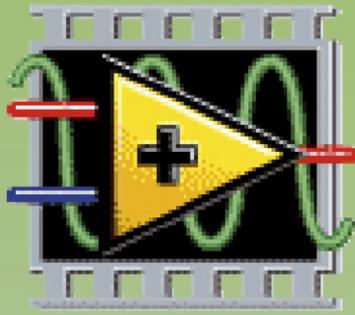


Corso

LabVIEW



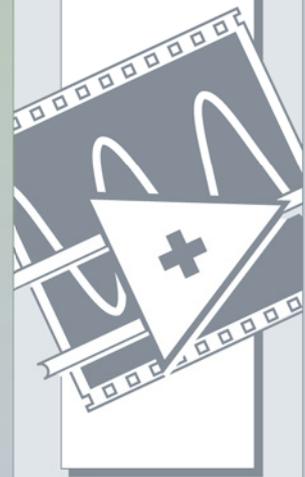
a cura di
FRANCESCO
FICILI



LabVIEW

Elettronica In
oltre l'elettronica

N° 144 - 152



Sommario

LEZIONI

1

LEZIONE 1

Iniziamo a conoscere la piattaforma software LabVIEW di National Instruments, un innovativo ambiente di sviluppo software che tramite il linguaggio-G permette di realizzare rapidamente applicazioni per l'acquisizione dati ed il controllo remoto di strumentazione hardware. Prima puntata.

9

LEZIONE 2

Entriamo nel vivo della programmazione imparando ad utilizzare le strutture di controllo, le strutture dati e i primi oggetti grafici. Proveremo anche a realizzare un primo esempio di programma. Seconda puntata.

17

LEZIONE 3

Continuiamo la descrizione delle strutture di controllo della piattaforma software LabVIEW, delle strutture dati e degli oggetti grafici. Terza Puntata.

25

LEZIONE 4

Proseguiamo la descrizione della piattaforma software LabVIEW analizzando gli oggetti grafici, i front panel, oltre al salvataggio e alla lettura dei file. Quarta Puntata.

33

LEZIONE 5

Studiamo una delle caratteristiche più apprezzate della piattaforma LabVIEW: l'interfacciamento ed il controllo di strumentazione esterna. Scopriremo il DAQ Assistant ed altro ancora. Quinta Puntata.

41

LEZIONE 6

Completiamo il programma di esempio riguardante l'acquisizione di segnali mediante la scheda USB-6008 di National Instruments: stavolta aggiungiamo un Tab Control per la gestione degli ingressi analogici e delle uscite digitali. Sesta Puntata.

49

LEZIONE 7

Dotiamo il programma che abbiamo creato per acquisire i dati di un'interfaccia per salvare in un file di testo i risultati dell'acquisizione mediante l'unità DAQ. Settima Puntata.

55

LEZIONE 8

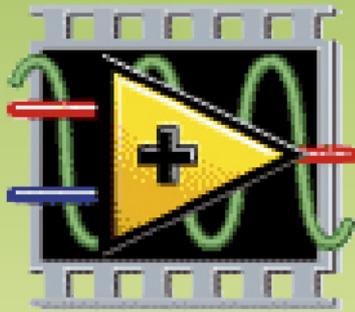
Concludiamo il corso dedicato a LabVIEW spiegando come generare report testuali e grafici tramite l'uso del Linguaggio-G; faremo anche una rapida panoramica sui toolkit disponibili.

63

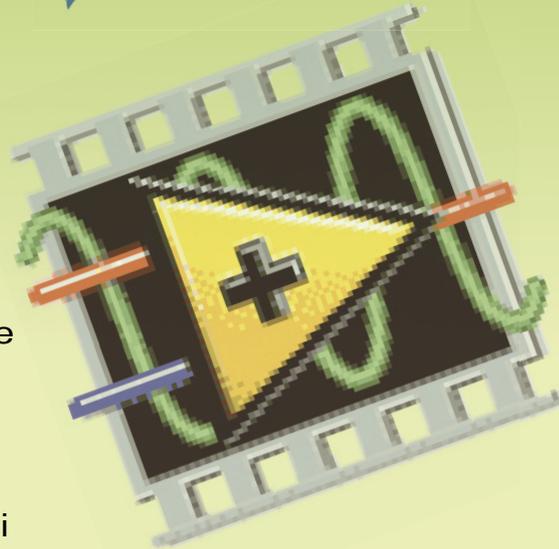
LEZIONE 9

Parliamo di LabVIEW 2010, l'ultima release, che presenta interessanti novità rispetto al passato, come ad esempio migliorie apportate al compilatore, nuovi programmi per soddisfare ed allo stesso tempo utilizzare il contributo degli sviluppatori.

Conoscere e usare LabVIEW



Iniziamo a conoscere la piattaforma software LabVIEW di National Instruments, un innovativo ambiente di sviluppo software che tramite il linguaggio-G permette il rapido sviluppo di applicazioni per l'acquisizione dati ed il controllo remoto di strumentazione hardware.
Prima puntata.



di
FRANCESCO
FICILI

LabVIEW

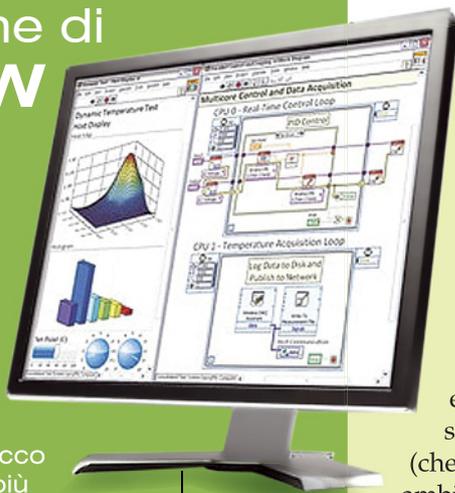
L'ambiente di sviluppo integrato LabVIEW (abbreviazione di Laboratory Virtual Instrumentation Engineering Workbench) nasce ufficialmente nel 1983, quando National Instruments, azienda statunitense specializzata nella produzione di hardware e software per l'acquisizione dati, decide di dotarsi di uno strumento software per il testing rapido dei propri prodotti hardware. Lo sviluppo di tale prodotto, inizialmente pensato per uso interno all'azienda, ebbe un tale successo che già nel 1986 vide la luce la prima versione commerciale, denominata LabVIEW 1.0, inizialmente pensata solo per sistemi

Macintosh. Il contenuto innovativo di questo versatile ambiente di sviluppo software risiedeva nel tipo di linguaggio di programmazione utilizzato: il linguaggio-G, abbreviazione di Graphical Language. A differenza di altri linguaggi di programmazione, come ANSI C e BASIC, che si basano su editor testuali e parole chiave per la generazione del codice, il linguaggio-G usa un approccio di tipo grafico alla programmazione. In sostanza un programma o sottoprogramma G, denominato VI (Virtual Instrument, data la sua naturale predisposizione per i sistemi di acquisizione dati) non esiste sotto forma di testo, ma



L'evoluzione di LabVIEW

La piattaforma di programmazione LabVIEW presenta ormai più di 20 anni di storia, attraverso i quali si sono susseguite diverse versioni ed aggiornamenti. Ecco quali sono le tappe più significative di quest'ambiente di sviluppo software.



Sviluppo di codice in LabVIEW su PC.

- 1986** - LabVIEW 1.0, versione per Macintosh.
- 1990** - LabVIEW 2.0, versione con linguaggio compilato.
- 1992** - Rilasciate versioni per Microsoft Windows e per SunOS.
- 1993** - LabVIEW 3.0, prima versione multiplatforma.
- 1997** - LabVIEW 4.0.
- 1998** - LabVIEW 5.0, prima versione per applicazioni real-time.
- 2000** - LabVIEW 6.0, Internet ready.
- 2003** - LabVIEW 7 Express, aggiunto supporto per PDA e FPGA.
- 2005** - Prima versione con supporto per DSP e dispositivi embedded.
- 2006** - LabVIEW 8.20, versione del ventennale.
- 2007** - LabVIEW 8.5 ottimizzazione dei sistemi multicore
- 2008** - LabVIEW 8.6, dedicato alle architetture e tecnologie parallele, FPGA e wireless
- 2009** - LabVIEW 2009.

può essere salvato solo come file binario, visualizzabile e compatibile solo con LabVIEW. Gli algoritmi e le strutture dati in LabVIEW sono generati attraverso un editor grafico, che fa uso di icone, blocchi ed altri oggetti grafici, uniti attraverso uno o più elementi di connessione. Tale tipologia di approccio rende il codice estremamente simile ad uno schema a blocchi. I blocchi e gli altri oggetti grafici implementano in genere funzioni di alto livello, in modo che lo sviluppo di codice anche piuttosto complesso diventi estremamente rapido. Tale contenuto, fortemente innovativo, ha permesso ad NI di ottenere, nei primi mesi del 1990, un brevetto a protezione dell'idea, rilasciato dalla US Patent Office. Due anni dopo, nel 1992, viene divulgata la prima versione multiplatforma, compatibile con Microsoft Windows, Mac OS e SunOS. Pochi mesi dopo viene realizzata anche una versione per sistemi Linux.

