LabVIEW 8.2

Dispense delle Esercitazioni dei moduli:

Misure per la Bioingegneria e l'Habitat

e

Strumentazioni Elettroniche

A.A. 2008 – 2009

Ing. Massimo Piotto

Introduzione

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) è un ambiente di sviluppo per applicazioni orientate principalmente all'acquisizione di dati, alla gestione degli strumenti di misura e all'analisi ed elaborazione dei segnali. Introdotto nel 1987 dalla National Instruments, attualmente ne è stata distribuita la versione 8.6.

LAbVIEW fornisce un ambiente di programmazione basato su un linguaggio di tipo grafico ad oggetti, denominato linguaggio "G" (Graphical Language), il quale consente di realizzare programmi in forma di diagrammi a blocchi. Il linguaggio G conserva molte similitudini con gli ambienti di programmazione tradizionali, tra le quali:

- tipi di dati e operatori di uso comune;
- strutture di controllo del flusso di programma;
- metodi di debug;
- funzioni di libreria.

La differenza sostanziale tra il linguaggio G e quelli tradizionali risiede nel controllo del flusso di programma.

Nei linguaggi tradizionali di tipo testuale, l'ordine di esecuzione delle istruzioni che costituiscono il codice del programma è determinato dall'ordine in cui le istruzioni sono scritte all'interno del codice stesso.

Nel linguaggio G, l'ordine di esecuzione è stabilito dal "flusso di dati", ovvero ciascuna istruzione viene eseguita non appena sono disponibili i suoi dati di ingresso. In questo modo, è possibile eseguire operazioni in parallelo: il parallelismo è una delle proprietà peculiari di LabVIEW.

I programmi che vengono generati prendono il nome di "*strumenti virtuali*" (Virtual Instrument, VI): il termine "*strumenti*" è determinato dal fatto che durante l'esecuzione i programmi sviluppati presentano agli utenti un'interfaccia analoga a quella degli strumenti di misura, mentre il termine "*virtuali*" dipende dal fatto che l'interazione avviene con un programma in esecuzione e non con un dispositivo fisico reale.

Le istruzioni, definite in fase di stesura del codice mediante il linguaggio grafico, vengono tradotte in modo trasparente in linguaggio C e successivamente compilate. Inoltre, l'utente ha la possibilità di inserire direttamente parti di codice scritto in linguaggio C/C++.

Un programma VI è composto da tre parti fondamentali:

- Pannello frontale (Front Panel)

- Diagramma a blocchi funzionale (Block Diagram)
- Icona/Connettore (Icon/Connector)

Pannello Frontale

Il pannello frontale è l'interfaccia grafica che permette di definire ed introdurre tutte le grandezze di ingresso (input del programma) e tutte le grandezze in uscita (valori delle misure, risultati dei calcoli, grafici, ecc.). Il nome deriva dal fatto che le varie grandezze possono essere definite in modo tale che esso assuma l'aspetto di un pannello frontale di uno strumento dotato di display, indicatori, manopole, tasti ecc.

Nel pannello frontale vengono inserite le seguenti entità distinte:

- **Controllori**: sono una variabile di ingresso che può essere modificata dall'utente agendo sul pannello frontale;
- **Indicatori**: sono una variabile di uscita il cui valore è modificato dal programma in esecuzione e non dall'utente.

Il pannello frontale, pertanto, sarà costituito da un insieme di controllori e indicatori che definiscono l'interfaccia utente.

Diagramma a blocchi

Il diagramma a blocchi è lo strumento grafico che consente di scrivere il codice del programma. Si presenta sotto una forma che può ricordare a grandi linee lo schema di un circuito elettrico.

Nel diagramma a blocchi sono presenti le seguenti entità distinte:

- **Nodi**: sono gli elementi di elaborazione elementare che effettuano le varie operazioni richieste dal programma e appaiono sotto forma di icone;
- **Collegamenti**: appaiono come dei fili che uniscono i vari nodi e permettono lo scambio di informazione ovvero il flusso dei dati.

In pratica, il diagramma a blocchi sarà costituito da un insieme di nodi e collegamenti che definiscono il codice del programma.

Icona e Connettore

La coppia icona/connettore consente di trasformare un programma in un oggetto che può essere impiegato all'interno di un altro programma VI, diventando così un sub-VI.

L'icona è un simbolo grafico di piccole dimensioni che rappresenta sinteticamente il VI stesso e viene esposto nell'angolo in alto a destra del pannello frontale. Quando un VI

viene impiegato all'interno di un altro programma come sub-VI, esso appare nel nuovo diagramma a blocchi con l'icona che lo identifica. Per poterlo collegare all'interno del nuovo diagramma a blocchi, è necessario conoscere la corrispondenza tra le aree dell'icona e gli elementi di ingresso/uscita del pannello frontale associato al sub-VI. Questa corrispondenza è determinata dal connettore.

Pertanto, il connettore determina la corrispondenza tra i terminali presenti nell'icona e le variabili di ingresso e uscita del programma.

Costruzione di un programma VI

Per la costruzione di un programma sono necessari tre passi fondamentali:

- costruzione del pannello frontale;
- costruzione del diagramma a blocchi;
- definizione dell'icona e del connettore.

Per compiere queste operazioni LabVIEW mette a disposizione una serie di strumenti raccolti in tre palette:

- tools palette;
- controls palette;
- functions palette.

Tools palette

Figura 1: Tools palette

La "tools palette" può essere mostrata sia quando è attiva la finestra del pannello frontale sia quando è attiva la finestra del diagramma a blocchi. In questa palette sono disponibili vari tools che consentono di selezionare, spostare, editare, collegare i vari

oggetti nonché introdurre dei breaking point e delle sonde per agevolare il debug del programma.

