5. Esempi di semplici programmi di acquisizione dati

Prima di passare alla descrizione di alcuni esempi di programmi scritti in LabVIEW è necessario comprendere ancora due concetti che sono fondamentali per qualunque sistema di acquisizione dati: il **buffer** ed il **triggering**.

Il **buffer** è un'area di memoria del PC riservata all'allocazione temporanea dei dati prima che questi siano acquisiti dal computer per la loro elaborazione o la memorizzazione su memoria di massa.

Per esempio, supponiamo che si voglia acquisire qualche migliaio di campioni in un secondo. Potrebbe essere difficile trasferire sul video o su grafico tutti quei dati in un secondo; l'operazione di allocare i dati nel buffer è, invece, estremamente veloce. Pertanto si possono in un primo tempo immagazzinare i dati in una memoria volatile; in seguito quei dati saranno a disposizione per eventuali analisi e visualizzazioni.

Il buffer deve essere quindi proporzionato alla velocità ed al volume dei dati che si stanno acquisendo (l'operazione più critica è l'acquisizione analogica).
Se la scheda di acquisizione ha un canale DMA, le operazioni di input analogico dispongono di un cammino hardware preferenziale che collega direttamente la scheda con la RAM del computer; i dati sono cioè acquisiti direttamente nella memoria del computer.

Non si usa il buffer quando è necessario acquisire un punto alla volta, scaricarlo sul disco ed in seguito analizzarlo.

Riassumendo: si usa un I/O bufferizzato quando:

- Occorre acquisire o generare molti campioni ad un frequenza maggiore della capacità del PC di scriverli su video o di immagazzinarli su hard disk;
- La frequenza di campionamento deve essere uniforme; in altre parole il PC non deve interrompere anche per un istante l'acquisizione per passare ad effettuare operazioni di I/O.

Si può usare un I/O non bufferizzato quando:

- Il set di dati è piccolo (per esempio ogni canale non acquisisce più di un dato al secondo);
- Occorre ridurre al minimo l'impiego di memoria RAM (il buffer occupa RAM).

La parola **triggering** si riferisce ad ogni metodo mediante il quale si inizia, si termina o si sincronizza una acquisizione. Il trigger è di solito un segnale (analogico o digitale) la cui condizione è analizzata per determinare il successivo corso dell'azione.

Il triggering fatto via software è il metodo più semplice ed intuitivo per realizzare tale controllo (ad esempio un controllo Booleano che realizzi le operazioni di start e di stop di un'acquisizione dati).

Il triggering fatto via hardware può essere a sua volta suddiviso in **triggering interno** e **triggering esterno**.

Si realizza un **triggering interno** programmando l'output digitale di una scheda di acquisizione in modo che venga generato un impulso quando un canale analogico in ingresso raggiunge un certo livello di tensione; nell'istante in cui viene generato questo impulso inizia l'acquisizione.

Si opera con un **triggering esterno** quando una scheda, in attesa di un impulso digitale da uno strumento esterno, inizia l'acquisizione nell'istante in cui viene generato tale impulso.

Gli strumenti virtuali di LabVIEW per effettuare l'acquisizione dati sono organizzati in sottomenu che corrispondono al tipo di operazione che essi svolgono: ingressi analogici, uscite analogiche, I/O digitali, operazioni di conteggio.

Per accedere alle suddette opzioni, ci si deve posizionare sul block diagram ed attivare il Functions Palette sul quale scegliere l'opzione **Data Acquisition**.
All'interno di questo menu, i VI (**V**irtual **I**nstruments) per l'acquisizione dati sono organizzati in 6 sottomenu (vedi Fig. 1.5):

1. Analog Input
2. Analog Output
3. Digital I/O
4. Counter
5. Calibration and Configuration
6. Signal Conditioning

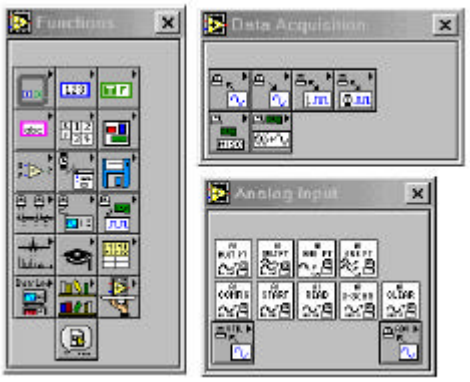


Fig.1.5 Menu delle Functions, sottomenu dei Data Acquisition e degli Analog Input

Ciascun menu contiene dei VI o dei sottomenu di VI organizzati in VI ***Easy***, VI ***Intermediate***, VI ***Utility*** e VI ***Advanced***.
Il sottomenu Analog Input, riportato in figura 1.5, mostra questo tipo di organizzazione: la prima riga contiene tutti i VI con ingressi analogici di tipo ***Easy***, la seconda riga contiene i VI di tipo ***Intermediate***, la terza riga contiene due sottomenu dei quali il primo consente di accedere ai VI di ***Utility*** ed il secondo ai VI di tipo ***Advanced***.

Inizieremo a sviluppare esempi che utilizzano VI di tipo *Easy*; in seguito si farà anche un esempio per il quale è necessario utilizzare VI di tipo ***Intermediate***, mentre i VI di tipo ***Advanced*** non rientrano negli scopi di questi appunti.

I VI di tipo ***Easy*** sono VI ad alto livello, ideali per semplici esigenze di DAQ o di I/O Digitale; essi includono un metodo semplificato per trattare gli errori: quando durante l'esecuzione di un VI si verifica un errore DAQ, viene visualizzata una finestra di dialogo relativa all'errore e si ha la possibilità di scegliere se interrompere l'esecuzione dl VI od ignorare l'errore.

I VI di tipo ***Intermediate*** hanno, rispetto ai precedenti, maggiori funzioni hardware, sono più flessibili e consentono una maggiore efficienza nello sviluppo delle applicazioni e sono i più indicati per la maggior parte dei casi. Essi presentano una gestione degli errori più flessibile; con ciascun VI è possibile trasmettere informazioni sulle condizioni di errore ad un altro VI e gestire gli errori da programma.

I VI di tipo ***Advanced*** sono le interfacce più a basso livello; pochissime applicazioni richiedono i VI ***Advanced*** ed, in generale, i VI ***Easy*** oppure ***Intermediate*** sono sufficienti per la maggior parte delle applicazioni.

Acquisizione Single Point

Il più semplice esempio di programma di acquisizione dati utilizza il VI **AI Sample Channel** (vedi Fig. 2.5); tale VI acquisisce la tensione generata da un trasduttore collegato al canale specificato di una scheda di acquisizione dati e ne restituisce il valore:

- **Device** è l'indirizzo della scheda DAQ specificato al momento della sua configurazione mediante il programma NI_DAQ Configuration Utility,
- **Channel** è una stringa di caratteri che specifica il numero del canale di input analogico,
- **High limit** e **Low limit** specificano il range del segnale in ingresso.

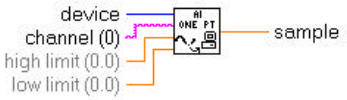


Fig.2.5 Schema degli input ed output di AI Sample Channel.vi

In Fig. 3.5 è mostrato il Front Pannel ed in Fig. 4.5 il relativo block diagram di un semplice programma che utilizza AI Sample Channel.vi. Questo programma effettua una serie di loop; ad ogni esecuzione del loop effettua l'acquisizione del canale al quale è collegato il segnale, diagramma il punto acquisito e controlla lo stato della variabile booleana collegata all'interruttore di stop e si arresta dopo che è stato premuto il bottone di stop.

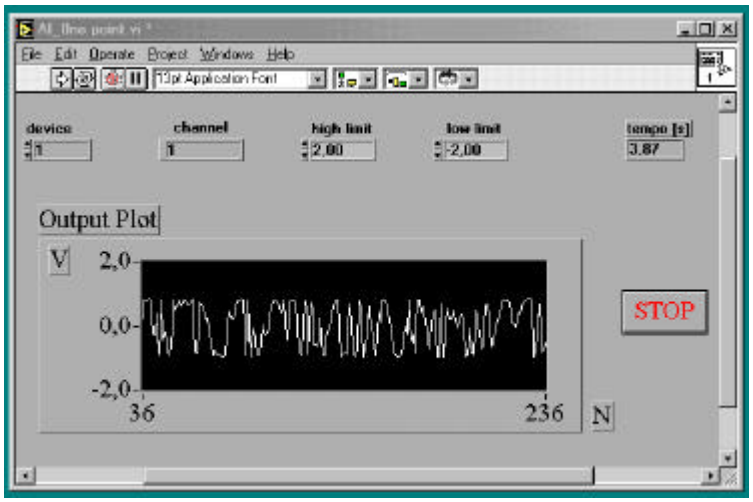


Fig. 3.5 Control Pannel del programma AI_One point.vi

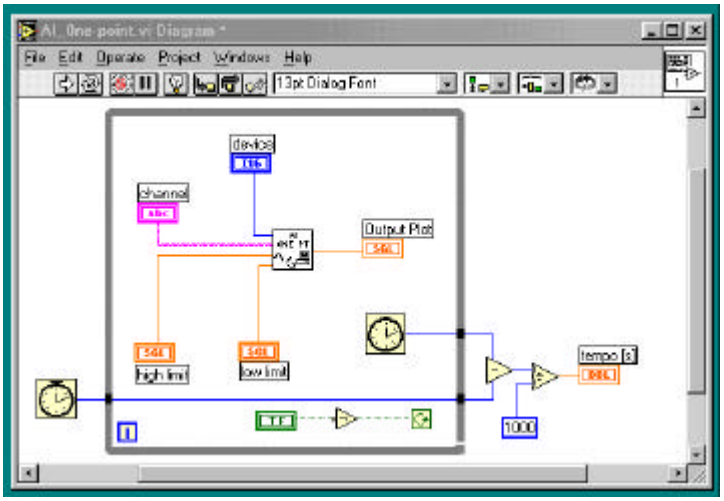


Fig. 4.5 Block diagram del programma AI_One point.vi (I° versione)

Questo programma consente di effettuare una importante considerazione: ricordando che ad ogni loop il programma effettua le seguenti operazioni:

- acquisizione di un punti
- disegno del punto sul grafico
- misura del tempo
- verifica dello stato dell'interruttore booleano

e che l'output grafico, effettuato al termine di ogni loop, è una delle operazioni che richiedono più tempo al sistema anche in presenza di una scheda grafica molto veloce, già alla frequenza di 100 Hz la riproduzione della sinusoide è molto scadente. Questo è dovuto al fatto che la frequenza di acquisizione è molto bassa (circa 61 Hz).