Da allora lo sviluppo e la diffusione di LabVIEW è in continuo aumento, il supporto per l'hardware NI è stato esteso anche ad altri grandi produttori di hardware per l'acquisizione dati e recentemente, tramite l'uso di LabVIEW e del linguaggio-G, è possibile sviluppare software non solo per Personal Computer (che è sempre stato il target tipico di questo ambiente) ma anche per sistemi embedded, come sistemi a microcontrollore, FPGA, Palmari e PLC. Programmi sviluppati in LabVIEW possono essere trovati, ai nostri giorni, in svariati contesti applicativi: si va dall'automazione industriale alle applicazioni di monitoraggio ambientale, dalla programmazione di dispositivi embedded al campo biomedico. Recentemente è addirittura possibile utilizzare LabVIEW con i prodotti Mindstorms di Lego. Con la fine dell'anno scorso, NI ha terminato lo sviluppo dell'ultima release, denominata *LabVIEW 2009*. Tale versione, che sarà quella utilizzata in questo Corso, oltre a distinguersi per il cambio della metodologia di numerazione (si passa dalla numerazione progressiva alla numerazione su base annua), perfeziona le nuove funzioni introdotte con la versione 8.0,

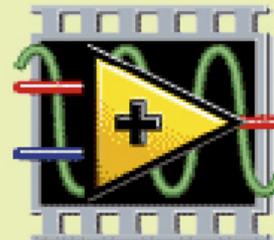


Fig. 1 - Schermata di avvio di LabVIEW 2009 e logo di LabVIEW.

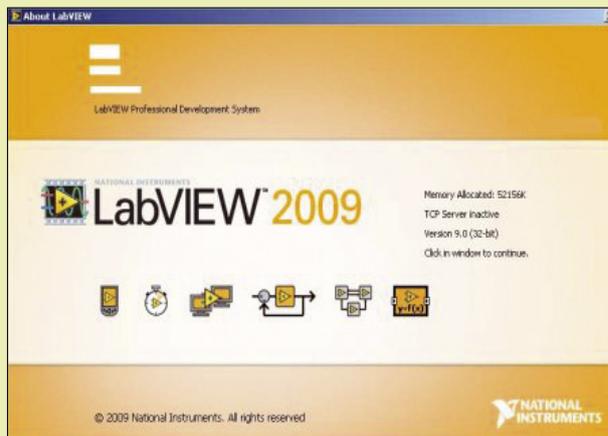
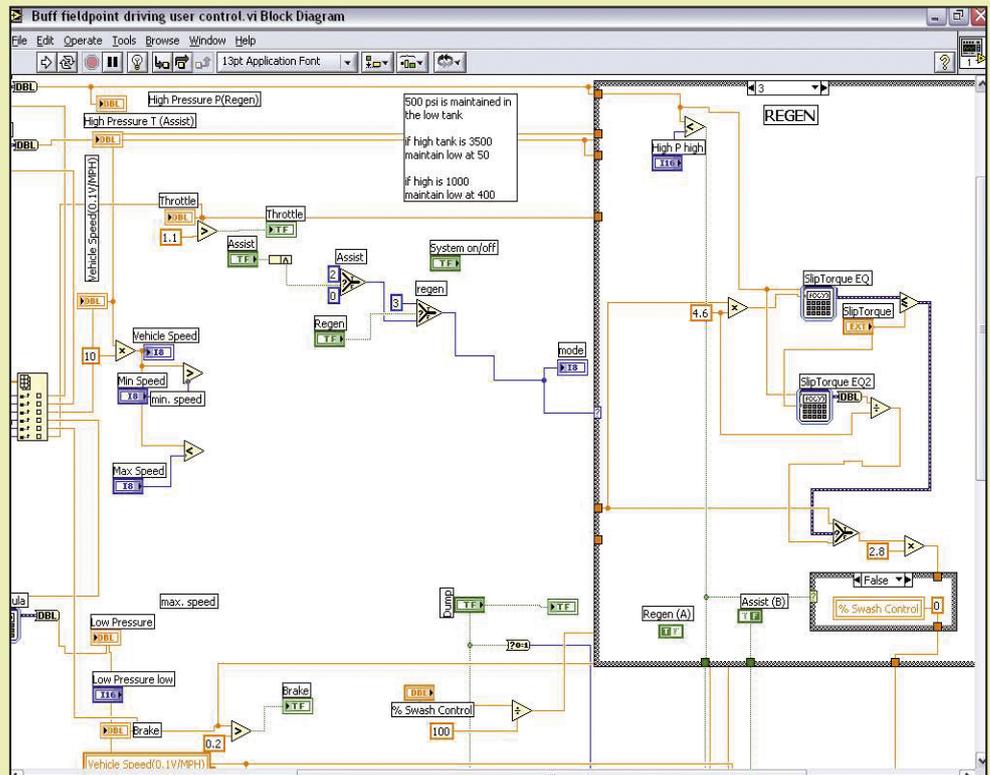


Fig. 2
Esempio di codice scritto in linguaggio-G.

come il supporto per la programmazione ad oggetti e la programmazione di target embedded. Versioni di prova di LabVIEW possono essere scaricate gratuitamente dal sito della National Instruments (www.ni.com), dove è possibile scaricare anche i vari driver per la strumentazione hardware e i tool aggiuntivi. Tali versioni contengono tutte le funzionalità delle versioni commerciali, ma dopo 30 giorni devono essere acquistate o divengono inutilizzabili trattandosi di versioni *trial*. Prossimamente sarà disponibile anche la versione LabVIEW Student Edition, acquistabile online.



PROGRAMMAZIONE GRAFICA

Come già sottolineato, la caratteristica più importante di LabVIEW è l'uso del linguaggio-G per lo sviluppo del codice, in luogo dei più tradizionali linguaggi di programmazione text-based. Ad un esperto programmatore,

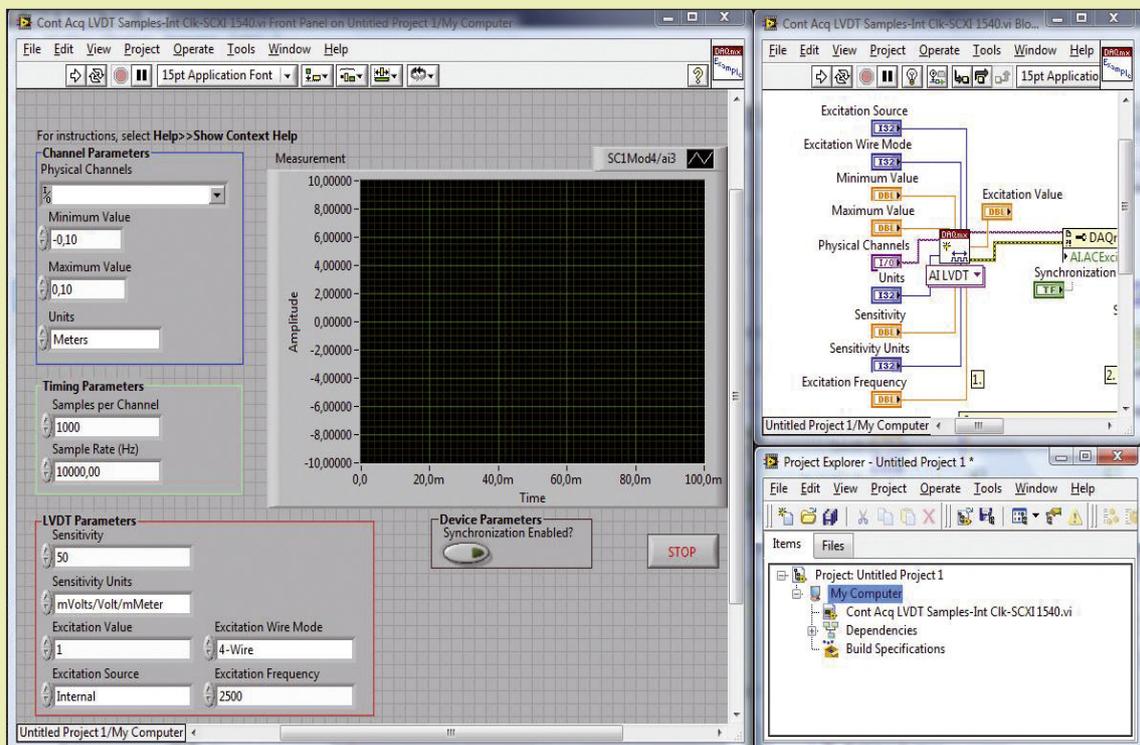


Fig. 3 - Esempio di progetto in LabVIEW.

