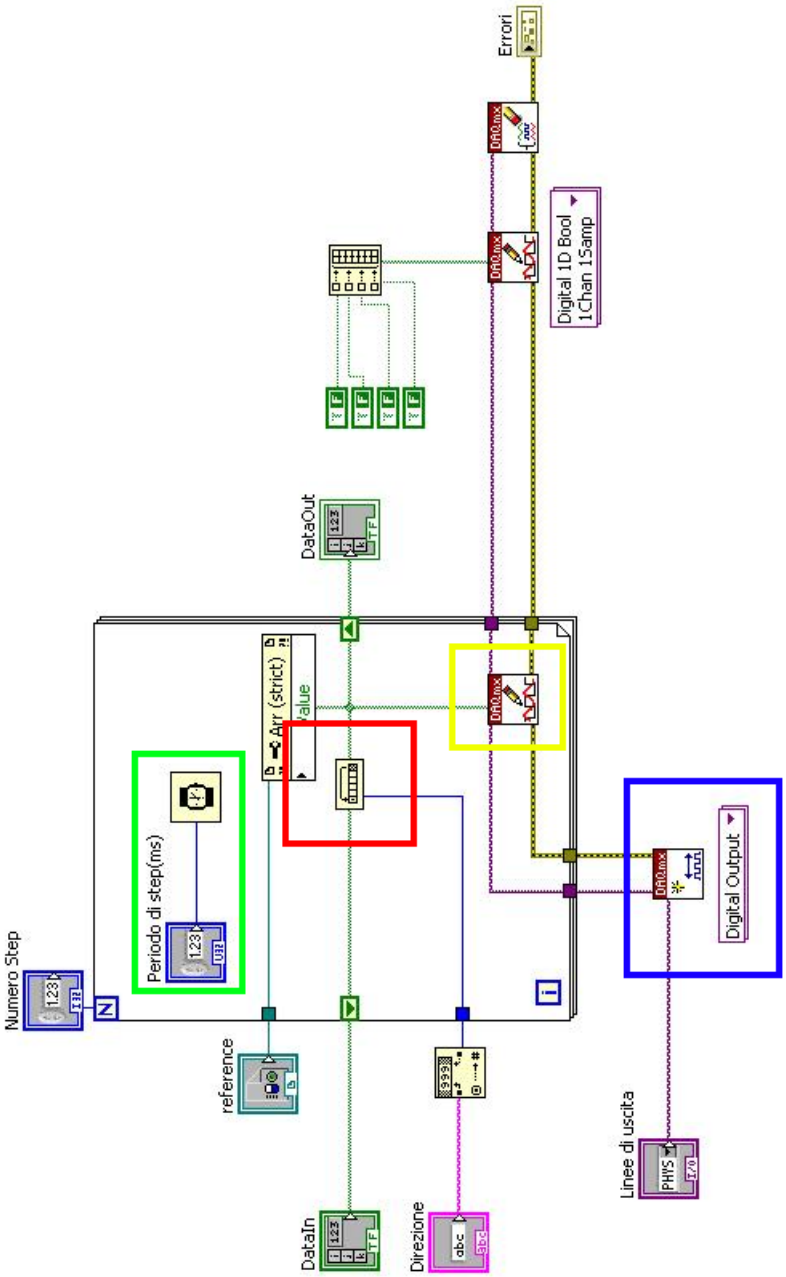



**Fig. 3.8:** Diagramma a blocchi del SubVI

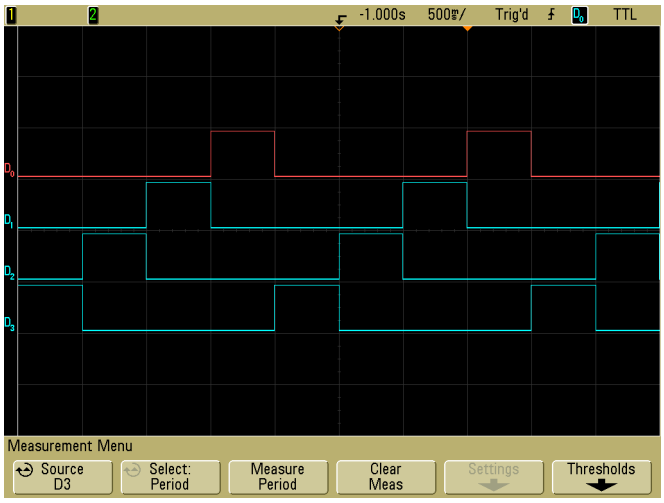



**Fig. 3.9:** Evoluzione temporale dei segnali digitali

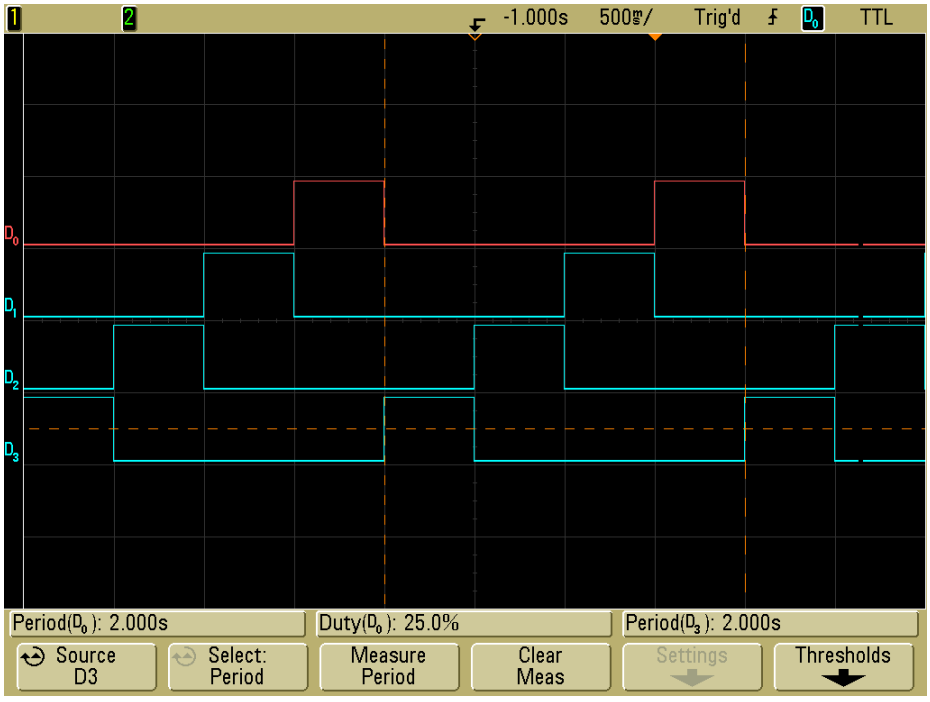



**Fig. 3.10:** Misura del periodo e del *duty cycle*

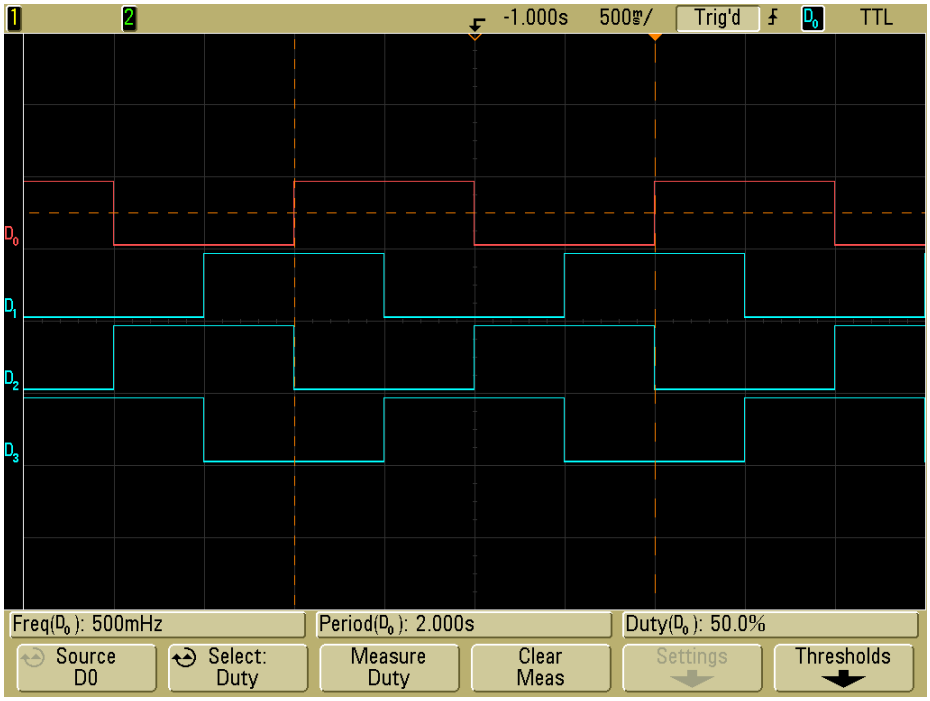



**Fig. 3.11:** Misura del periodo e del *duty cycle* in modalità *two-phases on*

# Conclusioni

L'obiettivo di questa tesi era di realizzare un sistema di automazione che sfruttasse la scheda NI DAQ 6008-USB, gestito tramite l'ambiente LabView. Il sistema di automazione prevedeva in particolare l'azionamento di due motori elettrici passo-passo unipolari, che permettevano rispettivamente di traslare e di ruotare un'apposita piattaforma, adibita a misure di compatibilità elettromagnetica. Nei precedenti capitoli si è cercato di fornire una documentazione più completa e chiara possibile della scheda DAQ 6008, illustrandone i principali pregi (ad esempio la possibilità di essere programmata interamente mediante LabView) e difetti (come la mancanza di un sistema di trigger versatile). Nel primo capitolo sono state descritte le caratteristiche hardware della scheda, con particolare attenzione alla circuiteria di ingresso e di uscita dei canali analogici e digitali, nonchè delle modalità di collegamento della scheda con carichi esterni. Nel secondo capitolo è stato visto, invece, come sfruttare la potente e versatile libreria DAQmx per realizzare le proprie applicazioni in LabView in modo efficiente. Infine, nel terzo capitolo sono stati esposti due esempi di utilizzo della scheda DAQ 6008 al fine di realizzare applicazioni che non fossero puramente didattiche come quelle descritte nel capitolo 2.