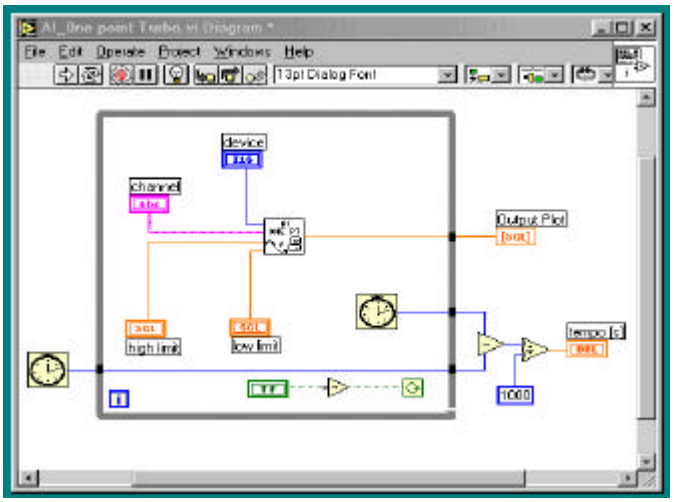


Fig. 5.5 Block diagram del programma AI_One point.vi (II° versione)

Un modo per velocizzare il programma è di riorganizzare l'acquisizione effettuando ad ogni loop le seguenti operazioni (vedi Fig. 5.5):

- acquisizione di un punto
- misura del tempo
- verifica dello stato dell'interruttore booleano

L'output grafico viene effettuato al termine dell'esecuzione di tutti i loop programmati; i dati acquisiti vengono memorizzati in un vettore e solamente alla fine del ciclo di acquisizione tutti punti acquisiti vengono rappresentati su grafico. La frequenza di acquisizione sale ad 83 Hz. Un ulteriore incremento di velocità del codice si può ottenere effettuando la misura del tempo solamente all'inizio ed alla fine del ciclo di acquisizioni come mostrato nell'esempio di figura 6.5a, 6.5b, 6.5c, 6.5d. E' interessante, dal punto di vista didattico, osservare che all'interno del block diagram viene utilizzata la struttura: la **sequence**. Infatti solo una struttura di questo tipo assicura che le varie operazioni avvengano in una successione ben definita; altrimenti è LabVIEW che decide la sequenza delle operazioni. Nel frame 1 della sequence (Fig. 6.5a) viene misurato l'istante iniziale; nel frame 2 (Fig. 6.5b) vengono effettuate le operazioni di acquisizione, nel frame 3 (Fig. 6.5c) viene misurato l'istante finale dell'acquisizione ed infine nel frame 3 (Fig. 6.5d) tutti i punti acquisiti vengono riportati su grafico.

Ricordiamo che la sequence consente di trasferire i dati da una sequence ad un qualunque altra successiva; bisogna usare un terminale chiamato **sequence local**. Per crearlo, posizionarsi con la freccia od il rocchetto sul bordo della sequence, premere il tasto destro del mouse: appare un menu di pop up; tra le varie opzioni scegliere **Add sequence local**. Quando appare sul diagramma, un sequence local si presenta sotto la forma di un piccolo rettangolo giallo. Se lo si collega ad una sorgente di dati, il rettangolo si trasforma in una freccia diretta verso l'esterno della cornice.

Nell'esempio di Fig. 6.5a il misuratore di tempo è collegato al sequence local e, dato che il dato trasferito è un numero intero, la freccia assume il colore blu.

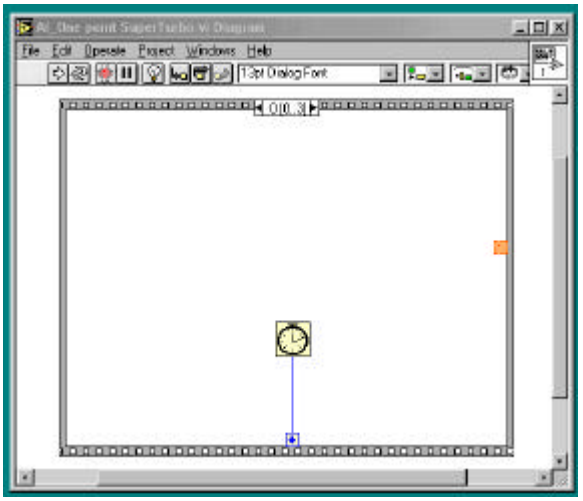


Fig. 6.5a Block diagram del programma AI_One point.vi (III° versione) – Frame n. 1

In Fig. 6.5b è illustrato il secondo frame. Si osservi che sulla cornice appare una freccia blu in ingresso che consente di trasferire in questo frame il valore che compare nel frame precedente; dato che questa operazione è rinviata ad una fase successiva, questo terminale risulta non collegato. Si nota, invece, una nuova sequence local che serve a trasferire ad una fase successiva il vettore che contiene i risultati acquisiti: freccia uscente di colore arancione.

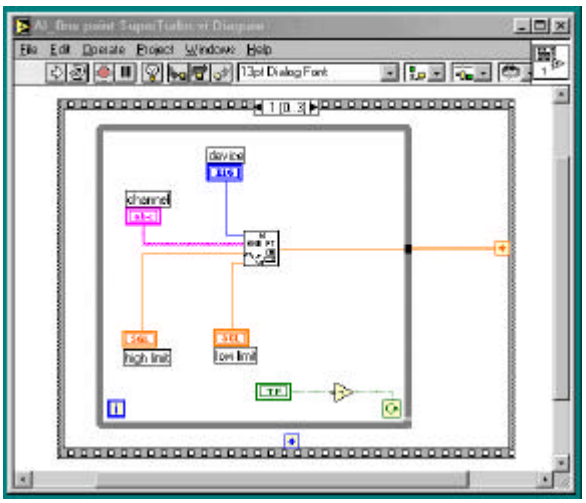


Fig. 6.5b Block diagram del programma AI_One point.vi (III° versione) – Frame n. 2

La Fig. 6.5c illustra il terzo frame. La freccia blu in ingresso trasferisce in questo frame il valore del tempo misurato in precedenza nel primo frame: questo consente di misurare il tempo trascorso tra l'istante che precede l'inizio del loop di acquisizione e l'istante immediatamente successivo alla conclusione del suddetto loop.

Inoltre si nota una nuova sequence local che serve a trasferire alle varie ***sequence*** il vettore che contiene i risultati acquisiti: freccia uscente di colore arancione.

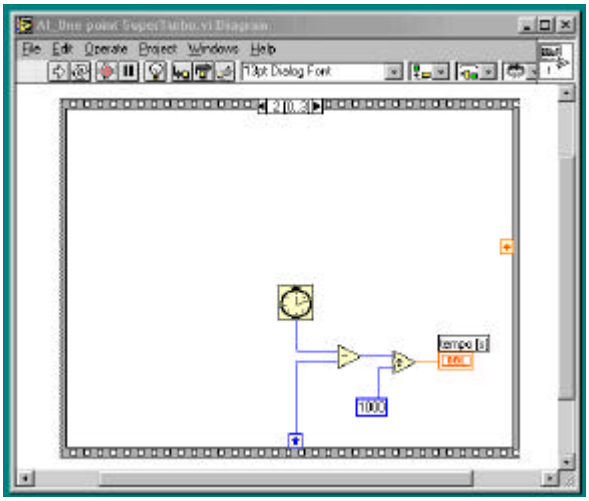


Fig. 6.5c Block diagram del programma AI_One point.vi (III° versione) – Frame n. 3

La Fig. 6.5d illustra il quarto frame. La freccia arancione in ingresso trasferisce in questo frame il valore del vettore delle acquisizioni generato in precedenza nel secondo frame. In questo frame si genera il diagramma dei punti acquisiti. In questo caso la frequenza di acquisizione sale ad 94 Hz.

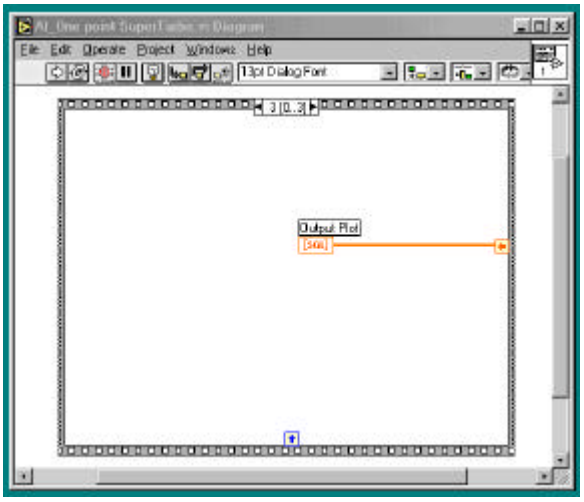


Fig. 6.5d Block diagram del programma AI_One point.vi (III° versione) – Frame n. 3

In conclusione questo tipo di VI consente di effettuare solo acquisizioni a bassa frequenza (fenomeni che non superino i 20-30 Hz). Nel caso si debba acquisire un segnale a frequenza superiore bisogna ricorre a codici di struttura differente.