	Automatic tools selection	Se abilitato, muovendo il cursore sopra un oggetto del pannello di controllo o del diagramma a blocchi, LabVIEW seleziona automaticamente il tool corretto
	Operating	Modifica il valore di un controllore o ne seleziona il testo al suo interno
	Positioning	Posiziona, ridimensiona e seleziona gli oggetti
	Labelling	Edita il testo e crea le etichette
	Wiring	Collega gli oggetti all'interno del diagramma a blocchi
	Object shortcut menu	Apre il menu dell'oggetto
	Scrolling	Scorre la finestra senza usare le barre di scorrimento
	Breaking point	Inserisce un breaking point nel diagramma a blocchi per interromperne l'esecuzione
	Probe	Inserisce una sonda in un collegamento del diagramma a blocchi per verificarne il valore
	Color Copying	Copia i colori
	Coloring	Imposta i colori di background e foreground del pannello frontale e degli oggetti

Controls palette

La "controls palette" può essere mostrata solo quando è attiva la finestra del pannello frontale. Essa mette a disposizione, suddivisi in una serie di categorie, una serie di controllori ed indicatori simili a quelli presenti in uno strumento reale (manopole, interruttori, indicatori analogici e digitali) nonché oggetti di tipo grafico per simulare il display di uno strumento.

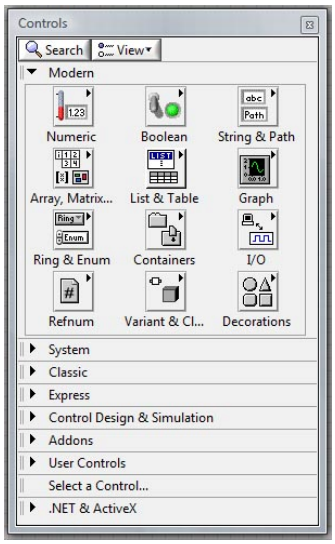


Figura 2: Controls palette con la categoria "Modern" espansa

Functions palette

La "functions palette" può essere visualizzata solo se è attiva la finestra del diagramma a blocchi. Essa mette a disposizione una serie di strutture e funzioni predefinite necessarie per la realizzazione del codice sorgente e suddivise in diverse categorie. Le varie funzioni, tra cui quelle matematiche, statistiche, logiche, acquisizione dati, colloquio con le periferiche, ecc., appaiono sotto forma di icona e vengono posizionate nel diagramma a blocchi mediante la tecnica del "drag and drop".

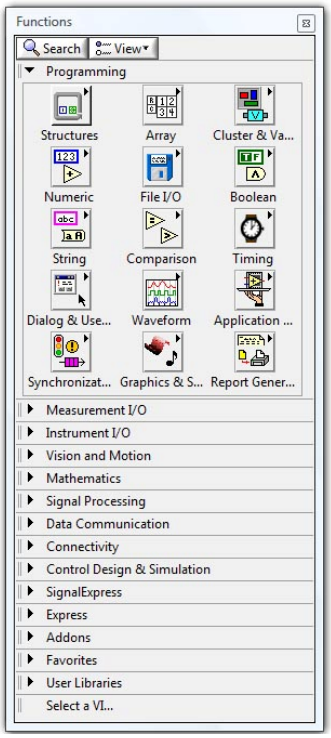


Figura 3: Functions palette con la categoria "Programming" espansa

Nella controls palette e nella functions palette sono presenti i seguenti due tasti per la “navigazione”:

	*Search*: consente di fare una ricerca testuale all’interno della palette
	*View*: consente di modificare la visualizzazione della palette

All’avvio di LabVIEW, compare la finestra di dialogo mostrata in figura 4 in cui è possibile scegliere tra aprire un programma già esistente o generare un nuovo programma. Premendo il tasto “Blank VI” LabVIEW apre un nuovo VI, denominato per default UNTITLED 1.VI e fa comparire sul video la finestra del pannello frontale e quella del diagramma a blocchi. La creazione del programma inizia con la “costruzione” del pannello frontale inserendo i vari controllori ed indicatori necessari, mediante la tools palette, la controls palette e la tecnica del “drag and drop”.

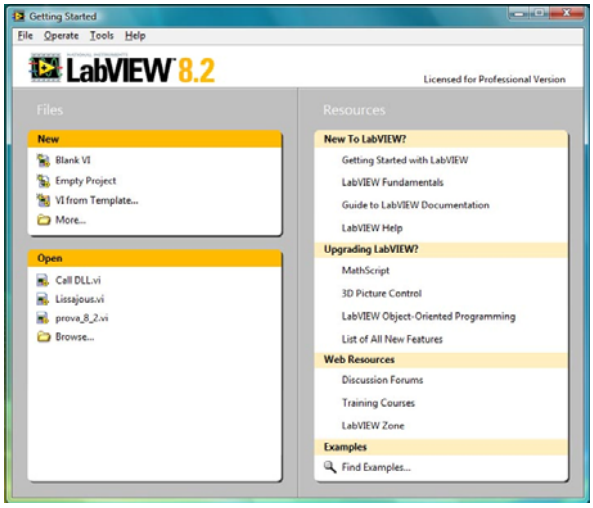


Figura 4: Finestra di avvio

Una volta completato il pannello frontale, si definisce la coppia icona/connettore. LabVIEW associa automaticamente un’icona e un connettore di default, i quali possono essere modificati. Inoltre, è necessario, mediante lo strumento wiring (rocchetto di filo), associare le varie zone del connettore ai controllori e indicatori introdotti nel front-panel.

Figura 5: Coppia Icona (sinistra) e Connettore (destra)

Terminata questa operazione, si può passare al diagramma a blocchi nel quale, per ogni controllore ed indicatore presente nel pannello frontale, appare un'icona ed un'etichetta. Le icone dei controllori presentano un bordo spesso mentre quelle degli indicatori hanno il bordo trasparente. In entrambi i casi, all'interno dell'icona è riportato il tipo di dati del controllore o dell'indicatore.

A questo punto, la costruzione del diagramma a blocchi prosegue con l'inserimento delle varie funzioni e strutture, e con il loro collegamento mediante lo strumento wiring ai controllori ed agli indicatori presenti. Nel caso in cui il collegamento risulti sbagliato, esso apparirà tratteggiato anziché continuo.

Terminato il diagramma a blocchi, il programma può essere messo in esecuzione mediante il tasto  presente nella toolbar sia del pannello frontale che del diagramma a blocchi. Durante l'esecuzione il tasto assume il seguente aspetto 

La medesima toolbar contiene anche i tasti di pausa  e di interruzione  del programma. La toolbar del diagramma a blocchi contiene inoltre i comandi per il debug quali l'esecuzione passo a passo del codice (), la visualizzazione del flusso dei dati () e il salvataggio del valore dei collegamenti in ogni momento del flusso di esecuzione () in modo tale da poter disporre, tramite una probe, del valore più recente del dato transitato attraverso il collegamento stesso. Nel caso in cui, finito il diagramma a blocchi, la freccia del tasto "run" appaia interrotta (), significa che nel programma sono presenti degli errori di scrittura del codice per cui non può essere mandato in esecuzione. Premendo il tasto compare una finestra di dialogo con la lista degli errori di scrittura del codice.