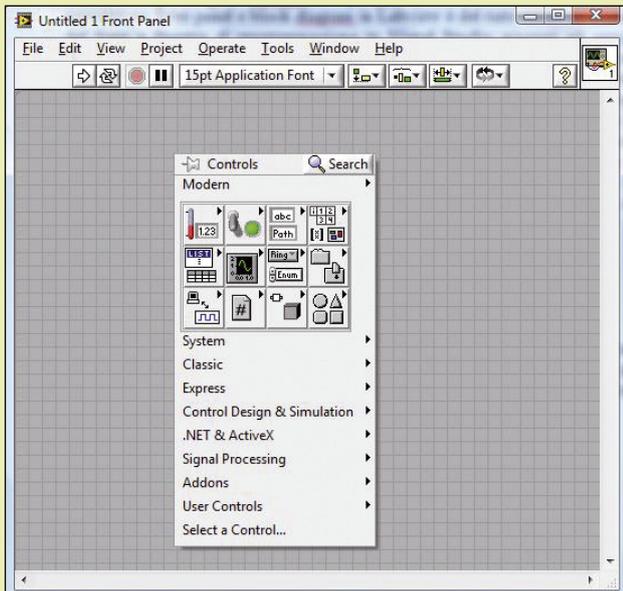


Fig. 4 - Front Panel vuoto di LabVIEW

Fig. 5
Toolbar
di LabVIEW.



citamente realizzato il multithreading, senza bisogno di esplicita gestione da parte del programmatore. È importante sottolineare che, nonostante il linguaggio supporti nativamente il multithreading, è possibile forzare un'esecuzione di tipo sequenziale, tramite l'uso di opportune strutture di controllo. Infine, nonostante l'approccio di alto livello (si consideri che in una scala di valori il linguaggio-G è sicuramente più astratto del Visual Basic), il tempo di esecuzione del codice è paragonabile a quello di un'eseguibile scritto in ANSI C.

che utilizza frequentemente C, Visual Basic o Java, questo tipo di approccio potrebbe apparire quantomeno inconsueto e ci si potrebbe chiedere che livello di controllo delle applicazioni si riesca ad ottenere e soprattutto, quali vantaggi si possono ricavare dalla programmazione grafica. Nonostante il livello di astrazione che si raggiunge utilizzando un editor grafico sia nettamente superiore rispetto a quanto si possa ottenere attraverso un approccio testuale, il livello di controllo dell'applicazione realizzata si spinge fino al singolo byte (e volendo fino al bit) di informazione, tramite l'uso di blocchi o funzioni specifici. Ciò è reso possibile dalla grande varietà di blocchi presenti, che vanno dalla manipolazione di stringe e stream di dati, fino alle operazioni orientate al byte e al bit. Naturalmente, come per ogni linguaggio di programmazione, il livello di controllo dipende dall'esperienza del programmatore nell'uso delle varie funzioni messe a disposizione dall'ambiente. Comunque, per venire incontro anche agli utenti più esigenti, all'interno di un programma LabVIEW possono essere integrati (sotto forma di VI) anche DLL o eseguibili scritti in C o in altri linguaggi di programmazione, in modo da rendere l'ambiente di sviluppo ancora più flessibile. I vantaggi ottenuti da questa filosofia di programmazione sono innumerevoli. Utilizzando il linguaggio-G è possibile realizzare applicazioni molto complesse unendo tra loro pochi blocchi e strutture grafiche di controllo ed interconnessione. Oltre a ciò, poiché i dati possono scorrere contemporaneamente attraverso blocchi e strutture di connessione non consecutive, viene impli-

L'AMBIENTE DI SVILUPPO

L'ambiente di sviluppo comprende tutta una serie di strumenti che permettono la realizzazione di un progetto completo, che porta, al termine del processo, alla realizzazione di un eseguibile o, eventualmente, di un pacchetto di installazione. Gli strumenti forniti dall'ambiente di sviluppo sono:

- una finestra di programmazione, attraverso la quale può essere generato un VI (composta da più elementi, che vedremo nel dettaglio in seguito);
- una finestra di *Project Manager*, per la

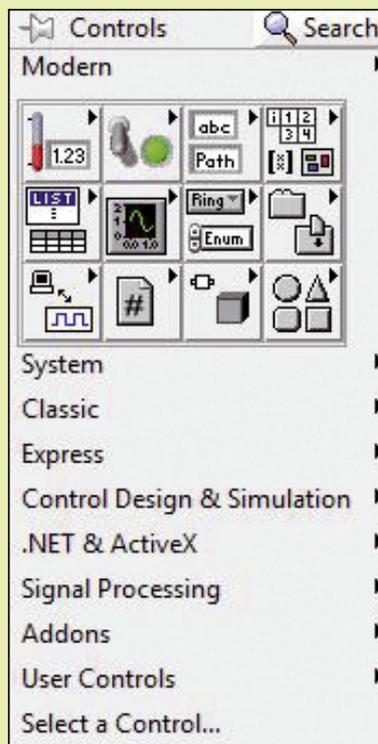


Fig. 6
Control
Palette di
LabVIEW.

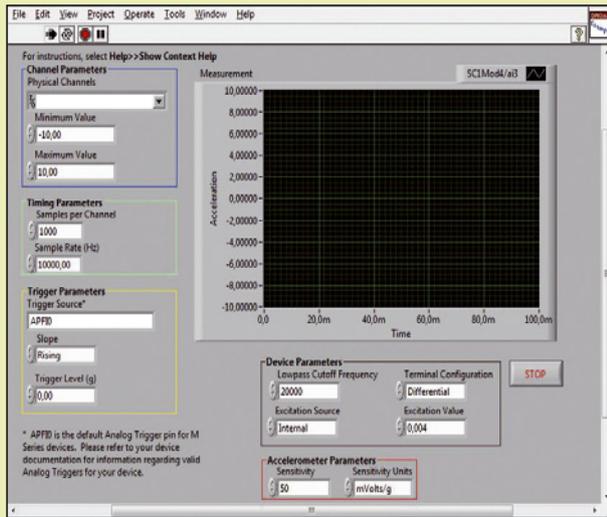


Fig. 7 - Esempio di front panel realizzato con LabVIEW.

gestione del progetto e la generazione di eseguibili ed installer;

- una ricca dotazione di librerie ed esempi di codice.

Tutti questi elementi sono simultaneamente accessibili una volta lanciato LabVIEW (da qui la definizione di *ambiente di sviluppo integrato*) e possono essere utilizzati per sviluppare l'applicazione finale. Nella Fig. 3 è rappresentato un esempio di progetto aperto sotto LabVIEW, dove sono visibili le finestre del VI e del *project manager*.

STRUTTURA DI UN VI

Un programma sviluppato in LabVIEW prende il nome di VI (*Virtual Instrument*). La struttura di un VI comprende tre elementi principali: un *front panel*, un *block diagram* ed una icona/ pannello connettori. Il *front panel* è l'interfaccia grafica di un programma LabVIEW (la denominazione deriva dalla somiglianza con i pannelli frontali degli strumenti reali) e rappresenta la porzione dell'applicazione che l'utente finale vede realmente quando il programma è in esecuzione. Il *block diagram* è invece il foglio di lavoro, ossia l'editor che il programmatore usa per la stesura del codice ed è naturalmente invisibile quando l'applicazione è in esecuzione. In realtà, se il programma è lanciato all'interno dell'ambiente di sviluppo, il *block diagram* può essere visualizzato per scopi di debug ed è possibile inserire indicatori (probe) e *breakpoint*. Il legame esistente tra *front panel* e *block diagram* in LabVIEW è del tutto simile a quello che esiste tra finestra del form e

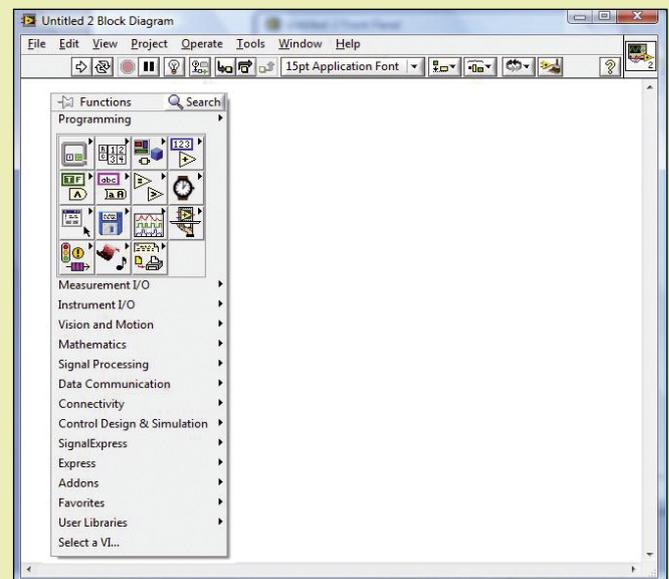
finestra di programmazione in Visual Studio: nel form vengono piazzati gli oggetti grafici che poi sono visualizzati in fase di esecuzione, mentre nella finestra di programmazione si scrive il codice che viene eseguito. Infine, un VI può essere rappresentato anche sotto forma di icona grafica, quando il VI stesso viene riutilizzato all'interno di un altro VI, come si fa in altri linguaggi di programmazione con le subroutine o le funzioni di libreria. Quando un VI viene utilizzato all'interno di un altro, si chiama *subVI*; allo scopo bisogna definire quali variabili accoglie in ingresso e quali presenta in uscita al termine dell'elaborazione, cosa che avviene tramite l'uso del pannello connettori. Un VI può avere un numero indefinito di ingressi ed uscite. Notate che di fatto anche le funzioni di libreria di LabVIEW sono dei *subVI*, con la differenza che non tutti presentano un *block diagram* e un *front panel*.

FRONT PANEL

Come detto in precedenza, il *front panel* è, in pratica, l'interfaccia utente di un programma LabVIEW. All'interno di un *front panel* possono essere inseriti controlli, indicatori e grafici che poi è possibile gestire, dal *block diagram*, come sorgenti di input e di output.

Vediamo da quali elementi è composto un *front panel*, aiutandoci con la Fig. 4. Un *front panel* contiene, al suo interno, un foglio di lavoro, una *toolbar* ed una *control palette*. Il foglio di lavoro è la parte in grigio, che può essere

Fig. 8
Block diagram vuoto.