Acquisizione Multy Points singolo canale

Nella maggior parte delle applicazioni che richiedono un'analisi nel tempo del segnale l'acquisizione di un punto alla volta, descritta in precedenza, non soddisfa le esigenze di velocità che questo tipo di acquisizioni richiede
Inoltre l'acquisizione single point difficilmente garantisce che l'intervallo di campionamento tra un punto e l'altro sia costante; infatti tali intervalli dipendono da diversi fattori difficilmente controllabili dal programmatore.

Descriviamo ora un semplice esempio di programma di acquisizione dati che utilizza il VI **AI Acquire Waveform.vi** (vedi Fig. 7.5); tale VI acquisisce **n** punti da un segnale collegato alla scheda di acquisizione dati al canale specificato e restituisce i valori della tensione misurata:

- **device** è l'indirizzo della scheda DAQ specificato al momento della sua configurazione mediante il programma NI_DAQ Configuration Utility,
- **channel** è una stringa di caratteri che specifica il numero del canale di input analogico,
- **number of samples** è il numero di acquisizioni che si è programmato di effettuare,
- **sample rate** è la frequenza di acquisizione prevista,
- **high limit** e **low limit** specificano il range del segnale in ingresso,
- **waveform** è la forma d'onda acquisita in un vettore,
- **actual sample period** è l'effettiva frequenza di campionamento misurata al termine dell'acquisizione. Questo numero può essere leggermente differente dal valore di campionamento richiesto in relazione alle possibilità offerte dall'hardware di cui si dispone.

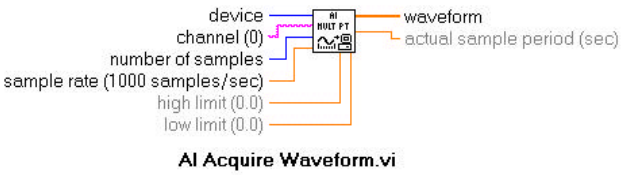


Fig. 7.5 Schema degli input ed output di AI Acquire Waveform.vi

In Fig. 8.5 è mostrato il Front Pannel ed in Fig. 9.5 il relativo block diagram di un semplice programma che utilizza AI Acquire Waveform.vi e, alla termine dell'acquisizione, disegna il grafico della forma d'onda acquisita.

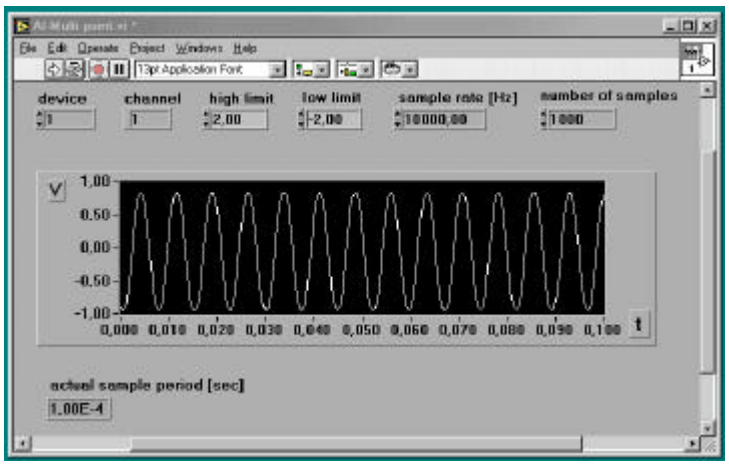


Fig.8.5 Control Pannel di un programma che utilizza AI Acquire Waveform.vi

Come si può immediatamente osservare il block diagram è molto semplice; il Vi **AI Acquire Waveform** esegue in pratica tutte le operazioni necessarie per effettuare una acquisizione dati.

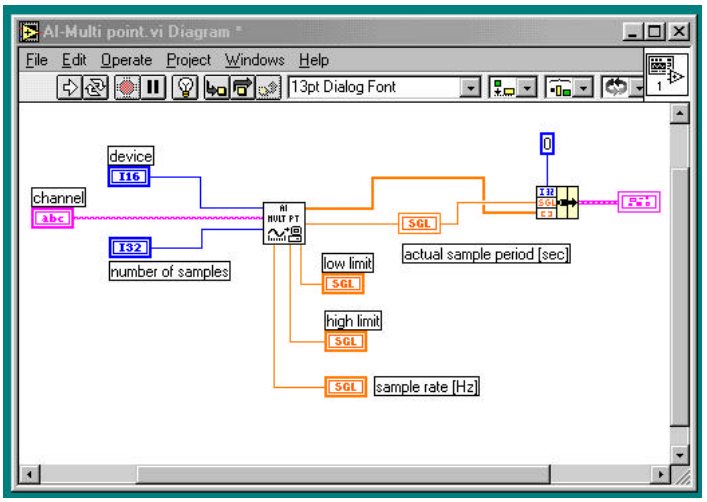


Fig. 9.5 Block Diagram di un programma che utilizza AI Acquire Waveform.vi

E' interessante osservare la funzione **Bundle** usata per la fase grafica il cui uso è illustrato in figura 10.5. Questa funzione consente di specificare il valore dell'ascissa iniziale del grafico $\mathbf{x_0}$ ed il passo **Dx** tra un punto e il punto successivo: specificando come Δx l'effettiva frequenza di campionamento misurata al termine dell'acquisizione (Actual sample period) l'asse x diventa l'asse dei tempi.

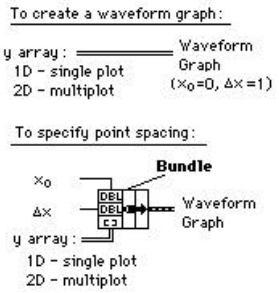


Fig. 10.5 Uso della funzione **Bundle** nel caso di un **Waveform Graph**

Infine l'esecuzione del programma consente di fare una interessante considerazione: se la frequenza del segnale che si intende acquisire può essere variata con continuità, si può individuare un valore delle frequenza di acquisizione limite:

- ♦ al di sopra di tale frequenza il segnale acquisito descrive con fedeltà il segnale da acquisire; perché ciò si verifichi la frequenza di acquisizione deve essere almeno il doppio della frequenza del fenomeno da acquisire;
- ♦ al di sotto di tale frequenza non si è in grado di ottenere una attendibile ricostruzione temporale del segnale ma si può solamente risalire al suo valore medio. Al crescere della frequenza del segnale da acquisire, si può andare incontro al fenomeno **aliasing** che può trarre in inganno lo sperimentatore in quanto si è portati ad individuare frequenze che in realtà non esistono.

Acquisizione Multy Points a più canali

Oltre alla funzione illustrata nell'esempio, esiste anche una funzione analoga che è in grado di effettuare l'acquisizione di più canali (Fig. 11.5)

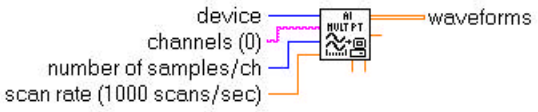


Fig. 11.5 Schema degli input ed output di AI Acquire Waveforms.vi

AI Acquire Waveform acquisisce da più canali in ingresso e mette a disposizione il numero specificato di campionamenti alla velocità di scansione indicata:

- **device** è l'indirizzo della scheda DAQ;
- **channel** è una stringa che specifica i canali analogici da acquisire; i canali nella stringa sono separati da una virgola (per esempio 1,2,4);
- **number of samples/ch** è il numero di campioni per canale da acquisire**;**
- **scan rate** è il numero di campioni da acquisire al secondo per ciascun canale;
- **high limit** e **low limit** specificano l'intervallo di valori del segnale in ingresso (i valori di default sono +10 V e – 10V rispettivamente);
- **waveform** è un vettore 2D contenete i dati letti dagli ingressi analogici, espressi in Volt;
- **actual scan period** è l'inverso della velocità di scansione reale utilizzata; questo numero può differire lievemente dalla velocità di campionamento richiesta, a seconda delle prestazioni dell'hardware.

L'esempio illustrato nella figura seguente mostra il VI **AI Acquire Waveforms** nel caso in cui si desideri effettuare una scansione su quattro canali. La sequenza di scansione è 1,2,4,6.
Per ciascun canale si acquisiscono 1000 campioni a 20000 Hz. **AI Acquire Waveforms** restituisce un vettore 2D; i dati relativi al primo canale sono registrati nella colonna 0, quelli del secondo canale nella colonna 1, e così via. La funzione **Index Array**, descritta in figura 13.5, estrae i dati per ciascun canale (un vettore 1D).

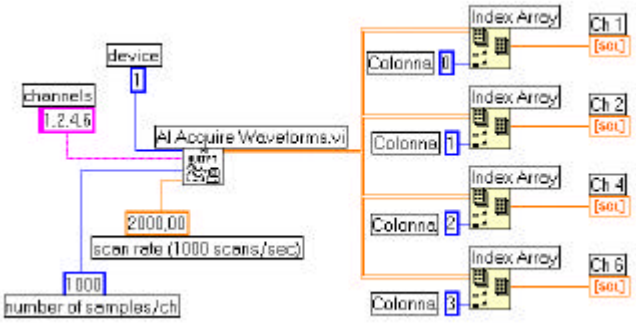
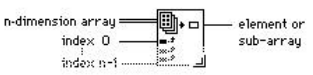


Fig. 12.5 Schema di un block diagram che effettua un'acquisizione multipoints e multicanale



Index Array

Returns an element of array at index. If array is multidimensional you must add additional index terminals by resizing or popping up and adding terminals. You can slice out arrays (e.g.: rows/columns) by disabling index terminals from the popup.