Il programma può essere salvato singolarmente oppure all'interno di una libreria creata per contenere i programmi relativi ad un argomento specifico.

LINGUAGGIO G

Il linguaggio su cui si basa la programmazione in LabVIEW viene denominato "linguaggio grafico" o "linguaggio G". Come già detto, questo linguaggio consente di comporre il codice del programma mediante l'uso di icone grafiche collegate tra loro da fili. Gli elementi che svolgono le operazioni elementari vengono detti nodi e LabVIEW tenta per quanto possibile di eseguire in parallelo tutti i nodi di programma. Tuttavia, un nodo può iniziare la propria attività solo se tutti i suoi elementi di ingresso contengono dei dati. Le comunicazioni tra i nodi, pertanto, determinano l'ordine nell'esecuzione degli stessi. Ad esempio, se due nodi sono interconnessi, quello che dei due fornisce dati di ingresso per l'altro viene eseguito per primo. Si ha, in questo modo, un flusso di programma determinato dal flusso dei dati. Questa è sostanzialmente la maggior differenza a livello di programmazione tra il linguaggio G ed i linguaggi testuali tradizionali. Infatti, per quanto riguarda i tipi di dati e le funzioni, il linguaggio G risulta simile agli altri linguaggi.

Vediamo ora in dettaglio gli elementi fondamentali di questo linguaggio.

Tipi di dati

Tra i tipi di dati resi disponibili dal programma ci sono:

- *numerici*: numeri interi con segno e senza segno a 8, 16, 32, 64 bit; numeri floating point e complessi a precisione singola (32 bit), doppia (64 bit) ed estesa (128 bit);
- *booleani*: true, false;
- *stringa*: dati in formato ASCII;
- *list and ring*: i controllori e gli indicatori di questo tipo consentono di associare ad un numero intero non negativo (0,1,2,…) una stringa. Se usati come controllori forniscono un menù di scelta a tendina e restituiscono in uscita il numero corrispondente alla stringa selezionata. Se usati come indicatori mostrano l'opzione sotto forma di stringa corrispondente al numero passato al loro ingresso. E' compito del programmatore gestire correttamente la corrispondenza tra numero e stringa. Nel caso delle liste è possibile determinare se è possibile fare scelte multiple da parte dell'utente;
- *enumerato*: è simile ai ring in quanto fornisce all'utente una lista di azioni possibili; in questo caso però il dato è costituito da una coppia di valori: una stringa e un numero. Il tipo enumerato è utile perché la manipolazione di numeri anziché di stringhe all'interno dello schema a blocchi è più semplice.

- *array*: è una collezione di dimensione variabile di elementi dello stesso tipo, i quali sono individuati da un indice compreso da 0 a (n-1) se n è il numero degli elementi dell'array: In genere, l'array viene manipolato come se si trattasse di una struttura dati atomica ovvero lungo il connettore connesso all'array fluiscono contemporaneamente tutti gli elementi dell'array stesso;
- *cluster*: è una collezione di dimensione variabile di dati anche di tipo diverso e l'ordine di inserimento all'interno del cluster ne determina anche il loro ordinamento logico.

Control	Indicator	Data Type	Use	Default Values
		Single-precision, floating-point numeric	Saves memory and does not overflow the range of the numbers.	0.0
		Double-precision, floating-point numeric	Is the default format for numeric objects.	0.0
		Extended-precision, floating-point numeric	Performs differently depending on the platform. Use only when necessary.	0.0
		Complex single-precision, floating-point numeric	Same as single-precision, floating-point, with a real and an imaginary part.	0.0 + i0.0
		Complex double-precision, floating-point numeric	Same as double-precision, floating-point, with a real and an imaginary part.	0.0 + i0.0
		Complex extended-precision, floating-point numeric	Same as extended-precision, floating-point, with a real and an imaginary part.	0.0 + i0.0
		8-bit signed integer numeric	Represents whole numbers and can be positive or negative.	0
		16-bit signed integer numeric	Same as above.	0
		32-bit signed integer numeric	Same as above.	0
		64-bit signed integer numeric	Same as above.	0
		8-bit unsigned integer numeric	Represents only non-negative integers and has a larger range of positive numbers than signed integers because the number of bits is the same for both representations.	0
		16-bit unsigned integer numeric	Same as above.	0
		32-bit unsigned integer numeric	Same as above.	0
		64-bit unsigned integer numeric	Same as above.	0
		<64.64>-bit time stamp	Stores absolute time with high precision.	12:00:00.000 AM 1/1/1904 (Universal Time)
		Enumerated type	Gives users a list of items from which to select.	—
		Boolean	Stores Boolean (TRUE/FALSE) values.	FALSE
		String	Provides a platform-independent format for information and data, which you can use to create simple text messages, pass and store numeric data, and so on.	empty string

		Array	Encloses the data type of its elements in square brackets and takes the color of that data type. As you add dimensions to the array, the brackets become thicker.	—
		A matrix of complex elements.	The wire pattern differs from that of an array of the same data type.	—
		A matrix of real elements.	The wire pattern differs from that of an array of the same data type.	—
		Cluster	Encloses several data types. Cluster data types appear brown if all elements in the cluster are numeric or pink if all elements of the cluster are of different types. Error code clusters appear dark yellow, while LabVIEW class clusters are crimson by default.	—
		Path	Stores the location of a file or directory using the standard syntax for the platform you are using.	empty path
		Dynamic	(Express VIs) Includes data associated with a signal and the attributes that provide information about the signal, such as the name of the signal or the date and time the data was acquired.	—
		Waveform	Carries the data, start time, and Δt of a waveform.	—
		Digital waveform	Carries start time, Δx, the digital data, and any attributes of a digital waveform.	—
		Digital	Encloses data associated with digital signals.	—
		Reference number (refnum)	Acts as a unique identifier for an object, such as a file, device, or network connection.	—
		Variant	Includes the control or indicator name, the data type information, and the data itself.	—
		I/O name	Passes resources you configure to I/O VIs to communicate with an instrument or a measurement device.	—
		Picture	Includes a set of drawing instructions for displaying pictures that can contain lines, circles, text, and other types of graphic shapes.	—

Tabella 1: Controllori e indicatori disponibili in LabVIEW

Strutture di controllo

Le strutture sono costrutti grafici che consentono di controllare il flusso di esecuzione. Ogni struttura è delimitata da un bordo esterno che può essere ridimensionato. Essa esegue, in una modalità che dipende dal tipo di struttura, la porzione di diagramma a blocchi inclusa entro il bordo che la delimita.