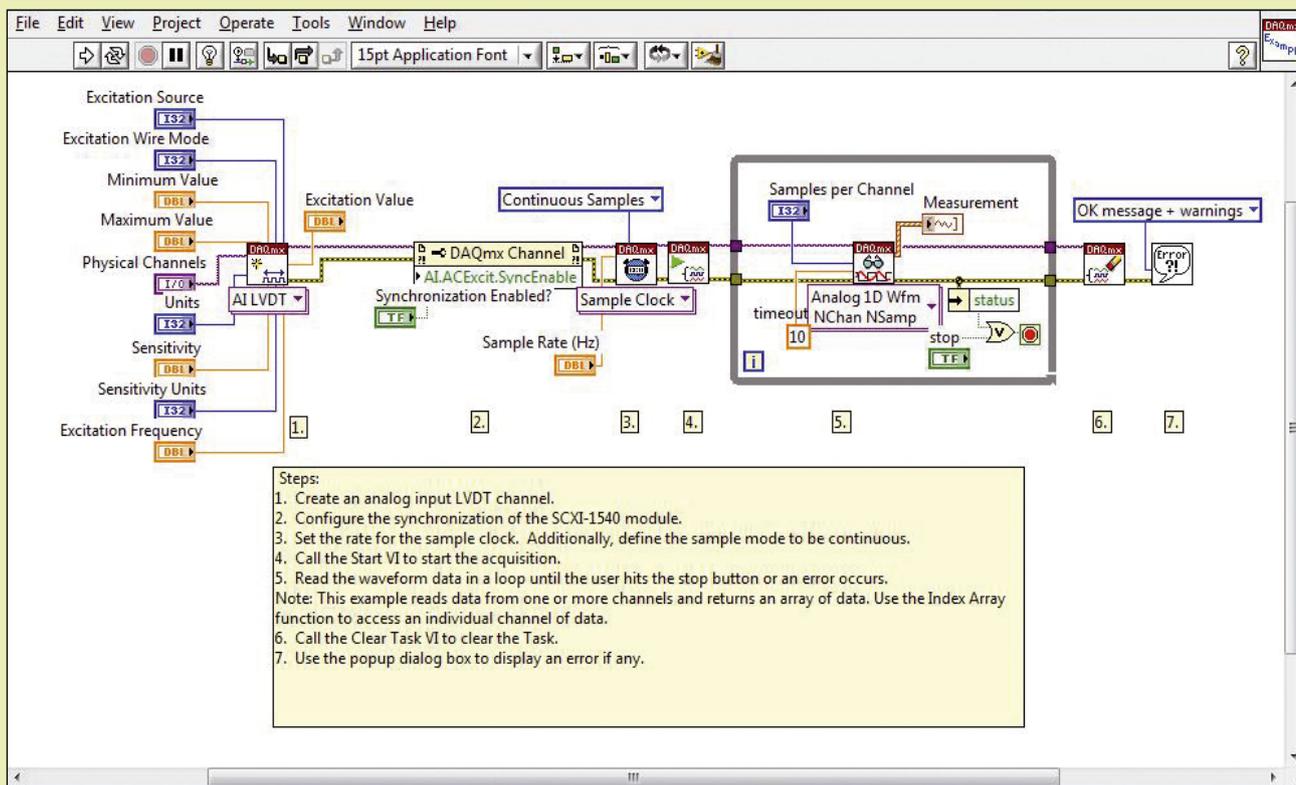


Fig. 9 - Tipico esempio di block diagram realizzato in LabVIEW.

utilizzata per l'inserimento dei vari oggetti grafici che compongono l'interfaccia. L'area di lavoro è virtualmente illimitata, dato che possono essere utilizzate *scrollbar* orizzontali e verticali per spostarsi nelle varie direzioni. La *toolbar*, rappresentata nella Fig. 5 è collocata in alto, immediatamente sotto le voci del menu; contiene i comandi principali per la gestione dell'applicazione e per il debug.

Procedendo da sinistra verso destra, sulla *toolbar* troviamo prima un tasto con una freccia bianca come icona; questo pulsante è denominato *Run* e serve a mandare in esecuzione il VI. Se il VI viene lanciato all'interno dell'ambiente di sviluppo, si attiva automaticamente la modalità di debug ed il *block diagram* può essere aperto per visualizzare lo scorrimento dei dati, sistemare *breakpoint*, lanciare esecuzioni step-by-step ed altro ancora.

Subito a seguire troviamo un pulsante con due frecce indicanti un ciclo ripetuto, denominato *Run continuously*: serve a generare un'esecuzione continua del codice, molto utile in fase di debug. Ancora a seguire, troviamo il pulsante *Abort Execution*, che serve ad arrestare immediatamente l'esecuzione del programma. Infine, tra i comandi principali, troviamo ancora il pulsante *Pause*, che naturalmente serve per mettere in pausa l'esecuzione del

codice. I pulsanti successivi servono a settare dimensioni, posizionamento e stile dei font, nonché all'allineamento degli oggetti grafici sul pannello. La *control palette*, rappresentata nella Fig. 6, è uno degli strumenti più importanti presenti in quest'ambiente di sviluppo; contiene tutti gli oggetti grafici che possono essere inseriti all'interno del pannello, quali indicatori numerici, testuali, manopole, LED, interruttori, pulsanti, grafici ed altro ancora. Tutti gli elementi sono raggruppati per categorie, come ad esempio controlli ed indicatori numerici, controlli ed indicatori booleani, grafici, controlli ed indicatori testuali, e via di

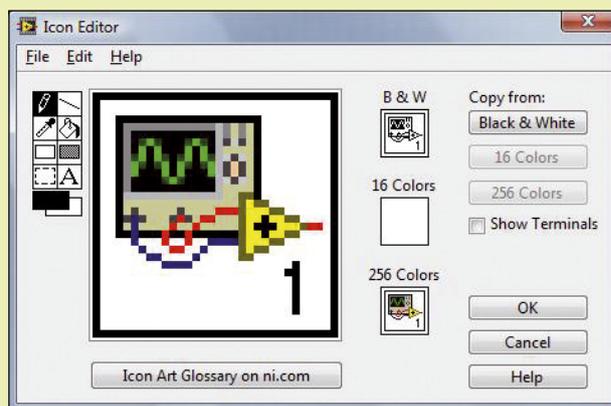


Fig. 10 - Icon editor di LabVIEW.

seguito. Le categorie di oggetti sono innumerevoli, quindi le prime volte si potrà avere l'impressione di perdersi all'interno dei vari sottomenu, ma con un po' di esperienza si riesce facilmente a trovare ciò che si sta cercando. Inoltre, per trovare un oggetto all'interno dei vari menu è possibile utilizzare la funzione di ricerca. Nella Fig. 7 è riportato un esempio di un tipico *front panel* di una applicazione realizzata con LabVIEW.

BLOCK DIAGRAM

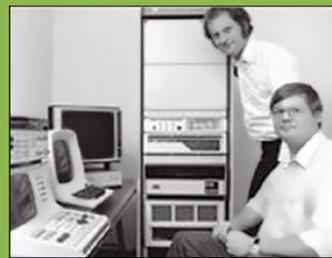
Ogni *front panel* è sempre accompagnato da un *block diagram*, che costituisce l'interfaccia di programmazione. Come detto in precedenza, diversamente dagli editor testuali viene utilizzato il linguaggio-G come linguaggio di programmazione. In Fig. 8 è rappresentato un *block diagram* privo di codice. Così come il *front panel*, anche il *block diagram* contiene un'area di lavoro, una *toolbar* ed una *control palette*. La *toolbar* contiene in parte gli stessi comandi accessibili dal *front panel*, con la medesima funzione, più alcuni strumenti di debug. La differenza sostanziale sta nella *control palette*, che invece degli oggetti grafici contiene *subVI*, strutture di controllo, costanti, variabili ed altri elementi utili alla generazione del codice. Anche in questo caso la quantità di oggetti è notevole e, soprattutto i neofiti potranno avere la sensazione di essersi persi in un oceano di funzioni che fanno le cose più disparate. Non preoccupatevi, perché con un po' di esperienza vi ritroverete dopo poco tempo a navigare agevolmente all'interno dei vari menu. Nella Fig. 9 è rappresentato un tipico *block diagram* di LabVIEW, all'interno del quale è stato sviluppato del codice in linguaggio-G. Come si vede, il codice assomiglia ad una sorta di diagramma a blocchi.

ICONA/PANNELLO CONNETTORI

Ogni VI è caratterizzato da un'icona, che diventerà quella dell'applicazione nel caso tale VI diventi un eseguibile, oppure la sua rappresentazione grafica su un *block diagram* nel caso venga utilizzato come *subVI*. L'icona del VI è visibile nell'angolo in alto a sinistra del *front panel*. Quest'icona è interattiva: cliccando su di essa è possibile editarla o creare dei connettori (i connettori sono necessari solo nel caso il VI debba essere utilizzato come *subVI*). In Fig. 10 è visibile l'*icon editor* integrato in LabVIEW.

Chi è National Instruments?