Fig. 13.5 Uso della funzione **Index Array** per l'estrazione di un elemento da una matrice

E’ importante menzionare una limitazione dell’I/O multicanale: se si impone un valore elevato al parametro **scan rate** e si diagrammano i risultati ottenuti in funzione del tempo (non in funzione dell’indice della matrice che li contiene), si osserva un ritardo di fase tra i vari canali. A cosa è dovuto questo fatto ?
La maggior parte delle schede di acquisizione è in grado di effettuare solamente una conversione A/D alla volta. Pertanto i dati sono digitalizzati in sequenza, un canale alla volta. Nasce pertanto un ritardo tra i vari canali (Fig. 14.5), detto **Interchannel Delay**.

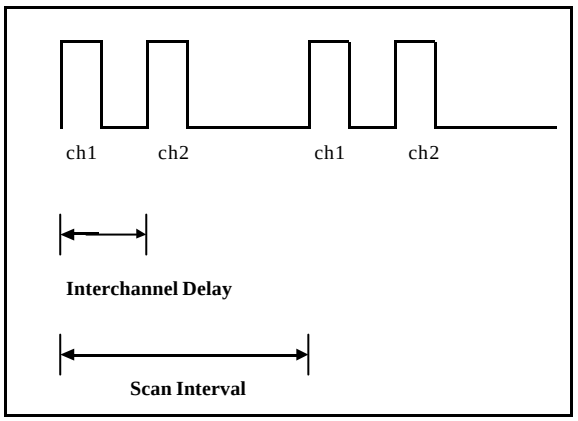


Fig. 14.5 Schematizzazione dell’Interchannel Delay

In generale questo ritardo è molto piccolo e nel caso di una misura di un segnale continuo o a bassa frequenza, questo ritardo non costituisce alcun problema. L’interchannel delay può essere così piccolo rispetto al periodo di acquisizione da far sembrare che la scheda acquisisca virtualmente tutti i canali simultaneamente. Per esempio, se l’interchannel delay è dell’ordine dei microsecondi e si stanno acquisendo segnali con frequenza di campionamento dell’ordine dell’Hertz, questo fenomeno è assolutamente trascurabile. Tuttavia, ad alta frequenza questo ritardo risulta evidente e possono sorgere dei problemi se i vari canali devono essere sincronizzati.

Real-time data acquisition

L'acquisizione in continua di dati (o real-time data acquisition) trasferisce i dati acquisiti dal buffer del sistema ad un output grafico o, nel caso di frequenze di campionamento elevate, a memorie di massa nel corso stesso dell'acquisizione senza che ciò comporti interruzioni nel processo di acquisizione o perdita di dati.
Questo approccio, solitamente, richiede l'uso di uno schema che prende il nome di **buffer circolare**:

- La scheda di acquisizione acquisisce i dati e li trasferisce nel buffer; questo processo di trasferimento è molto veloce in quanto sfrutta i canali DMA (accesso diretto alla memoria) senza coinvolgere in alcun modo l'attività del microprocessore.
  In generale una normale periferica (per esempio la tastiera) effettua una chiamata al microprocessore ogni volta che vuole trasferire un'informazione dalla periferia al sistema centrale: questo avviene tramite una chiamata tramite il proprio canale di IRQ, che consiste in un passaggio di stato, cioè in una variazione del livello di tensione, in altre parole un vero e proprio semaforo. Il microprocessore ciclicamente controlla lo stato dei vari IRQ. Se rileva che su un determinato canale c'è una richiesta, il microprocessore la esamina ed esegue la relativa procedura richiesta. Nel caso del DMA tutto questo è bypassato ed i dati sono trasferiti direttamente in memoria RAM.
- E' evidente che dopo un po' di tempo il buffer (cioè l'area di memoria riservata a deposito temporaneo dei dati) si esaurisce. E' necessario allora svuotare il buffer prima che questo sia completamente pieno. Lo schema è quello illustrato in Fig. 15.5.

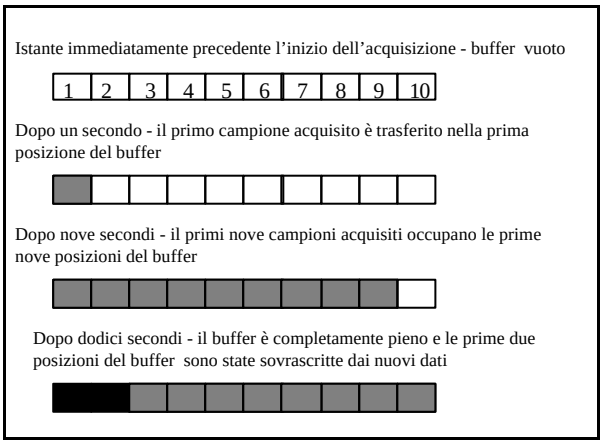


Fig.15.5 Schema della gestione dei dati mediante un **buffer circolare**

Per poter effettuare un'acquisizione continua è necessario trasferire la prima parte del buffer su memoria di massa. In questo modo si evita la perdita di dati.

E' evidente che perché il buffer non si saturi, è necessario che la **velocità del trasferimento dei dati** sulla memoria di massa non sia inferiore alla velocità di acquisizione. In questo caso il collo di bottiglia è rappresentato dal processo di scrittura su hard disk: innanzitutto è necessario che il processore non esegua contemporaneamente altre operazioni che potrebbero compromettere la continuità nel trasferimento dei dati (ad esempio l'attivazione dello screen saver, lo spostamento del mouse, ecc.) .

Inoltre, nel caso di alte frequenze di acquisizione, è necessario che il formato con cui si trasferiscono i dati sia il più compatto possibile. In questo caso il formato binario è il più conveniente.
Per esempio la scrittura su hard disk di un file di **100 numeri interi ad 8 bit**, cioè 100 numeri compresi tra 0 e 256 (un numero a 8 bit è un numero compreso tra 0 e 2^8=256) **richiede 100 bytes**,

mentre il corrispondente **file ASCII richiede 400 bytes**. Questo è dovuto al fatto che ogni numero intero ad 8 bit in formato binario richiede un solo byte (1 byte = 8 bit) mentre ogni singola cifra di tale numero in formato testo richiede un byte a cui si deve aggiungere un byte per il delimitatore di spazio che separa un numero dall'altro.

E' evidente che ai vantaggi nella alta velocità di scrittura e nella ridotta occupazione di spazio, si contrappone lo svantaggio determinato dall'impossibilità di leggere i file con un comune editor di testo.

In Fig. 16.5 è riportato il control pannel ed in Fig. 17.5 il block diagram di un programma che effettua un'acquisizione in continua; i risultati dell'acquisizione sono riportati su di un grafico.

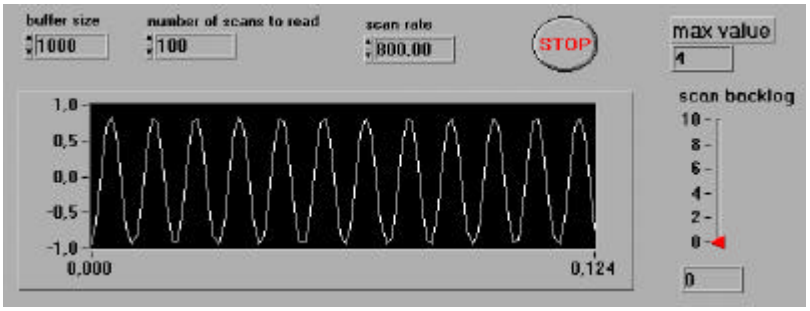


Fig. 16.5 Control pannel di un VI che effettua acquisizione in continua

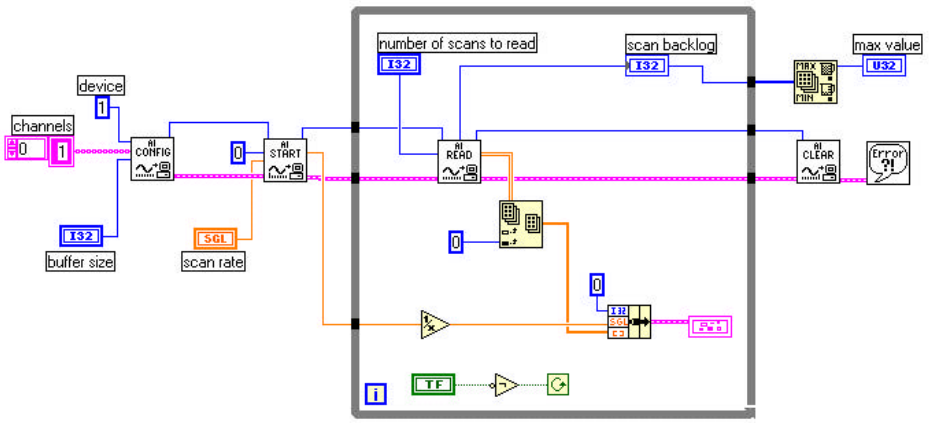
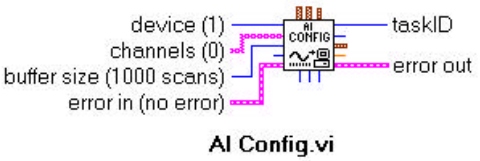


Fig. 17.5 Block diagram di un VI che effettua acquisizione in continua

In questo programma compaiono quattro nuove icone:

1. **AI Config**: configura l'operazione di input analogico per un set di canali definito ed alloca un buffer nella memoria del computer.