Tra le strutture fornite dal G language, quelle che analizzeremo nel corso di queste esercitazioni sono:

1. *for loop*: ripete l'esecuzione della porzione di codice in esso contenuta un determinato numero di volte. La struttura esegue il diagramma contenuto nel rettangolo che la delimita per i=0,1,….N-1.

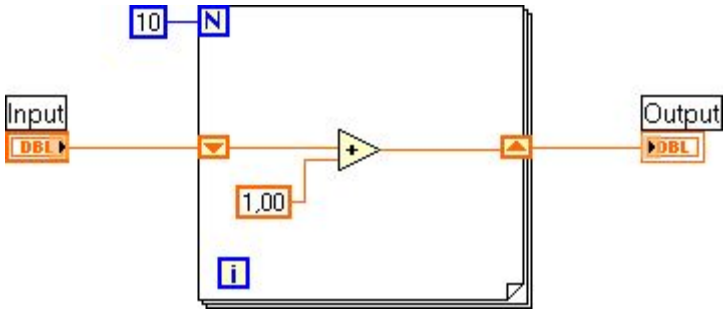


Figura 6: Esempio di struttura for

Per trasferire il valore assunto da una variabile da una iterazione a quella successiva si utilizzano gli shift register. Uno shift register è costituito da una coppia di terminali opposti posizionati sul bordo verticale della struttura: il terminale di destra memorizza il valore assunto al termine di una iterazione della variabile ad esso collegata, mentre quello di sinistra rende disponibile tale valore all'iterazione successiva. Ovviamente i due terminali dello shift register devono essere collegati allo stesso tipo di dati. L'esempio riportato in figura 6 esegue una porzione di codice che in linguaggio C potrebbe essere scritto:

```
shift_register = input;
for (i = 0; i < 10; i ++)
shift_register = shift_register +1;
output = shift_register;
```

Di solito è opportuno inizializzare lo shift register collegando una costante o un controllore al terminale posizionato sul bordo sinistro della struttura in modo da resettare il valore dello shift register alla prima iterazione del loop. Se lo shift register non viene inizializzato, il ciclo usa il valore scritto nello shift register durante l'ultima esecuzione del ciclo oppure utilizza il valore di default per il tipo di dati se il ciclo non è mai stato eseguito. È possibile definire degli "stacked shift register" che consentono di accedere ai dati di iterazioni precedenti del ciclo. Questi registri ricordano i valori da iterazioni multiple precedenti e li rendono disponibili a quelle successive. Nella "pila" di registri, quello posizionato più in alto conserva il dato più recente.

La struttura *for* consente di indicizzare ed accumulare automaticamente dei valori all'interno di array: questa proprietà prende il nome di "autoindexing". Quando si collega un array dall'esterno verso l'interno, i componenti del vettore sono assegnati alla porzione di codice interna al ciclo uno alla volta, ad iniziare

dal primo, uno per ogni iterazione. Viceversa, se uno scalare viene collegato dall'interno di un ciclo verso un vettore esterno, i valori dello scalare assunti a ciascuna iterazione vengono accumulati nel bordo esterno della struttura per poi essere memorizzati nel vettore di uscita al termine dell'intero ciclo.

Questa funzione di autoindexing è abilitata per default nel ciclo *for*: per disabilitarla è necessario visualizzare il menù di pop-up con il tasto destro del mouse sul collegamento in entrata e/o in uscita dal ciclo for (denominato "tunnel") e scegliere "disable indexing".

2. *while loop*: ripete l'esecuzione della porzione di codice in esso contenuta fintanto che non si verifica una determinata condizione; ad ogni ciclo incrementa di 1 una variabile di conteggio inizialmente inizializzata a 0.

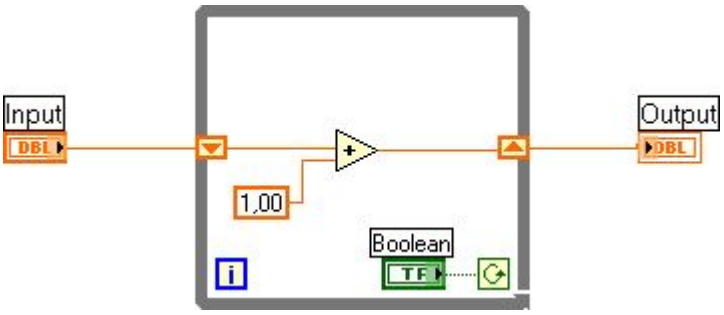


Figura 7: Esempio di struttura while

Anche per il ciclo *while* è possibile definire degli shift-register mentre la funzione autoindexing è per default disabilitata: per abilitarla è necessario usare il menù di pop-up dei collegamenti.

E' importante notare che la variabile booleana collegata al terminale di condizione deve essere all'interno della struttura in modo che il suo valore venga controllato ad ogni iterazione. Se viene posizionata esternamente, il suo valore viene controllato solo all'inizio del ciclo e quindi il ciclo stesso viene eseguito una volta sola (se la condizione di blocco è soddisfatta) o infinite volte (se la condizione di blocco non è soddisfatta).

3. *case*: la struttura case permette l'elaborazione condizionata di sub-diagrammi a seconda del valore assunto da una variabile di controllo. La variabile di controllo può essere un numero intero, un booleano, una stringa oppure un dato di tipo

enumerato. Il numero di sub-diagrammi dipende dal tipo di variabile di controllo e solo uno dei sub-diagrammi è visibile volta per volta. Se la variabile è di tipo booleano, i sub-diagrammi sono due: uno per il valore “true” e uno per il valore “false”. In questo caso, si ha un comportamento analogo al comando “ if…then….else” dei linguaggi testuali. È possibile definire un caso di default per trattare i valori fuori dall’intervallo specificato. Gli ingressi sono disponibili per tutte le condizioni previste anche se alcune di esse possono non averne bisogno; le uscite, invece, devono essere definite per ogni condizione prevista oppure si può specificare che venga usato il valore di default per quel tipo di dati.