National Instruments nasce ufficialmente nel 1976 nel garage di James Truchard, nella città di Austin (Texas - U.S.A) con tre soli addetti, incluso lo stesso

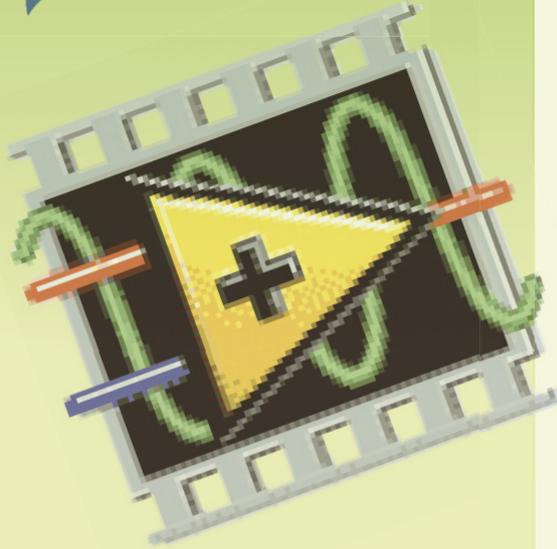
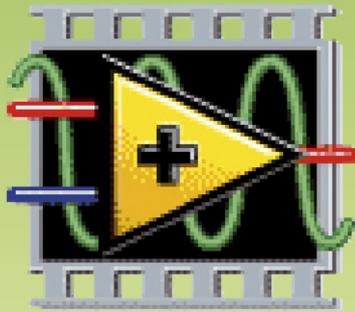


La prima sede di NI, nel garage di James Truchard, ad Austin, Texas.

Truchard. Da quell'anno la società è cresciuta fino a diventare una multinazionale con più di 5.000 dipendenti operante in 41 paesi del mondo. James Truchard è attualmente Presidente, socio e CEO della società, la cui sede principale è sempre rimasta nella città di Austin. Negli ultimi undici anni la società è sempre comparsa nella prestigiosa rivista Fortune 100 tra le "100 best company to work for". Di seguito, le tappe principali della crescita di National Instruments.

- 1976** – Inizia lo sviluppo del primo prodotto NI, nel garage di James Truchard, ad Austin, Texas.
- 1977** – Viene annunciato lo sviluppo del primo prodotto, una scheda di acquisizione dati GPIB, per connettere strumentazione di misura ad un microcomputer. La base Aerea di San Antonio, in Texas diventa il primo cliente ufficiale di NI.
- 1986** – Viene annunciato il rilascio della piattaforma software LabVIEW.
- 1988** – Viene annunciata la prima scheda di acquisizione dati per PC IBM. In questa decade vengono aperte sedi in Giappone (1987), Francia ed Inghilterra (1988) ed Italia (1989).
- 1994** – Viene messo on-line il primo sito web della NI, come supporto alla vendita dei prodotti e supporto tecnico per i Clienti.
- 1996** – Viene lanciato lo standard PXI. In questa decade vengono aperte sedi decentrate in altri 28 paesi del mondo.
- 2002** – Viene aperto il primo centro di produzione oltremare, a Debrecen in Ungheria.
- 2004** – Viene annunciata la piattaforma embedded CompactRIO, un controllore industriale ad alte prestazioni, basato sulla tecnologia RIO (Reconfigurable I/O).

Conoscere e usare LabVIEW



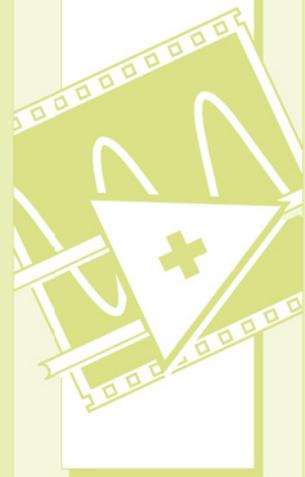
Entriamo nel vivo della programmazione imparando ad utilizzare le strutture di controllo, le strutture dati e i primi oggetti grafici. Proveremo anche a realizzare un primo esempio di programma. Seconda puntata.

di
FRANCESCO
FICILI

In questa seconda puntata muoviamo i primi passi nella programmazione tramite l'uso di LabVIEW e del linguaggio-G. Nella puntata precedente è stato introdotto l'ambiente di sviluppo, proprietario National Instruments, LabVIEW (*Laboratory Virtual Instrumentation Engineering Workbench*), un innovativo ambiente di sviluppo software che, per la realizzazione del codice, fa uso della programmazione grafica, in luogo di quella text-based. Dopo aver descritto i componenti principali dell'ambiente (ossia il front panel, il block diagram e l'icona/pannello dei connettori) ed aver illustrato i tool

di cui il programmatore dispone, in questa puntata inizieremo ad occuparci della programmazione vera a propria, illustrando strutture di controllo e strutture dati utilizzate in LabVIEW, ma anche descrivendo i primi oggetti grafici fondamentali. Nella parte conclusiva di questa puntata realizzeremo un primo semplice esempio di software in LabVIEW che, come abbiamo detto nella puntata precedente, prende il nome di VI (*Virtual Instrument*). Nel seguito, per semplicità di trattazione si farà uso dei termini LabVIEW e linguaggio-G per indicare il linguaggio di programmazione, nonostante tale terminologia non sia

LabVIEW



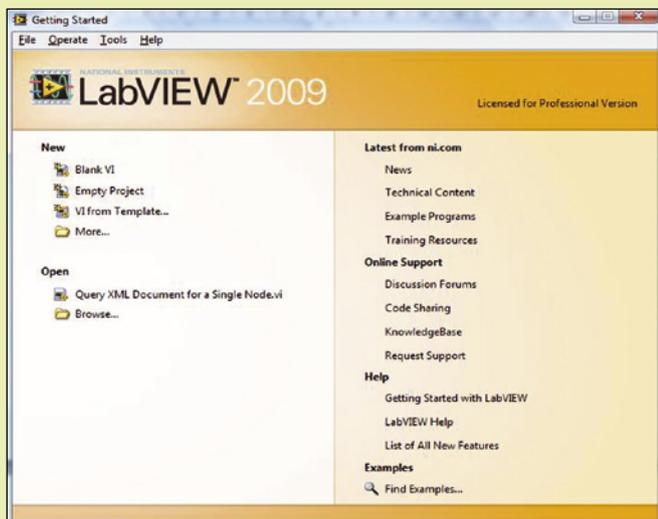


Fig. 1 - Schermata iniziale di LabVIEW.

esatta. Ricordiamo infatti che con LabVIEW s'intende l'ambiente di sviluppo, mentre il linguaggio utilizzato all'interno di LabVIEW è il linguaggio-G. Tuttavia, dato che tale tipologia di linguaggio si usa solo in questo ambiente, con il termine LabVIEW viene spesso indicato anche il linguaggio-G.

Prima di addentrarci nei dettagli della programmazione vera e propria, verrà illustrato brevemente come installare ed avviare l'ambiente. Per ottenere una copia di valutazione di LabVIEW bisogna collegarsi al seguente indirizzo web: <http://www.ni.com/trylabview>.

Da questa pagina può essere effettuato il download dell'ultima versione di LabVIEW, che può essere installata in modalità di valutazione (trial) valida per trenta giorni, alla scadenza dei quali non è più possibile avviare l'ambiente. Per attivare la modalità di valutazione (dopo aver decompresso il file) basta lanciare l'installer ed alla fine del processo di installazione, tra le varie opzioni, scegliere *evaluation version*. A questo punto è possibile avviare per la prima volta l'ambiente, la cui schermata iniziale si presenta come in **Fig. 1**. Come si può vedere, sono disponibili diverse sezioni, ognuna delle quali contiene una serie di opzioni:

- **New:** attraverso questa sezione può essere creato un nuovo VI, un nuovo progetto, oppure si può realizzare un VI partendo da un Template; nelle ultime versioni viene caldeggiato l'uso dei progetti, per uno sviluppo più organico del codice, sebbene il progetto (che è indispensabile per la realizzazione di un eseguibile o di un installer) possa essere creato indifferentemente prima o dopo la scrittura del VI.
- **Open:** da questa sezione si può aprire un

VI salvato precedentemente e vengono anche elencati gli ultimi VI aperti.

- **Latest from ni.com:** questa sezione contiene una serie di link ad alcune risorse del sito della NI, come news, esempi di codice e contenuti tecnici.
- **Online support:** sezione relativa al supporto tecnico ed ai forum degli sviluppatori LabVIEW.
- **Help:** questa sezione contiene link alla *Getting Started Guide* e all'Help in linea; da essa è possibile accedere ad una lista delle caratteristiche dell'ambiente aggiunte recentemente e non presenti nelle versioni precedenti.
- **Examples:** questa sezione apre una nuova finestra contenente una serie di esempi di codice, divisi per argomenti; è utilissima sia ai principianti che agli esperti, in quanto molti problemi relativi alla realizzazione del codice possono essere risolti riutilizzando proprio tali esempi. La sezione comprende anche un motore di ricerca dedicato.

STRUTTURE DATI IN LabVIEW

Come ogni altro linguaggio di programmazione, LabVIEW gestisce diversi tipi di dati e dispone dei più comuni tipi di strutture dati. I più importanti tipi di dati gestiti in LabVIEW sono:

- *Interi*; con segno o senza, a 8, 16 e 32 bit;
- *Floating Point*; complessi o meno, a precisione singola, doppia ed estesa;
- *Stringhe*; questa tipologia racchiude sia i caratteri singoli, che gli insiemi di caratteri, ossia appunto, le stringhe;
- *Boolean*; dati di tipo booleano (true e false).