AI Config.vi

Principali parametri in ingresso:

device è un numero intero che viene definito in fase di configurazione della scheda, come illustrato in precedenza, e serve ad identificarla;

channels è una stringa di caratteri che serve a definire i canali che verranno utilizzati per l'acquisizione;
buffer size è un numero intero che definisce la dimensione del buffer in base al numero di scansioni che si intende effettuare ed alloca la memoria necessaria.

Principali parametri in uscita:

task ID è un numero intero, associato al device ed ai canali e serve ad identificarli;
Error Out è un cluster contenente tutte le informazioni sugli errori.

2. **AI Start**: inizia l'acquisizione bufferizzata

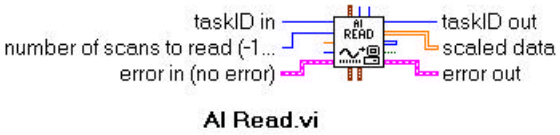
Principali parametri in ingresso:

task ID è il numero intero definito in precedenza; poiché esso è un input ed un output per i vari VI che seguono AI Config.VI, si crea una dipendenza sequenziale tra i vari VI che effettuano l'acquisizione garantendo in questo modo una corretta sequenzialità nelle varie fasi del processo di acquisizione;
number of scans to acquire è un numero intero che definisce il numero di scansioni che si intende effettuare per ogni canale. Se si impone che questo parametro valga 0, LabVIEW acquisisce i dati e li trasferisce nel buffer senza interruzione.
scan rate è un numero che definisce la frequenza di acquisizione per ogni canale.

Principali parametri in uscita (entrambi definiti in precedenza):

task ID
error Out

3. **AI Read**: legge i dati dal buffer allocato da Ai Config.VI



Principali parametri in ingresso:

task ID (definito in precedenza);
number of scans to read è un numero intero che definisce il numero di punti che si intende leggere dal buffer.

Principali parametri in uscita:

task ID (definito in precedenza);
error Out (definito in precedenza);
scaled Data è un matrice bidimensionale che contiene i dati letti dal buffer (ogni colonna di dati è associata al corrispondente canale indicato nella Channel list.

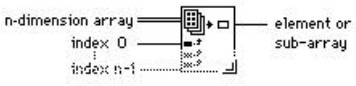
4. **AI Clear**: cancella il buffer e disalloca tutte le risorse coinvolte nell'acquisizione.

AI Clear.vi

E' utile, infine, fare qualche considerazione sulla struttura del programma.

Ponendo a zero il parametro **number of scans to acquire**, il programma effettua una acquisizione continua.
La funzione **while loop** consente di effettuare ad ogni ciclo la lettura del buffer. Il programma si arresta quando l'interruttore boleoano all'interno del ciclo cambia di stato.

In questo codice viene anche utilizzata la funzione Index Array il cui uso descriviamo in breve:



Index Array

Tale funzione consente di estrarre da una matrice un elemento. Se la matrice ha più dimensioni è necessario includere nuovi indici: posizionare il rocchetto sul piccolo rettangolo pieno denominato index e, come al solito, premere il tasto destro del mouse scegliendo la funzione **add dimension.**
Per estrarre una intera colonna, come nel presente esempio, posizionare il rocchetto sul primo rettangolo (index) e, come al solito, premere il tasto destro del mouse scegliendo la funzione **Disable Indexing**; a secondo terminale collegare una costante intera il cui numero identifica la colonna (nel caso presente 0) che si vuole estrarre.

E' interessante osservare che, grazie alla miglior gestione del sistema di acquisizione che questo tipo di VI effettua, si può operare a frequenze di acquisizione molto elevate. Nel caso illustrato in Fig. 17.5 si sta operando ad una frequenza di 800 Hz e, contemporaneamente, in tempo reale si visualizza l'acquisizione su grafico.

Nel caso in cui si rinunci alla fase grafica in tempo reale ed si decida di salvare i dati su hard disk in formato binario, si raggiungono frequenze di acquisizione in continuo estremamente più elevate.

Se si esegue il programma illustrato nelle figure 17.5 e 18.5, si osserva infine che è sufficiente ad esempio muovere il mouse, aprire il programma gestione risorse durante l'esecuzione ed il parametro **scan backlog** passa dal valore 0 a valori molto elevati; questo parametro è la differenza tra i dati immessi nel buffer ed i dati letti dal buffer. Se questo numero è positivo è evidente che si ha una perdita di dati, cioè il buffer si riempie più velocemente di quanto si riesca a leggere da esso; in questo modo si verifica una perdita dei dati in quanto il buffer viene sovrascritto prima venga effettuata la lettura.

E' evidente che nel caso illustrato in figura l'operazione di diagrammazione su video è molto gravosa per il microprocessore che, per questo motivo, non riesce ad effettuare tutte le letture dal buffer.

Buona norma è pertanto quella di ridurre al minimo le operazione che deve effettuare il microprocessore per ridurre al minimo possibile la perdita di dati.

Analisi di un segnale mediante FFT (Fast Fourier Transforms)

L'analisi dei dati è una delle fasi più importanti dell'intero processo di acquisizione. In questo capitolo esamineremo in dettaglio un programma molto elementare in grado di fornire lo spettro in frequenza di un segnale.

Ricordiamo in breve le fasi essenziali di un processo di acquisizione: un sensore rileva la grandezza fisica che si intende acquisire e la trasforma in un segnale elettrico; questo segnale viene trasmesso al sistema di acquisizione che a sua volta lo converte in un valore digitale e lo memorizza su di un buffer di memoria o su hard disk. E' opportuno sottolineare che:

- ciascuno di essi corrisponde al valore che la grandezza fisica ha assunto in un ben preciso istante;
- due valori successivi sono stati acquisiti in istanti che distano tra loro di un intervallo di tempo Δt.

In altri termini: il trasduttore genera un segnale continuo nel tempo; tuttavia l'informazione su quella parte del segnale che è compresa tra due conversioni A/D successive viene completamente ignorata a differenza di quanto avviene nel caso di un'acquisizione analogica in cui la memorizzazione progredisce in modo continuo (esempio tipico è quello della memorizzazione su nastro magnetico).

Le trasformate di Fourier sono uno strumento matematico molto potente di analisi del segnale, che può essere applicato sia a segnali rilevati in funzione del tempo che a segnali rilevati in funzione dello spazio. Le DFT (Discrete Fourier Transforms) si applicano nel caso in cui si intenda analizzare un segnale digitalizzato, espresso quindi in modo discreto.

All'inizio del secolo XIX il matematico francese Jean Baptiste Joseph Fourier dimostrò che ogni funzione **g(t)** dal comportamento ragionevolmente periodico di periodo **T** può essere costruita mediante la somma di un certo numero (possibilmente infinito) di seni e coseni.
Nel 1823 egli pubblicò un lavoro sulla teoria della trasmissione del calore: in questo lavoro egli introdusse per la prima volta la rappresentazione di una funzione espressa in una serie di seni e di coseni, oggi nota come serie di Fourier.

La trasformazione di un segnale dipendente dal tempo nel corrispondente spettro in frequenza può essere illustrata nel modo seguente: un segnale acustico, udibile da un orecchio umano, consiste in una fluttuazione di pressione nel campo di frequenze compreso tra 30 Hz e 20 kHz. A differenza di un microfono, che cattura l'andamento temporale delle fluttuazioni di pressione, l'orecchio umano cattura direttamente la frequenza. L'analisi spettrale di un segnale acustico è dovuta al fatto che le cellule poste nell'orecchio interno, sfruttano il fenomeno della risonanza, effettuando un'analisi selettiva in frequenza: se si stimola l'apparato uditivo con un segnale acustico costituito da un puro tono sinusoidale, solo le cellule "calibrate" su detta frequenza vengono eccitate. In un certo modo l'orecchio effettua un'analisi di Fourier.

Non è compito di questo breve corso illustrare in dettaglio i fondamenti matematici che vi sono alla base delle trasformate di Fourier; tuttavia forniamo qui di seguito le formule essenziali che consento l'uso di questo fondamentale strumento matematico.

L'integrale di Fourier di una funzione **g(t)** continua nel tempo è definito come segue:

$$\mathbf{X(f) = \int_{-\infty}^{+\infty} x(t)\, e^{-i 2 \pi f t} dt} \qquad \mathbf{(1.5)}$$

Da questa relazione si deduce che, se è noto l'andamento temporale della funzione **x(t)**, è nota anche la sua distribuzione spettrale.
Come già detto, il processo di acquisizione digitale campiona il segnale **x(t)** in un certo numero di valori discreti; la durata di tale processo è finita e vale $\mathbf{T_M}$. Sia **N** il numero totale dei valori

acquisiti dal segnale analogico **x(t)** durante il tempo $\mathbf{T_M}$; ne segue che la distanza temporale tra i singoli campioni acquisiti vale $\mathbf{DT = T_M / (N-1)}$.
Pertanto un generico istante $\mathbf{t_k}$, corrispondente alla **k-esima** acquisizione, è individuato dalla relazione seguente:

$$\mathbf{t_k = k\,DT} \qquad \text{ove} \qquad \mathbf{k=0,1 \ldots. N-1}$$

Da quanto detto, risulta evidente che un'acquisizione discreta di valori consentirà solamente una stima della trasformazione di Fourier per valori discreti della frequenza.