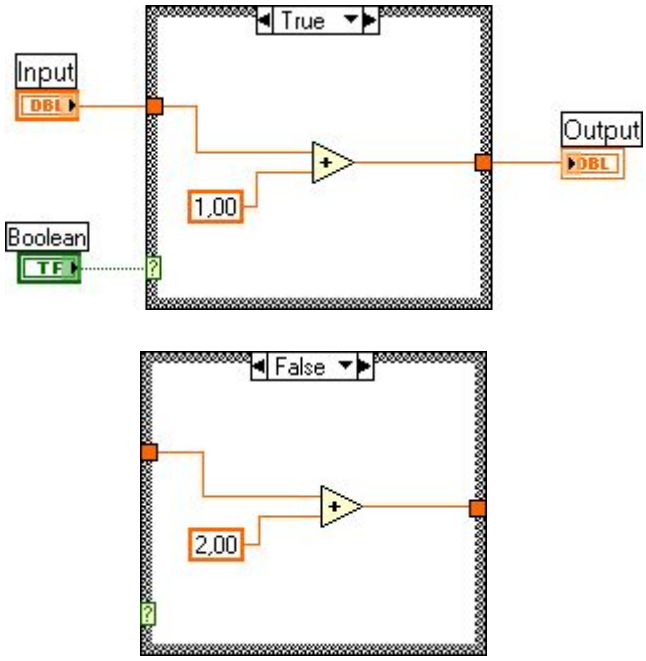


Figura 8: Esempio di struttura case con variabile di controllo di tipo booleano (per esigenze didattiche entrambi i sub-diagrammi sono mostrati)

4. *Sequence*: la struttura sequence permette l’elaborazione di uno o più sub-diagrammi in ordine sequenzale. I vari sub-diagrammi vengono chiamati “frame” e appaiono come i fotogrammi di una pellicola. LabVIEW mette a disposizione due diverse tipologie di strutture sequence:
   - “flat sequence”: i fotogrammi sono affiancati l’uno all’altro e vengono eseguiti in sequenza temporale da sinistra a destra e ciascun frame può essere eseguito se i suoi ingressi sono disponibili; le uscite di ciascun frame sono disponibili appena l’esecuzione del frame stesso si è conclusa;

- “stacked sequence”: i fotogrammi sono impilati e ciascuno è individuato dal proprio numero d’ordine a partire da zero. I frame sono eseguiti in sequenza dal primo e l’esecuzione della struttura può iniziare solo quando tutti gli ingressi necessari a ciascun frame sono disponibili; le uscite dei vari frame saranno disponibili solo quando l’esecuzione di tutta la sequence si sarà conclusa. Per rendere disponibile su più “frame” il valore assunto da una variabile appartenente ad un determinato frame, è necessario usare la “local sequence”. Scegliendo l’opzione “add local sequence” del menù pop-up della sequence, comparirà sul bordo del frame corrente un quadrato vuoto: una volta collegato un filo tra tale simbolo e una variabile, all’interno del quadrato comparirà una freccia rivolta verso l’esterno mentre in tutti i frame successivi la freccia sarà rivolta verso l’interno.

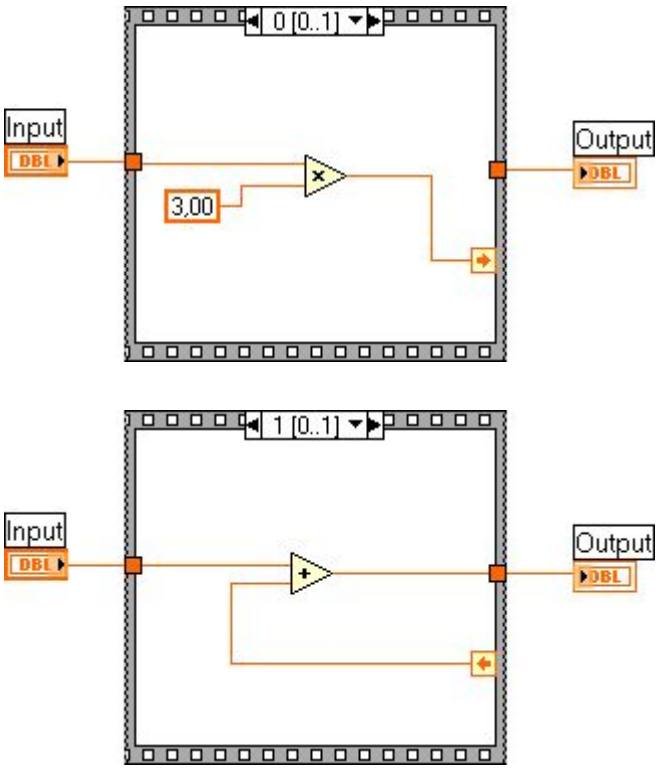


Figura 9: Esempio di struttura “stacked sequence” con due frame (per esigenze didattiche entrambi i frame sono mostrati)

Le sequence flat consentono di evitare l’uso delle “local sequence” e di poter usufruire dei dati all’esterno appena disponibili; per contro, occupano molto spazio nel diagramma a blocchi e, quindi, sono da evitare quando la sequenza prevede molti frame.

Le strutture sequence devono essere usate con una certa attenzione, in quanto interrompono il naturale parallelismo di LabVIEW imponendo una esecuzione sequenziale del programma. Interrompono, di fatto, quello che in LabVIEW viene definito il “naturale flusso dei dati”. Tuttavia, esse risultano molto utili nel caso si voglia forzare LabVIEW ad eseguire certe operazioni in una opportuna sequenza.

5. *Formula node*: la struttura formula node permette di scrivere al suo interno una serie di formule matematiche ed espressioni aritmetiche separate dal carattere “;”. La sintassi usata per la definizione delle espressioni è la stessa di molti linguaggi di programmazione testuali; inoltre, possono essere inseriti dei commenti racchiusi dalla coppia di delimitatori “/*…*/”. Le variabili di ingresso e di uscita sono definite mediante operazioni di pop-up sul bordo della struttura stessa.