Oltre a questi tipi di dati, sono disponibili delle strutture dati. Le più comuni strutture dati in LabVIEW sono le seguenti:

- *Array*: insieme di dati dello stesso tipo, con indice. Gli array possono essere sia monodimensionali (vettori) che bidimensionali (matrici).

Tabella 1 - Associazione tra i tipi di dati e i colori in LabVIEW.

Tipo di dato	Colore
Interi	Blu
Floating-point	Arancio
Stringhe	Rosa
Boolean	Verde

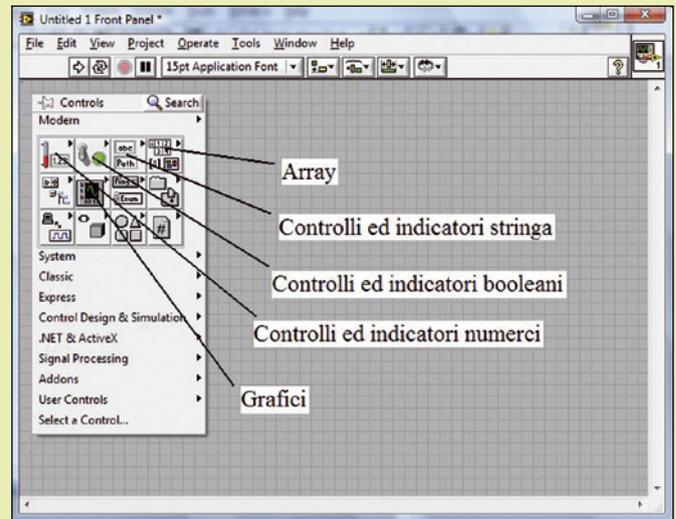
Fig. 2 - Alcuni contenuti della control palette.

- *Cluster*: il tipo *cluster* è di fatto un macro-contenitore, all'intero del quale vengono inseriti insieme di dati indicizzati (come per gli *array*), ma di diversi tipi. Un tipico esempio di *cluster* è quello all'interno del quale vengono veicolati i messaggi di errore, che per loro natura contengono dati di carattere diverso (ad esempio un codice errore di tipo *string* ed un segnalatore di errore di tipo *boolean*).

Oltre a questi tipi e strutture dati standard, LabVIEW gestisce anche altri tipi di dati (come ad esempio il tipo *waveform*) ma al momento non ci occuperemo di queste altre strutture, limitandoci a considerare solo i tipi standard appena descritti. È importante notare che, per aiutare il programmatore nell'identificazione di un particolare tipo di dato veicolato da un cavo tra due o più blocchi, LabVIEW usa le icone ed i colori. Le associazioni tra i tipi di dati elencati precedentemente ed i colori sono riportate nella **Tabella 1**. Vediamo adesso come utilizzare i controlli e gli indicatori per costruire un front panel. In LabVIEW è possibile inserire una vasta gamma di controlli e indicatori, che sono rappresentati sotto forma di oggetti grafici nel front panel e sotto forma di variabili nel block diagram. Gli oggetti grafici che è possibile inserire richiamano (nella forma e nel nome) i controlli e gli indicatori che tipicamente possono essere trovati su un pannello frontale di uno strumento da laboratorio. Gli oggetti grafici più comunemente utilizzati in LabVIEW sono i seguenti:

- pulsanti, LED e Switch: associati a variabili di tipo *boolean*;
- controlli ed indicatori numerici, barre e *knob* (manopole): associati a variabili numeriche, sia di tipo intero che *floating-point*;
- controlli ed indicatori di tipo stringa: associati a variabili di tipo stringa;
- tabelle: associate ad *array* sia monodimensionali che bidimensionali;
- grafici: possono essere associati a variabili numeriche o ad *array*.

Proviamo a questo punto a generare alcuni oggetti grafici e vediamo come la loro crea-



zione sul front panel produce la generazione automatica di alcune variabili sul block diagram. Per prima cosa, dal menu di partenza (**Fig. 1**) creiamo un VI vuoto, selezionando l'opzione *Blank VI* (sotto la sezione *New*). Sul front panel del VI appena aperto accediamo alla control palette, premendo il pulsante destro del mouse in una zona qualsiasi dell'area di lavoro. Come si può vedere dalla **Fig. 2**, la control palette contiene una serie di oggetti grafici, divisi per sezione. Oltre alle sezioni evidenziate con i commenti ne esistono altre che contengono altri tipi di controlli; per il momento limitiamoci a considerare i cinque gruppi menzionanti in precedenza. Per inserire un oggetto bisogna navigare con il puntatore del mouse all'interno dei menu (i quali si espandono automaticamente al passaggio del puntatore) fino ad individuare l'oggetto desiderato, selezionarlo e trascinarlo con il mouse nella posizione voluta, quindi premere il tasto sinistro per posizionarlo. Inseriamo i seguenti oggetti:

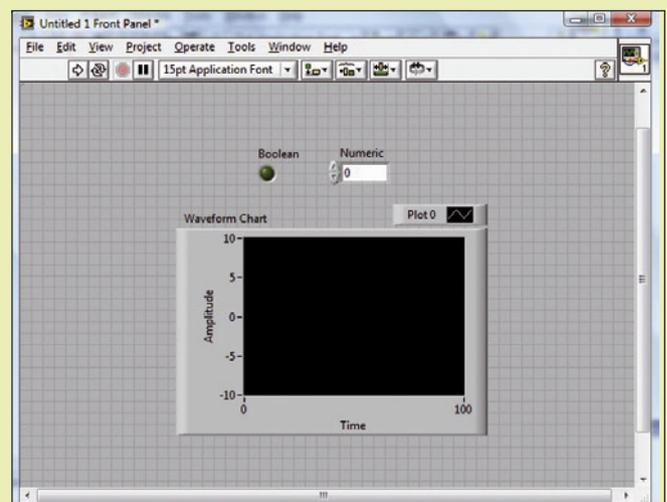
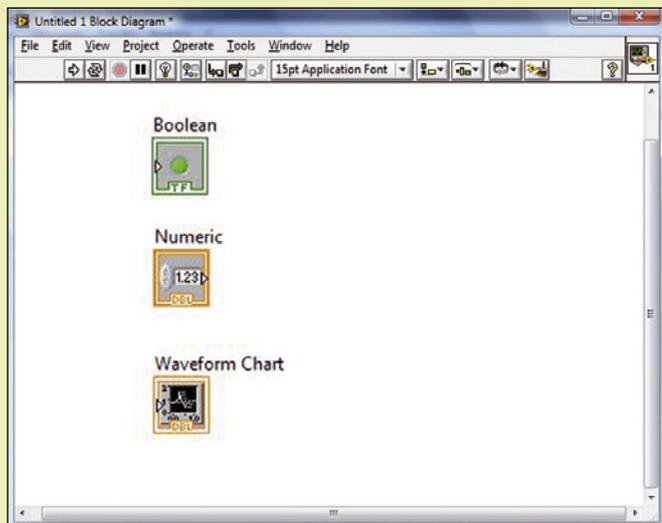


Fig. 3 - Front Panel dopo l'inserimento degli oggetti grafici.



- un LED rotondo, percorso *Boolean/Round LED*;
- un controllo numerico, percorso *Numeric/Numeric Control*;
- un grafico di tipo chart, percorso *Graph/Waveform Chart*.

Se tutto viene eseguito correttamente, dovreste ottenere un pannello di controllo simile a quello illustrato nella Fig. 3.

Ora apriamo il block diagram associato al pannello frontale appena modificato, che appare come illustrato nella Fig. 4. Notate che l'inserimento dei tre oggetti grafici sul front panel ha prodotto la generazione automatica, da parte dell'ambiente, di tre variabili associate: la variabile *boolean* (associata al LED), la *numeric* (associata al controllo numerico) e la *waveform chart* (associata al grafico). I nomi vengono assegnati automaticamente (ed in maniera univoca) dall'ambiente, ma possono essere modificati a piacimento dal programmatore. In questo primo programma sono stati inseriti degli oggetti grafici e delle variabili associate, ma se proviamo ad eseguirlo non succede nulla, pur non rilevando alcun errore. Questo perché mancano ancora le strutture di controllo che permettono l'esecuzione di uno specifico algoritmo. Nel prossimo paragrafo analizzeremo proprio l'implementazione di tali strutture in LabVIEW, in modo da essere in grado di realizzare un primo semplice esempio di programmazione.

STRUTTURE DI CONTROLLO IN LabVIEW

Come ogni altro linguaggio di programmazione, LabVIEW dispone di una serie di strutture di controllo, costituenti gli strumenti che il linguaggio mette a disposizione del program-

Fig. 4 - Block Diagram relativo al front panel illustrato nella Fig. 3.

matore per la realizzazione degli algoritmi. Le strutture utilizzate sono quelle classiche dei linguaggi di programmazione; si tratta quindi di cicli e strutture di selezione multipla. Oltre a queste strutture di base, l'ambiente ne mette a disposizione altre, rese necessarie dalla particolare tipologia di programmazione, come si vedrà più chiaramente nelle puntate successive. Le strutture di controllo, essendo elementi relativi alla programmazione, possono essere utilizzate solo all'interno di un block diagram e non sono accessibili sui front panel. Sono tutte raggruppate sotto la voce *Structures* della control palette (Fig. 5). Ricordiamo che per aprire la control palette (sia in un block diagram che in un front panel) è sufficiente cliccare su un punto qualsiasi dell'area di lavoro con il tasto destro del mouse. Una volta aperta, la control palette può anche essere fissata sull'area di lavoro in maniera permanente cliccando sull'icona a forma di puntina collocata nell'angolo in alto a sinistra.