Nel caso delle DFT l'integrale di Fourier dato dall'espressione (1.5) assume la forma seguente:

$$\mathbf{X_n = X(f_n)} = \sum_{k=0}^{N-1} \mathbf{x_k}\, \mathbf{e}^{i2\delta k\frac{n}{N}} \tag{2.5}$$

ove $\mathbf{f_n}$ è dato da:

$$\mathbf{f_n} = \frac{\mathbf{n}}{\mathbf{N}\cdot \ddot{A}\mathbf{T}} \qquad \text{ove} \qquad \mathbf{0 \le n \le (N-1)}$$

Pertanto il diagramma della funzione $\mathbf{x_k}$ espressa in funzione del tempo riporta sull'asse delle ascisse i valori corrispondenti agli istanti in cui sono state effettuate le varie acquisizioni che, come già detto, distano tra loro di **DT** e sull'asse delle ordinate i valori che la funzione assume in tali istanti.

Nel caso dello spettro in frequenza di un segnale illimitato nel tempo (ad esempio una funzione armonica od un rumore) sull'asse delle ascisse vengono riportati i valori corrispondenti alle frequenze $\mathbf{f_n}$ distanziate tra loro di:

$$\ddot{A}\mathbf{f} = \frac{\mathbf{1}}{\mathbf{N}\cdot \ddot{A}\mathbf{T}}$$

e sull'asse delle ordinate il valore che si ricava applicando l'espressione (2.5) (cioè la DFT) moltiplicato per un fattore di scala pari a **1/N**.
Nel caso di un segnale a forma di impulso questo fattore di scala è uguale a **DT**.

Bisogna infine ricordare che il risultato di una FFT (cioè il suo output), sia che venga applicata ad una funzione definita in campo reale che ad una definita in campo complesso, è una funzione complessa, avente cioè una parte reale ed una parte immaginaria.

LabVIEW mette a disposizione del programmatore due VI, che calcolano la DFT/FFT di un segnale:

- Real FFT.VI
- Complex FFT.VI

La differenza tra i due VI consiste nel fatto che Real FFT.VI calcola la FFT di un segnale avente valori definiti in campo reale, mentre Complex FFT.VI calcola la FFT di un segnale avente valori definiti in campo complesso.

I fenomeni che si verificano nella realtà solitamente generano segnali reali; pertanto nella maggior parte delle applicazioni è possibile utilizzare la Real FFT.VI.

Quando il segnale è costituito sia da una parte reale che da una parte immaginaria si deve utilizzare Complex FFT.VI: tale tipo di segnale è molto frequente nel campo delle telecomunicazioni. Passiamo ad esaminare in dettaglio un block diagram di un semplice programma che effettua la DFT di un segnale sinusoidale.

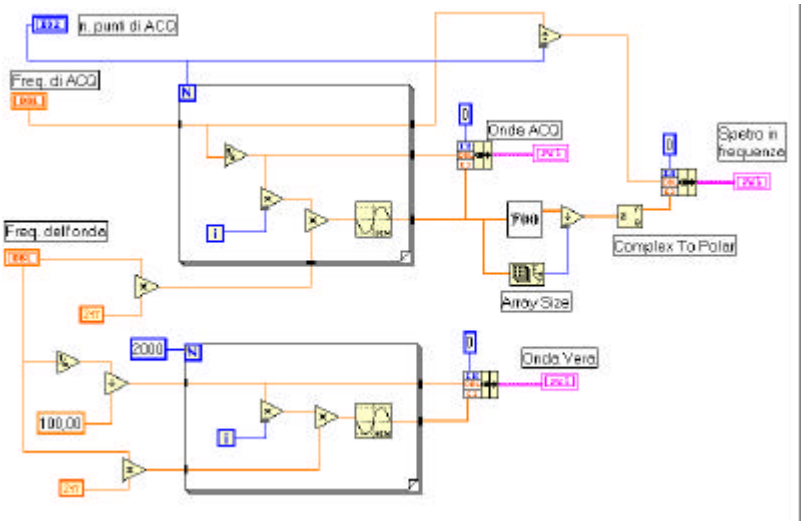


Fig.18.5 Schema del blok diagram di un programma che effettua la FFT di una funzione sinusoidale.

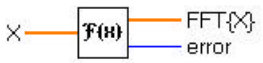
Ricordiamo che vale la relazione seguente:

$$y(t_i) = \sin(2\neq f\ t_i) = \sin(2\neq f\ i\ \Delta T_{ACQ}) = \sin(2\neq f\ i\ f_{ACQ})$$

ove:

- **f** è la frequenza della sinusoide
- $\mathbf{t_i}$ l'istante in cui se ne calcola il valore
- $\mathbf{f_{ACQ}}$ la frequenza di acquisizione e ΔT_{ACQ} il corrispondente periodo
- **i** è il contatore del punto che si sta calcolando.

Una volta definita la frequenza della sinusoide da analizzare, un primo ciclo **for-loop** ne calcola il valore in N punti alla frequenza di acquisizione $\mathbf{f_{ACQ}}$ stabilita dall'utente ed un secondo ciclo suddivide il periodo della sinusoide in cento punti e calcola il valore in tali punti. Il primo ciclo simula un reale processo di acquisizione campionato mentre il secondo descrive con buona approssimazione l'onda vera. Tali cicli sono diagrammati sotto il nome di **onda acquisita** ed **onda vera**. Il vettore contenente l'onda acquisita viene elaborato mediante la funzione Real FFT che ne calcola la Fast Fourier Transforms; a tale funzione si accede mediante la **Functions palette** tramite la successione **Analysis** e **Signal Generation**.



Come spigato in precedenza, ogni elemento del vettore output della FFT deve essere moltiplicato per il fattore di scala 1/N in modo da ottenere la corretta ampiezza delle varie componenti della frequenza. Questa operazione viene effettuata mediante la funzione:

che calcola la dimensione del vettore in ingresso alla FFT. A tale funzione si accede mediante la **Functions palette** tramite la sub-palette **Array**. Dato che si intende solo studiare la parte reale della FFT, cioè la distribuzione in frequenza delle ampiezze, occorre individuare separatamente la parte reale e la parte immaginaria dell'output della FFT. Tale oparazione può essere effettuata mente la funzione **Complex to Polar** a cui si accede mediante la **Functions palette** tramite la successione **Numeric** e **Complex**.

Il diagramma dello spettro in frequenza del segnale riporta sull'asse delle ascisse la frequenza (il passo è pari alla frequenza di acquisizione divisa per il numero di punti acquisiti) e sull'asse delle ordinate l'ampiezza scalata per il numero dei punti.

La Fig. 19.5 (caso A) riporta il relativo Control Panel. Questa figura illustra il caso in cui la **frequenza dell'onda sia pari a 25 Hz** e la **frequenza di acquisizione pari a 75 Hz**. Si vede chiaramente che nel diagramma che riporta l'onda acquisita il periodo di tale onda coincide perfettamente con quello dell'onda vera anche se la forma d'onda è lontana dall'originale. Lo spettro in frequenza mostra due picchi: uno a 25 Hz come prevedibile trattandosi di una sinusoide calcolata a quella frequenza, l'altro a 50 Hz. Questi 50 Hz possono essere visti come la frequenza in corrispondenza della quale si verifica il fenomeno dell'**aliasing**, come vedremo meglio in un esempio successivo.

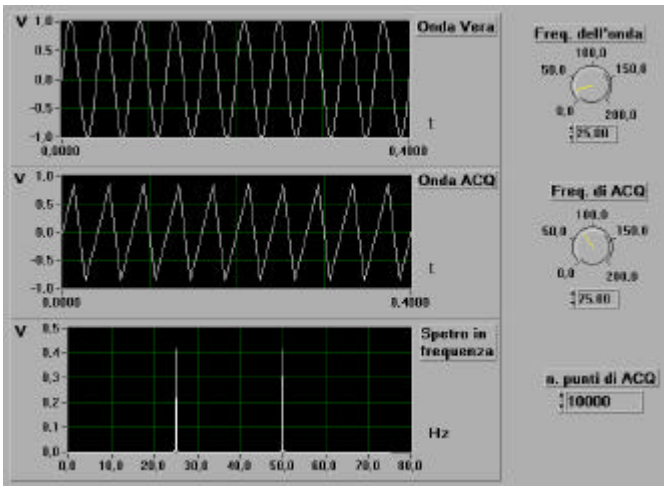
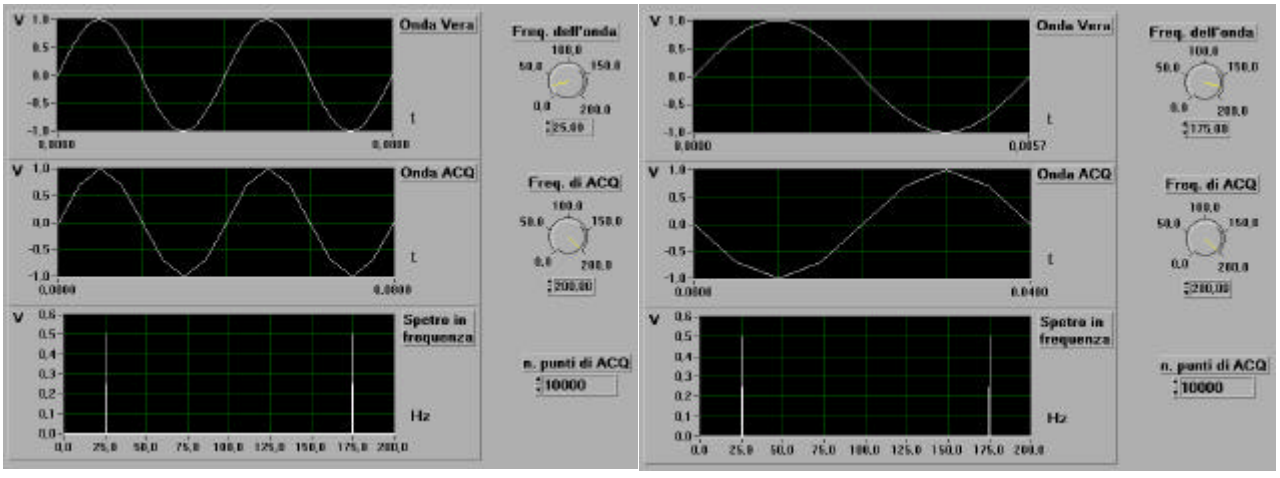


Fig.19.5 Schema del control panel di un programma che effettua la FFT di una funzione sinusoidale (Caso A)

In Fig. 20.5 sono riportato due control panel analoghi a quello di Fig. 19.5; il control panel contrassegnato dalla lettera (B) mostra che la frequenza di acquisizione è salita a 200 Hz. In questo caso la ricostruzione del segnale è molto migliorata; sullo spettro in frequenza si osserva sempre il picco a 25 Hz mentre la frequenza di aliasing si sposta a 175 Hz.