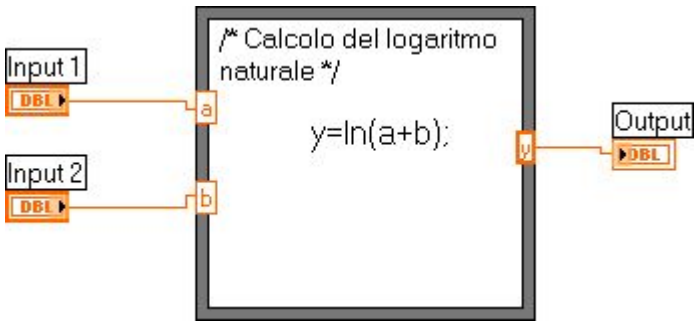


Figura 10: Esempio di struttura formula node

VARIABILI LOCALI E VARIABILI GLOBALI

LabVIEW associa ad ogni controllore o indicatore presente nel pannello frontale un terminale nel diagramma a blocchi. In alcuni casi, si ha la necessità di disporre del valore di un indicatore e/o di un controllore in vari punti del diagramma a blocchi che non sono raggiungibili direttamente attraverso il wiring normale. Per soddisfare questa necessità, si possono creare delle “variabili locali” che consentono di compiere operazioni di lettura o scrittura in un qualunque punto di un diagramma a blocchi del valore associato ad un controllore o ad un indicatore. Scrivere un valore all’interno di una variabile locale equivale ad inserire il valore direttamente dal pannello frontale. Ad ogni elemento del pannello frontale è possibile associare un numero qualsiasi di variabili locali le quali assumeranno

come nome quello del controllore o dell'indicatore cui fanno riferimento e che possono essere o di lettura o di scrittura.

Un uso comune delle variabili locali è quello all'interno di cicli for o while in modo da consentire che l'informazione sia disponibile sia all'interno sia all'esterno dei cicli stessi.

Nel caso in cui si abbia la necessità di condividere dell'informazione tra due o più programmi VI in esecuzione contemporaneamente si può definire una "variabile globale". Essa si configura come un pannello frontale, senza diagramma a blocchi, nel quale devono essere riportati gli indicatori e i controllori che si intende condividere tra i vari programmi VI.

L'uso delle variabili locali e globali deve avvenire con una certa cautela per evitare operazioni contemporanee di lettura e scrittura da parti diverse del diagramma a blocchi, nel caso di variabili locali, o da programmi diversi, nel caso di variabili globali. In genere è meglio evitare che variabili associate allo stesso indicatore o controllore siano in certi casi di lettura e in altri casi di scrittura.

PROPERTY NODE

Un "property node" consente di modificare durante l'esecuzione del programma gli attributi di un controllore o di un indicatore. Ad esempio, è possibile in run-time modificare il colore di un oggetto, renderlo invisibile, farlo lampeggiare, ecc. E' possibile scegliere se leggere l'attributo dalla variabile o scriverlo all'interno della stessa. Particolare interesse assumono i property node quando si riferiscono a grafici o diagrammi in quanto consentono di effettuare modifiche in run-time di alcuni parametri quali ad esempio i limiti degli assi delle ascisse e delle ordinate.

POLIMORFISMO

Il polimorfismo è la proprietà di un nodo di un diagramma a blocchi di adattarsi automaticamente a dati aventi rappresentazioni differenti, appartenenti a tipi o strutture diverse. Ciò significa che è, ad esempio, possibile sommare tra loro gli elementi corrispondenti di due array collegando quest'ultimi all'ingresso del nodo "somma" o ancora è possibile sommare un dato scalare a tutti gli elementi di un array collegando all'ingresso del nodo "somma" lo scalare e l'array.

RAPPRESENTAZIONE GRAFICA DEI DATI

I dati di uscita di un programma possono essere rappresentati in forma grafica in modo da avere una funzionalità simile a quella di un display degli strumenti di misura.

Il linguaggio G mette a disposizione due tipologie diverse di rappresentazioni grafiche:

- *Chart* (diagramma)**:** visualizzano i dati "punto a punto" nel senso che ogni volta che giunge un nuovo punto esso viene visualizzato sul display;
- *Graph* (grafici): visualizzano i dati a partire da un vettore in un unico momento.

  Tra le tipologie messe a disposizione di LabVIEW, consideriamo le tre più utilizzate:
  - *Waveform chart***:** riceve in ingresso singoli punti o blocchi parziali di punti raggruppati in un array. In questo modo, è possibile vedere il valore attuale di una variabile e confrontarlo con quelli precedenti. Per ogni nuovo punto, il programma incrementa il valore dell'asse x di 1 partendo da 0. È possibile, inoltre, collegare un dato di tipo waveform che contiene i dati, l'istante iniziale e il valore delta t per visualizzare i dati in funzione dell'istante di acquisizione. Per ottenere una chart multi traccia, è necessario collegare i singoli punti appartenenti alle diverse tracce in un cluster con la funzione "bundle"; per passare un blocco di punti per più tracce è necessario fare un array di cluster.
  - *Waveform graph*: consente di tracciare un grafico collegando al suo ingresso un array; LabVIEW riporta in ordinata l'ampiezza dei singoli dati del vettore in funzione dell'indice dell'array. I dati risultano, pertanto, riportati sul grafico equispaziati con l'asse x che parte da zero e viene incrementato di uno per ogni punto. È possibile collegare un cluster costituito da due scalari e un array: i due scalari indicano il valore iniziale di x e il valore di delta x, rispettivamente, mentre l'array contiene i valori delle ordinate. È possibile, inoltre, collegare un dato di tipo waveform che contiene i dati, l'istante iniziale e il valore delta t per visualizzare i dati in funzione dell'istante di acquisizione. Per ottenere un graph multi traccia è necessario generare un array 2D, dove ogni riga è una singola traccia, mediante la funzione build array se le tracce sono tutte delle stesse dimensioni; altrimenti si deve generare un array di cluster dove ogni cluster contiene un array 1D di dimensioni diverse.
  - *XY graph***:** accettano in ingresso un cluster formato da due array: uno per l'asse x e uno per l'asse y. In questo modo, è possibile rappresentare qualunque tipo di dati, anche non equispaziati. Per ottenere un grafico multi traccia è

necessario costruire un array, in cui ogni elemento è un cluster contenente l'array x e l'array y.

Nella tabella successiva sono riassunte le modalità principali per avere grafici a singola e a multi traccia.