CICLI FOR E WHILE

Partiamo analizzando i cicli, che tipicamente vengono indicati anche con il termine inglese *loop*. LabVIEW dispone di due differenti tipi di *loop*, un ciclo a conteggio (ciclo *for*) ed un ciclo con condizione di uscita (ciclo *while*).

Ciclo For

È il classico ciclo a conteggio, che ritroviamo normalmente anche nei tradizionali linguaggi di programmazione basati su stringhe di testo.

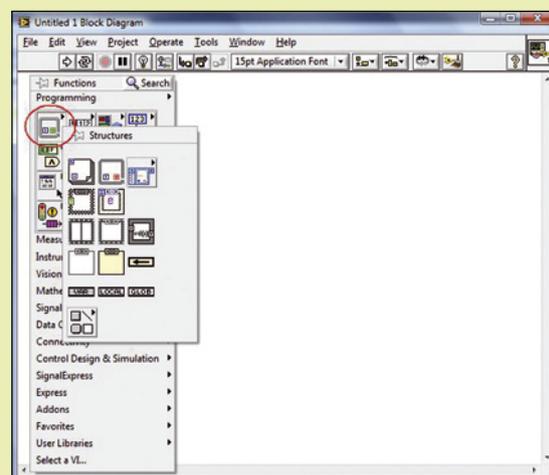


Fig. 5 - Posizione delle strutture di controllo all'interno della control palette.

Fig. 6 - Ciclo For vuoto. Il parametro **N** è stato collegato ad una costante, mentre la variabile indicante il conteggio è stata collegata ad un indicatore numerico sul front panel.

Tutto il codice grafico contenuto all'interno dei margini del ciclo for viene ripetuto finché l'indice di conteggio non raggiunge il valore indicato dal programmatore. Nella **Fig. 6** è riportato un ciclo for vuoto di LabVIEW. L'icona che lo rappresenta è facilmente riconoscibile, ed è costituita da tre fogli sovrapposti, ad indicare la ripetizione del ciclo per un numero stabilito di volte. La struttura dispone di due parametri di controllo: in alto a sinistra troviamo il parametro *N* (intero) ed in basso a sinistra il parametro *i* (sempre intero). *N* è un ingresso cui può essere collegato sia un valore costante, sia una variabile, ed indica il numero di volte per cui verrà ripetuto il ciclo, mentre la variabile rappresenta il contatore del ciclo. Nell'esempio della **Fig. 5** il parametro *N* è stato collegato ad una costante di valore pari a 100. In questo modo tutto il codice grafico che verrà inserito all'interno del ciclo verrà ripetuto 100 volte. Inoltre la variabile *i* è stata collegata ad un indicatore numerico sul pannello di controllo, che conterrà un valore indicante l'indice di avanzamento del ciclo.

Ciclo While

Un altro ciclo classico gestito da LabVIEW è il *while*. Anche questo, come il ciclo *for*, è presente in molti altri linguaggi di programmazione (in alcuni linguaggi esiste il ciclo *repeat...until*, ma è sostanzialmente la stessa cosa) ed è caratterizzato da un'esecuzione ciclica del codice contenuto al suo interno finché non si verifichi una certa condizione di uscita, che viene controllata ad ogni esecuzione del ciclo stesso. Nella **Fig. 7** è riportato un ciclo *while* vuoto di LabVIEW. Anche l'icona del ciclo *while* è piuttosto intuitiva, in quanto costituita da una freccia disposta in modo da richiudersi su se stessa. Il ciclo dispone di un ingresso (condizione di uscita dal ciclo) e di una uscita (indice); continua ad eseguire il codice contenuto al suo interno finché la condizione di uscita non viene verificata. Nell'esempio visibile nella **Fig. 7**, la condizione di uscita è stata collegata ad un controllo booleano (un tasto di stop) sul front panel. In questo modo il ciclo *while* viene ripetuto fin quando non si interviene sul front panel premendo il pulsante di stop.

STRUTTURA CASE

Come altri linguaggi di programmazione,

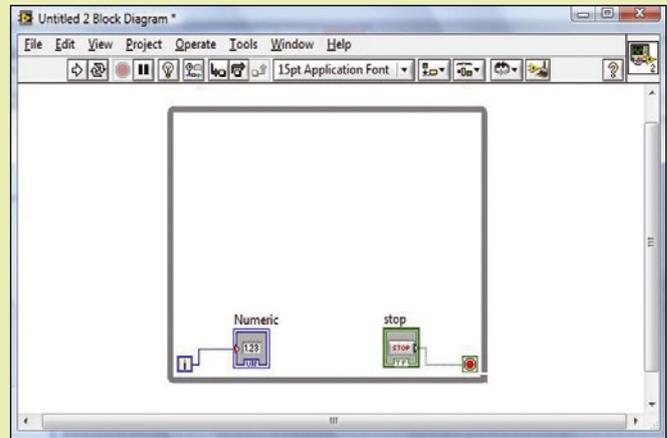
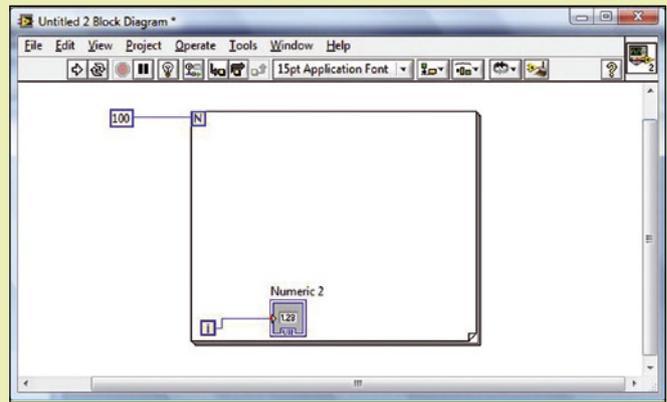


Fig. 7 - Ciclo While vuoto. La condizione di uscita del ciclo è stata collegata ad un controllo booleano sul front panel, mentre l'indice è stato collegato al solito indicatore numerico.

LabVIEW dispone delle cosiddette strutture di selezione multipla, ossia di strutture che permettono di eseguire due porzioni differenti di codice in funzione di una variabile di selezione. Strutture equivalenti a questa sono ad esempio la struttura *if...then...else* del Visual Basic oppure la *switch...case* del C. LabVIEW usa una singola struttura che svolge entrambe le mansioni della struttura *if...then* e dello *switch...case*; essa è denominata *Case Structure*. Nella **Fig. 8** è rappresentata un struttura di tipo *Case*; come vedete, essa è dotata di un indice di scorrimento che permette di scorrere tra i vari casi. Per impostazione predefinita, la struttura, prevedendo un ingresso di selezione di tipo booleano, presenta due casi: *True* e *False*. Oltre all'indice di scorrimento, la struttura è dotata di un ingresso, rappresentato con il punto interrogativo e colorato di verde (ad indicare, in questo caso, un ingresso di tipo *boolean*). Su quest'ingresso va cablata la variabile di selezione, che può essere un pulsante, un interruttore o anche un controllo di tipo numerico; in base allo stato di tale

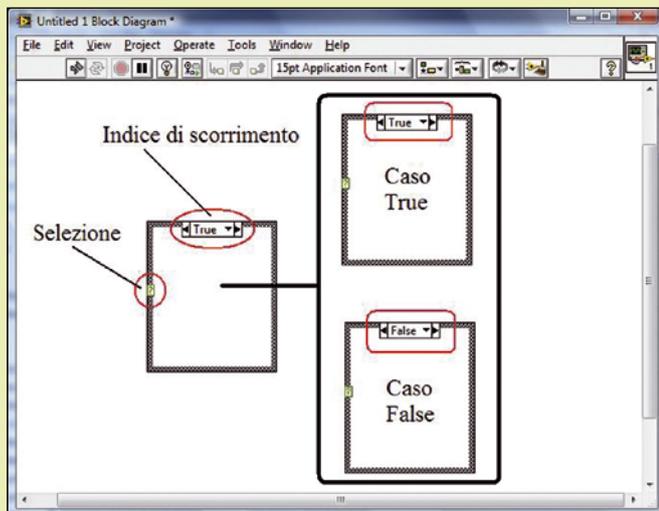


Fig. 8 - Case Structure.