(B) (C)

Fig. 20.5 Schema del control panel di un programma che effettua la FFT di una funzione sinusoidale (Casi B e C)

La Fig. 20.5 (C) riporta il caso in cui la frequenza dell'onda è passata a 175 Hz (la frequenza di aliasing del caso B) mentre la frequenza di acquisizione si è mantenuta pari a 200 Hz. La figura mostra che la frequenza dell'onda acquisita è pari a 25 Hz: ecco un tipico esempio di frequenza di aliasing.
Infine in Fig. 21.5 è riportato un caso duale rispetto a quanto illustrato in Fig. 19.5. La frequenza di acquisizione è pari a 25 Hz con una frequenza dell'onda da acquisire pari a 75 Hz; siamo cioè in una condizione in cui la frequenza di campionamento $\mathbf{f_s}$ è molto inferiore alla frequenza del segnale da campionare (il teorema di Nyquist che dice che la frequenza di campionamento deve essere pari ad almeno il doppio della frequenza che si intende acquisire).
E' evidente anche in questo caso il fenomeno dell'aliasing: con questa frequenza di campionamento l'acquisizione ricostruisce un'onda di frequenza pari a 0,3 Hz mentre il fenomeno ha una frequenza pari a 75 Hz.

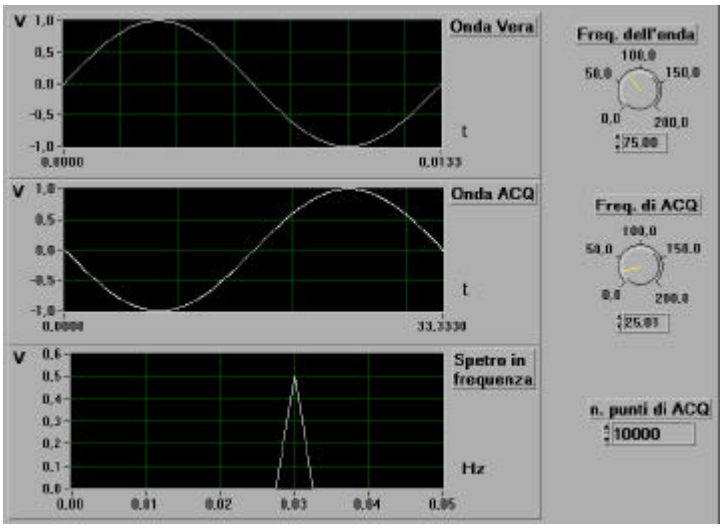


Fig.21.5 Schema del control panel di un programma che effettua la FFT di una funzione sinusoidale (caso c).

Queste breve descrizione deve mettere in guardia dal pericolo costituito dal sotto-campionamento.

Nelle figure 22.5 e 23.5 sono riportati il control panel ed il block diagram di un programma che effettua una reale acquisizione di dati e ne effettua la successiva analisi di Fourier mediante FFT.

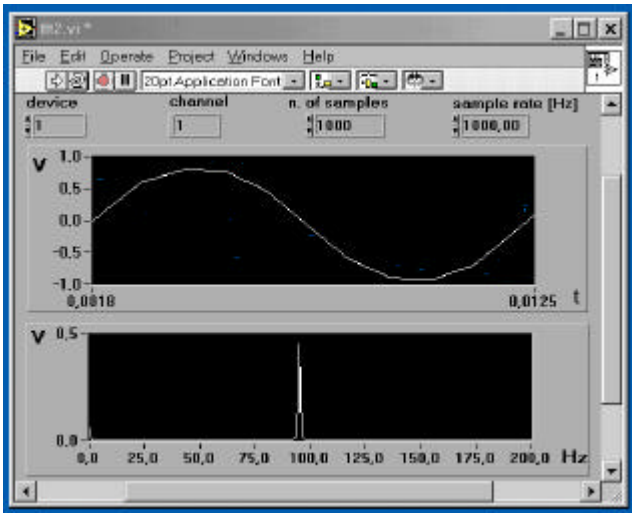


Fig.22.5 Schema del control panel di un programma che acquisisce dati ed effettua la FFT dei dati acquisiti.

La struttura è assolutamente simile a quanto descritto in precedenza eccezion fatta per il fatto che in questo caso si è utilizzato il VI che effettua l'acquisizione multi-point, già illustrato in precedenza.

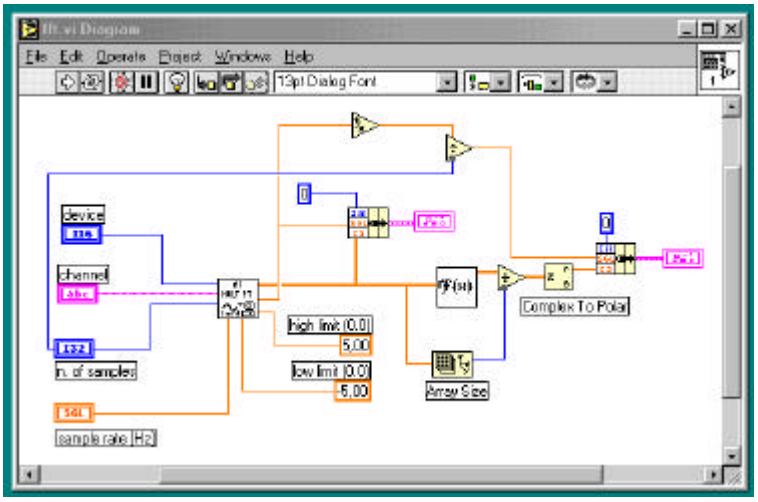


Fig.23.5 Schema del block diagram di un programma che acquisisce dati ed effettua la FFT dei dati acquisiti.

Comunicazione seriale RS-232C

La comunicazione seriale è assai diffusa per trasmettere dei dati tra un computer ed una periferica (un altro computer od uno strumento programmabile da remoto).

I vantaggi di questo protocollo di comunicazione sono legati essenzialmente al basso costo, alla semplicità del sistema nel suo complesso ed alla possibilità di comunicare a grandi distanze; tuttavia a questi vantaggi si contrappongono gli svantaggi dovuti alla complessità di programmazione dei protocolli ed alla bassa velocità di trasmissione. Quest'ultima limitazione è dovuta al fatto che la comunicazione seriale tra il trasmettitore ed il ricevitore avviene inviando i dati **un bit alla volta** attraverso un singola linea di comunicazione; nasce quindi immediata la limitazione dovuta al fatto che una porta seriale non può comunicare contemporaneamente con più strumenti.
La comunicazione seriale è molto diffusa anche perché la maggior parte dei personal computer hanno in dotazione una o due porte seriali.

Il cavo di connessione standard usa i classici connettori a 9 o a 25 pin. Nella maggior parte dei casi si utilizzano solo tre linee: una prima per trasmettere i dati (**transmit data**), una seconda per ricevere i dati (**receive data**) ed infine una terza (**ground**) che è la terra di riferimento; si è quindi in presenza di un classico collegamento di tipo single-ended, che è sensibile ai disturbi. Pertanto la massima distanza tra trasmettitore e ricevitore è di 100m, limite che in pratica è frequentemente violato.

Coloro che hanno familiarità con il modem non troveranno difficoltà nel riconoscere il significato dei vari parametri che contraddistinguono questo protocollo:

1. **port number**: numero della porta seriale (in genere 1 per COM1 e 2 per COM2);
2. **baud rate**: velocità di trasmissione espressa in bit al secondo (nell'esempio 38400 baud);
3. **data bits**: numero di bit che servono a costruire il singolo carattere da trasmettere;
4. **stop bit**: un bit che segue il carattere trasmesso ed avverte il ricevitore dell'arrivo del successivo;
5. **parity bit**: un bit trasmesso in coda al carattere, il quale riporta una condizione legata alla somma dei vari bit che costituiscono il carattere stesso. In generale vi sono due stati: odd (dispari) che controlla se il valore della somma è dispari, even (pari) che controlla se il valore della somma è pari. Come risulta evidente questo è un carattere di controllo.

Si accede alle funzioni che gestiscono la comunicazione seriale tramite il cammino **Functions**, **Instrument I/O**, **Serial** (Fig. 24.5).