TIPO	*SINGOLA TRACCIA*	*MULTI TRACCIA*
WAVEFORM CHART	SINGOLO PUNTO	CLUSTER DI PUNTI
WAVEFORM GRAPH	ARRAY 1D	ARRAY 2D (STESSA DIMENS.) ARRAY DI CLUSTER
XY GRAPH	CLUSTER DI 2 ARRAY 1D (X,Y)	ARRAY 1D DI CLUSTER

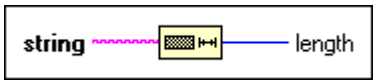
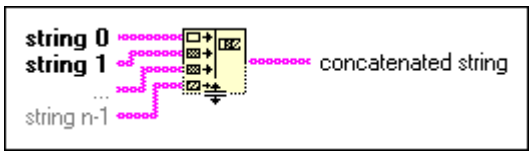
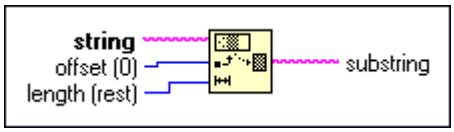
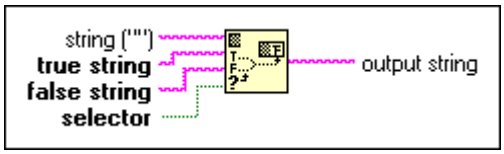
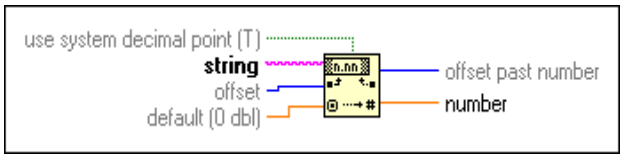
PRINCIPALI FUNZIONI DI LIBRERIA

LabVIEW mette a disposizione dell'utente molteplici funzioni di base che consentono:

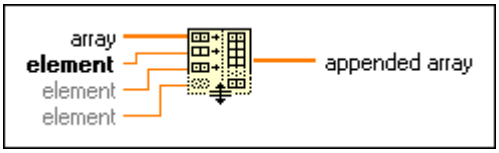
- di effettuare elaborazioni su dati numerici, stringhe, vettori;
- di gestire strumenti collegati ad un bus 488;
- di gestire una comunicazione seriale o attraverso Internet.

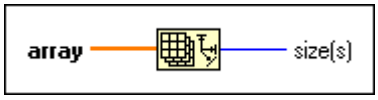
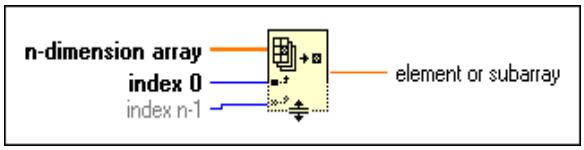
Tutte le funzioni sono raccolte nella *"functions palette"* disponibile nel diagramma a blocchi. Illustriamo brevemente quelle di maggiore interesse per queste esercitazioni:

- *Funzioni numeriche:* sono le funzioni che permettono di effettuare operazioni aritmetiche elementari, operazioni logaritmiche, trigonometriche e conversioni numeriche. Inoltre, sono disponibili le principali costanti matematiche fondamentali.
- *Funzioni per stringhe:* queste funzioni consentono di effettuare vari tipi di elaborazione sulle stringhe come la conversione da stringa a valore numerico e viceversa oppure la concatenazione di più stringhe in una unica. Dal momento che i comandi per gestire in remoto gli strumenti di misura devono essere inviati come stringhe, tali funzioni risultano essere rilevanti. Nella tabella successiva sono riportate le funzioni più utilizzate.

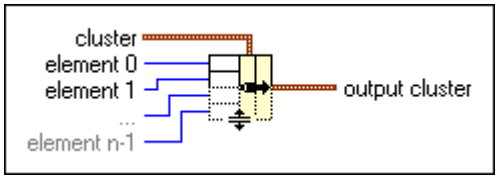
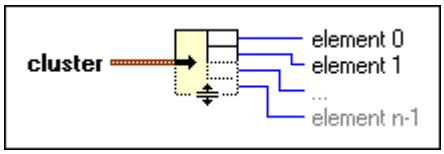
	*String length*: fornisce in uscita la lunghezza della stringa di ingresso pensata come vettore di byte;
	*Concatenate strings*: concatena le stringhe di ingresso in un'unica stringa di uscita;
	*String subset*: restituisce la sotto-stringa ottenuta dalla stringa di ingresso a partire da "offset" e contenente un numero di caratteri pari al valore specificato in "lenght"
	*Append true/false string:* concatena la stringa collegata al terminale "string" con la stringa collegata al terminale "false string" o con quella collegata al terminale "true string" in base al valore assunto dalla variabile booleana "selector".
	*Fract/exp string to number*: interpreta i caratteri 0..9, +, -, e, E e il punto decimale presenti nella stringa a partire da offset e li trasforma in un numero

- *Funzioni booleane:* queste funzioni permettono di effetturare differenti tipi di elaborazioni su variabili di tipo booleano come AND, OR, NOT, XOR, ecc.
- *Funzioni per gli array:* tali funzioni consentono di effettuare elaborazione sugli array quali la ricerca di un elemento dato un indice, la ricerca dell'elemento massimo oppure la conversione di un array in cluster e viceversa. Quelle più usate sono:

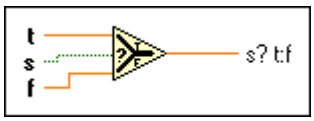
	*Build array*: concatena qualsiasi numero di array o di singoli elementi collegati all'ingresso nell'ordine prestabilito dal collegamento –

	dall'alto verso il basso – e fornisce in uscita un singolo array;
	*Array size*: fornisce il numero degli elementi dell'array;
	*Index array*: fornisce in uscita l'elemento dell'array collocato nella posizione indicata da index. Nel caso in cui l'array ha più dimensioni è necessario indicare un ulteriore indice per ogni dimensione aggiuntiva.