Sebbene a prima vista l'utilizzo della libreria DAQmx, in combinazione con la scheda DAQ 6008, possa sembrare fin troppo semplice e intuitivo, è di fondamentale importanza comprendere al meglio quali sono i limiti fisici del dispositivo, nonchè il funzionamento interno della libreria DAQmx stessa. In caso contrario, non solo si realizzerebbero applicazioni di scarsa efficienza, sprecando magari molto tempo nel tentativo di capire come mai non funzionano come previsto, ma si rischierebbe anche di danneggiare il dispositivo in uso. A tal proposito, la versione del programma che è stato sviluppato in questa tesi e che controlla i motori passo-passo, è solo l'ultima di varie versioni che sono state realizzate, ognuna delle quali migliorava qualche aspetto di quella precedente. Lo sviluppo di nuove versioni, infatti, era frutto di nuove conoscenze acquisite nell'ambito del dispositivo NI DAQ 6008, della libreria DAQmx, nonchè del funzionamento dei motori passo-passo. Nonostante ciò, quella proposta non è che una rudimentale versione del progetto, che ne realizza le funzionalità di base e da cui è possibile, in futuro, trarre una serie di spunti per migliorare il sistema e, eventualmente, implementare

ulteriori funzionalità. Ad esempio al pannello frontale si può aggiungere un elemento grafico che consenta di localizzare la posizione della piattaforma del servomeccanismo mediante un sistema di coordinate; in questo modo l'utente finale potrebbe gestire l'azionamento del servomeccanismo non solo premendo sui pulsanti di inizio rotazione ma anche specificando la precisa posizione della piattaforma rispetto ad un sistema di riferimento. Inoltre, si potrebbe gestire la rotazione contemporanea di entrambi i motori. Una delle ipotesi di partenza del progetto prevedeva infatti che si potesse azionare solo un motore alla volta; questo comportava una notevole semplificazione nella realizzazione del programma, ma ciò non toglie che in un futuro *upgrade* si cerchi di sfruttare il funzionamento *multitasking* dell'ambiente LabView. Qualora si ritenesse necessario, infine, si potrebbero introdurre dei sensori di fine corsa che inibiscono l'azionamento di un motore al raggiungimento di una posizione prestabilita: in questo caso la scheda DAQ verrebbe utilizzata non solo per generare dei segnali, ma anche per acquisirli da sorgenti esterne.

In definitiva, questo lavoro di tesi si pone come punto di partenza per tutti coloro che avranno la necessità di utilizzare la scheda di acquisizione ed elaborazione dati NI DAQ-6008 per applicazioni prevalentemente scritte in ambiente LabView, nonchè per chiunque necessiti di utilizzare il sistema di automazione sopra descritto.

# Elenco delle figure

# Bibliografia

[1] National Instruments, *Ground loops and returns*, Measurement Fundamentals (2010).