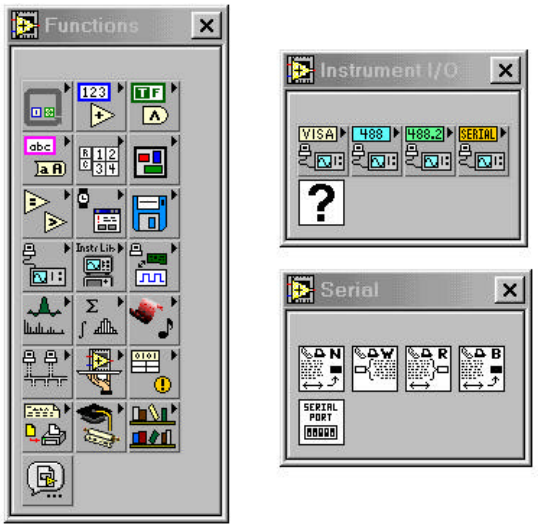


Fig. 24.5 Menu delle Functions, sottomenu degli Instrument I/O e dei Serial

Elenchiamo ora le principali funzioni, contenute nella libreria **Instrument I/O >> Serial**, che vengono utilizzate per la gestione delle porte seriali:

1. inizializzazione della porta seriale (**Serial Port Init**):
   - **flow control** è un cluster che definisce i parametri di handshaking; per default questa subroutine non usa hanshaking;
   - **buffer size** indica le dimensioni dei buffer di input ed output assegnati dal VI;
   - **port number** specifica la porta utilizzata per la comunicazione;
   - **baud rate**, **data bits** e **parity** stabiliscono i parametri di comunicazione per la porta selezionata.

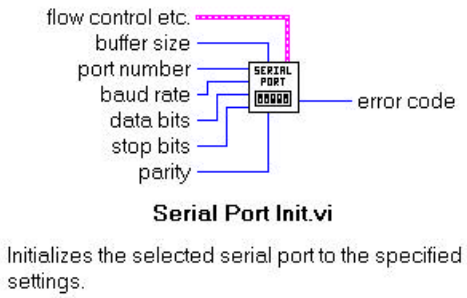


Fig. 25.5 Serial Port Init

2. scrittura sulla porta seriale (**Serial Port Write**):
   raccoglie i dati in stringhe che vengono scritte sulla porta seriale indicata da **port** number:

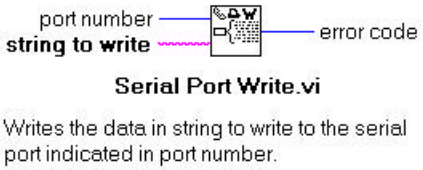


Fig. 26.5 Serial Port Write

3. lettura dalla porta seriale (**Serial Port Read**):
   legge dalla porta seriale indicata da **port number** il numero di caratteri specificati da **request byte count**.

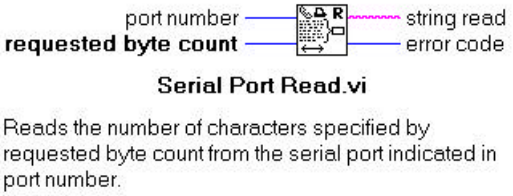


Fig. 27.5 Serial Port Read

Concludiamo questa sezione dedicata alle comunicazioni seriali con un semplice esempio di acquisizione dati mediante un sistema remoto. Il segnale in tensione viene da un dispositivo di conversione A/D intelligente **(GANTNER ISM112)** accessibile da remoto tramite un protocollo seriale RS-232.

Il codice è suddiviso nelle seguenti fasi:

1. apertura della porta seriale con specificazione dei parametri in ingresso;
2. scrittura sul modulo remoto di una stringa di caratteri contenente l'istruzione di trasferire sul buffer del modulo GANTNER ISM112 il contenuto della variabile 1 (contenete a sua volta il valore della tensione all'ingresso analogico 1). Tale stringa ha la seguente espressione:

   $ [indirizzo del modulo] R [numero della variabile] Cr

   ove:
   - l'indirizzo del modulo è uguale a **01**,
   - il numero della variabile è uguale a **1**,
   - Cr sta per **Carriage Return** e corrisponde al numero esadecimale **0D**.
3. fase di attesa per consentire l'effettuazione del comando,
4. lettura dal buffer di una stringa di 10 caratteri,
5. conversione di tale stringa di caratteri in un numero reale.

I punti da 2 a 5 sono iterati N volte; al termine del ciclo di iterazione tutti questi valori sono trasferiti alla funzione Standard Deviation.VI che ne calcola il valore medio e la deviazione standard.

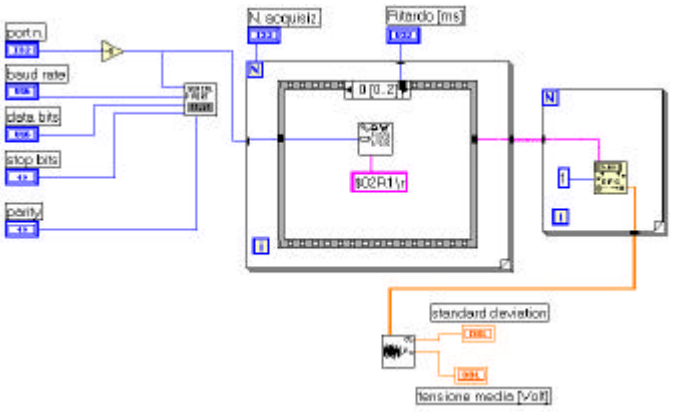


Fig. 28.5 Esempio di comunicazione seriale RS-232C (Block diagram – Prima Sequence)

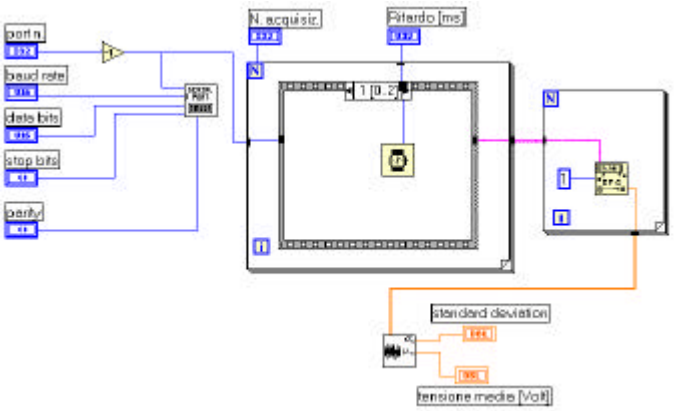


Fig. 29.5 Esempio di comunicazione seriale RS-232C (Block diagram – Seconda Sequence)

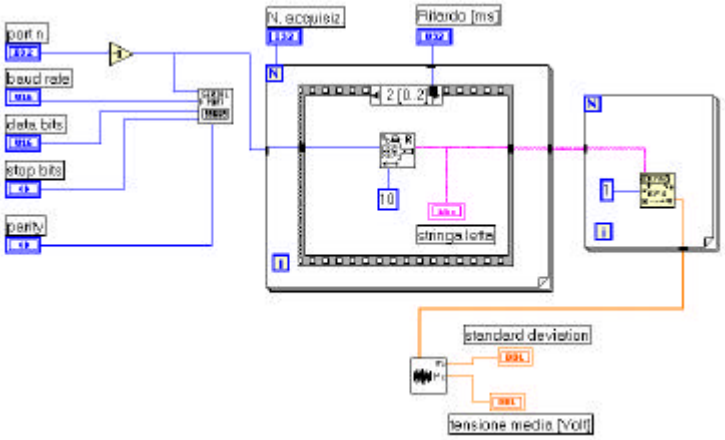


Fig. 30.5 Esempio di comunicazione seriale RS-232C (Block diagram – Terza Sequence)

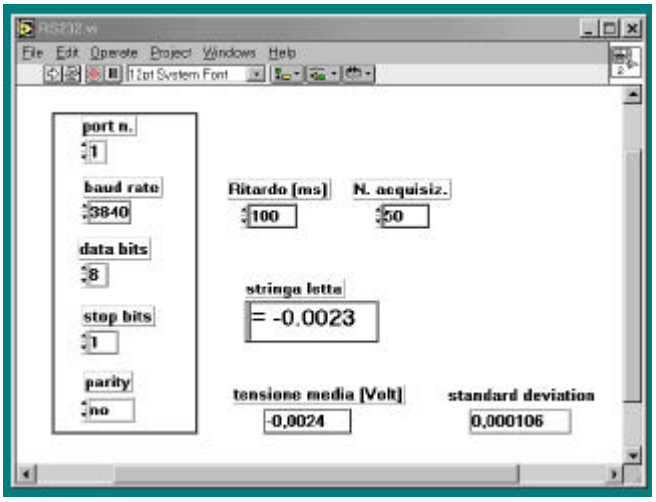


Fig. 31.5 Esempio di comunicazione seriale RS-232C (Control panel)

Nel codice sopra riportato merita di essere sottolineata la presenza di una funzione utilizzata per la prima volta: si tratta della funzione che converte una stringa di caratteri contenenti i numeri da 0 a 9, il carattere "+", il carattere "-", il carattere "e", il carattere "E" ed il carattere "." scritta in formato numerico in notazione ingegneristica , esponenziale, frazionale in un valore numerico a partire da un valore di offset predefinito. Nel presente esempio la stringa è del tipo " = ±b.bbbb " pertanto per trasformarla in un numero l'offset deve essere uguale ad 1 per scartare tra i caratteri il carattere "="

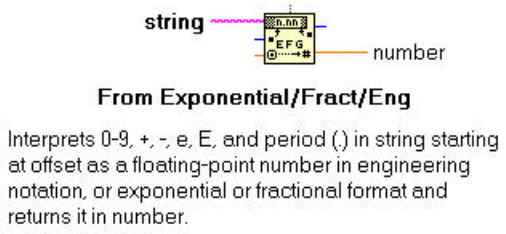


Fig. 31.5 Funzione per convertire una stringa di caratteri in un valore numerico