- *Funzioni per i cluster*: queste funzioni consentono di costruire un cluster una volta dati i suoi elementi o di scomporre un cluster nei suoi costituenti. Ricordiamo:

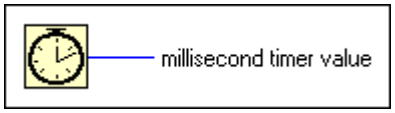
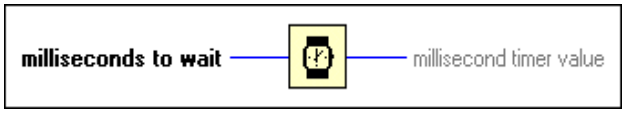
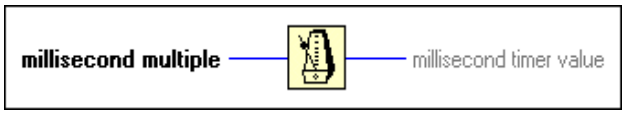
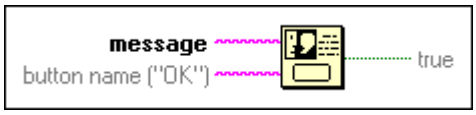
	*Bundle*: riunisce i componenti in ingresso in un cluster;
	*Unbundle*: scompone un cluster nei suoi componenti.

- *Funzioni per i confronti:* consentono di confrontare quantità di ingresso che, a seconda dei casi, possono essere di tipo numerico stringa, array o cluster. Oltre a quelle usualmente note citiamo:

	*Select*: restituisce il valore connesso all'ingresso "t" oppure quello connesso all'ingresso "f" a seconda che il selettore booleano collegato a "s" valga,

	rispettivamente, “true” o “false”.

- *Funzioni di tempo e di dialogo:* consentono di misurare intervalli di tempo, di sospendere le operazioni per un periodo di tempo oppure di inviare messaggi all’utente. Analizziamo:

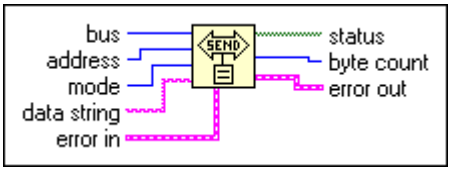
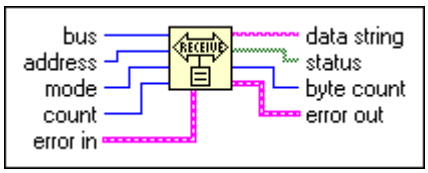
	*Tick count*: restituisce il valore dell’orologio di sistema espresso in millisecondi (ms);
	*Wait*: sospende l’esecuzione per un numero di millisecondi pari al valore specificato in ingresso;
	*Wait until next ms multiple*: sospende l’esecuzione finché il valore dell’orologio di sistema diventa un multiplo del valore di ingresso. E’ impiegata solitamente per sincronizzare degli eventi o i cicli di un loop;
	*One button dialog box*: visualizza nel pannello frontale una finestra di dialogo che contiene un messaggio e un bottone.

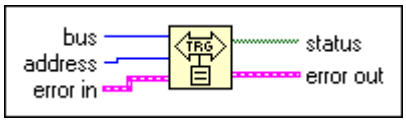
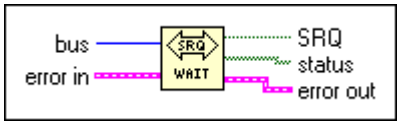
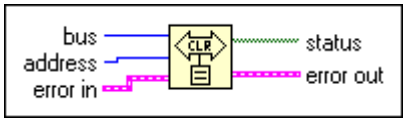
- *Funzioni di instrument I/O:* gestiscono le comunicazioni tra personal computer e strumenti di misura attraverso vari tipi di protocolli (VISA, VXI, RS232, IEEE488). Quelle che ci interessano maggiormente sono quelle IEEE488 raccolte nella libreria 488 e 488.2. Quest’ultime, in genere, sono quelle più utilizzate in quanto permettono di pilotare gli strumenti in modo conforme allo standard IEEE488.2. Queste funzioni possono essere suddivise in quattro categorie funzionali:
    - Single device;
    - Multiple device;
    - Bus management;

- Low level.

Le funzioni richiedono in ingresso l'indirizzo del dispositivo su cui eseguire l'operazione richiesta; generalmente, si riporta l'indirizzo primario; nel caso in cui si abbia anche l'indirizzo secondario si deve impiegare la funzione "makeaddr" per porre l'indirizzo primario e quello secondario nel formato corretto. L'indirizzo primario di default della scheda inserita nel computer è 0 senza indirizzo secondario. Viene richiesta, inoltre, l'indirizzo del bus che per default è 0.

Le funzioni maggiormente utilizzate sono:

	*Send*: permette di inviare byte ad un dispositivo connesso sul bus. Al terminale "bus" si deve indicare l'indirizzo dell'interfaccia che diventa talker e in "address" l'indirizzo del dispositivo che diventa listener. Al terminale "mode" si deve indicare il modo di segnalazione della terminazione dei dati al listener; è un intero che può assumere i seguenti valori: • 0: nessun terminatore finale; • 1: imposta ad 1 la linea EOI e spedisce NL (line feed); • 2: imposta ad 1 la linea EOI. Al terminale "data string" si collega la stringa contenente i dati da inviare.
	*Receive*: permette di leggere i byte trasmessi da un dispositivo connesso al bus. In questo caso, il terminale "bus" è il listener e il terminale "address" è il talker. Al terminale "mode" si collega un intero senza segno a 16 bit che indica il carattere di fine messaggio. Per valori da 0 a 255 indica il codice ASCII del carattere; se vale 256 oppure non si collega nulla la trasmissione termina quando viene posta al valore logico 1 la linea EOI. In "count" si deve specificare il numero massimo di byte da ricevere. Pertanto, la ricezione dei dati termina quando si verifica una delle seguenti condizioni: • È stato letto il numero di byte indicato in count

	• Si è verificato un errore; • Si è andati in time out; • La linea EOI vale 1; • Si è letto il carattere specificato in mode.
	*Trg*: consente di inviare un "trigger" software ad un dispositivo per provocare l'esecuzione di qualche operazione. Richiede l'indirizzo del bus e quello del dispositivo. Il comando risulta molto utile nel pilotaggio degli strumenti in quanto consente di dare inizio alle misure in un istante preciso.
	*Wait SRQ*: interrompe l'esecuzione del programma fino a quando un dispositivo collegato sul bus invia una richiesta di servizio (SRQ). Questo comando può essere impiegato, ad esempio, per aspettare che uno strumento abbia il dato pronto prima di effettuare la lettura.
	*Clr*: invia il messaggio "clear" ad uno strumento consentendo di resettare il contenuto dei registri interni.