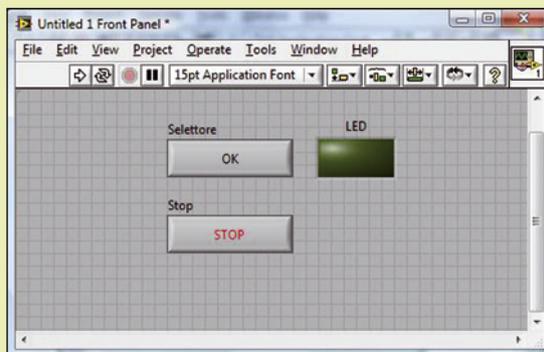


Fig. 9 - Front Panel del programma di esempio.

variabile di selezione verrà eseguito il caso associato. Nel caso sull'ingresso venga collegata una variabile di tipo differente dal *boolean* (ad esempio un controllo numerico), la struttura case si adatta al differente tipo di input, con un comportamento simile alla *switch...case* del C. Nell'ipotesi che la variabile presenti un numero di casi superiore a due, per aggiungerne altri bisogna cliccare con il tasto destro del mouse sul bordo e selezionare una delle due opzioni *Add Case After* o *Add Case Before*. A questo punto abbiamo tutti gli elementi necessari per la realizzazione di un primo esempio, semplicissimo, però in grado di far cogliere al lettore le potenzialità di LabVIEW e del linguaggio-G. In genere come primo approccio nei comuni linguaggi di programmazione è spesso utilizzato il programma "Hello World!" che punta a far comparire questa scritta sul monitor del PC. Noi invece faremo un esempio decisamente più familiare agli appassionati di microcontrollori: faremo accendere un LED (virtuale) alla pressione di un tasto sul front panel per poi farlo spegnere alla successiva pressione dello stesso

tasto. Apriamo un nuovo VI, e, utilizzando la control palette, posizioniamo sul pannello frontale un pulsante OK (*Boolean/OK Button*), un pulsante Stop (*Boolean/Stop Button*) ed un LED quadrato (*Boolean/square LED*). La forma di pulsanti e LED (come di tutti gli altri oggetti grafici) può essere modificata posizionando il cursore sul bordo fino a quando non compare il simbolo della freccia: a questo punto basta cliccare con il tasto sinistro del mouse e trascinare l'oggetto finché non assume le forma desiderata. Per cambiare il nome basta invece fare doppio clic sulla label e modificare il nome. Rinominiamo i tre oggetti nel seguente modo: al posto di "OK Button" scriviamo "Selettore" e al posto di "boolean" scriviamo semplicemente "LED". La label del pulsante di stop va bene così. A questo punto dovrete avere ottenuto un front panel simile a quello riportato nella Fig. 9. Prima di procedere con il block diagram è necessario fare un'ultima modifica al front panel: cliccando con il tasto destro del mouse sul pulsante "Selettore" evidenziamo le proprietà meccaniche (*Mechanical Action*) e modifichia-

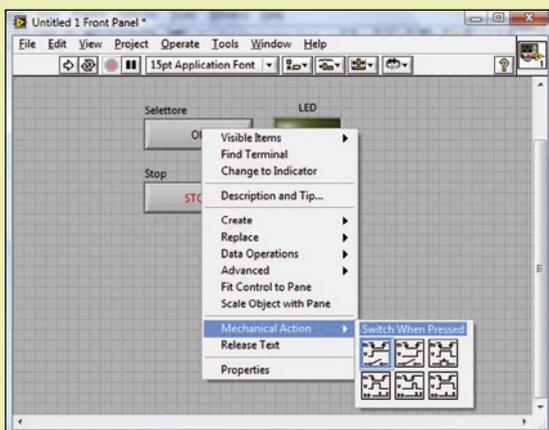


Fig. 10 Front Panel del programma di esempio.

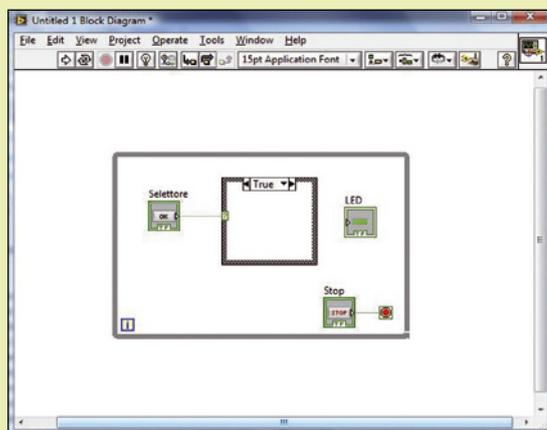


Fig. 11 Costruzione del block diagram.

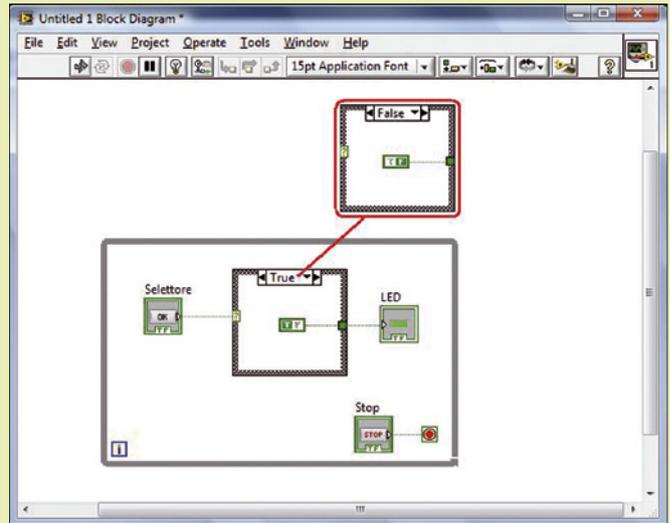
Fig. 12 - Block diagram definitivo del programma di esempio.

mole in "Switch when pressed", come si vede nella Fig. 10. Questo fa sì che il pulsante abbia un comportamento bistabile, restando premuto anziché tornare in posizione una volta rilasciato. A questo punto possiamo aprire il block diagram e modificare il codice. Creiamo un ciclo *while* ed inseriamo tutte le costanti al suo interno.

Collegiamo il pulsante di stop alla condizione di uscita del *while*, questa farà in modo che il programma interrompa l'esecuzione alla pressione del pulsante stop. Per collegare una variabile (o un subVI) ad un dato ingresso, bisogna posizionarsi con il puntatore del mouse vicino all'uscita della variabile, finché esso non cambia forma, passando da frecciolina a rochetto di filo; a questo punto basta cliccare e trascinare fino al punto in cui si desidera effettuare la connessione ed infine cliccare nuovamente con il tasto sinistro del mouse. Se l'operazione viene eseguita correttamente, compare un filo di connessione tra i due terminali, il cui colore rappresenta il tipo di variabile utilizzata.

Dopo aver collegato il pulsante di stop, creiamo una struttura *Case* e colleghiamo il pulsante di selezione all'ingresso del *Case*, come mostrato nella Fig. 11.

A questo punto vanno inseriti due piccolissimi frammenti di codice all'interno di *True* e *False* del *Case*. Selezioniamo, tramite la control palette, due costanti booleane, un *true* (*Boolean/true constant*) ed un *false* (*Boolean/false constant*) e posizioniamole rispettivamente all'interno del *true* e del *false* della struttura *Case*. Adesso colleghiamo le due costanti al terminale d'ingresso del LED, passando attraverso i bordi del *case*. Passando attraverso i bordi compare automaticamente un quadratino colorato sul punto di passaggio: questo quadratino viene denominato tunnel e simboleggia il fatto che il dato viene portato al di fuori della struttura. A questo punto, se tutto è stato eseguito correttamente, dovrete avere un block diagram come quello visibile nella Fig. 12. Nella figura in questione è stato evidenziato anche il *false* della struttura *Case*, sebbene sia visibile solo un caso per volta. Ora possiamo mandare in esecuzione il programma appena scritto e verificarne il funzionamento. Per avviare il programma in modalità di debug, premiamo il simbolo della



freccetta, che trovate in alto a sinistra sulla barra dei tool. Come si vede, premendo il tasto "Selettore" è possibile comandare l'accensione e lo spegnimento del LED, mentre con il tasto stop si blocca l'esecuzione del programma. Il risultato è visibile nella Fig. 13.

Questo primo, semplicissimo esempio, evidenzia la potenza e la versatilità di LabVIEW nella scrittura del codice. Con pochi semplici passi abbiamo costruito un'interfaccia grafica ed un semplice algoritmo di controllo. Nelle prossime puntate vedremo come è possibile creare software ben più complessi, in grado di interfacciarsi anche con periferiche di I/O, con tempi di sviluppo estremamente inferiori rispetto all'uso di ambienti tradizionali. ■

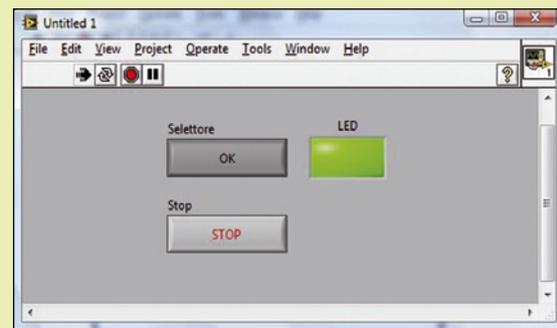
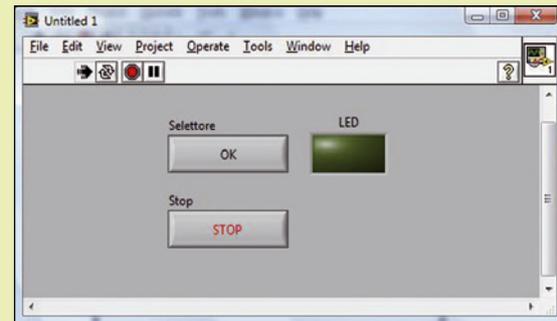
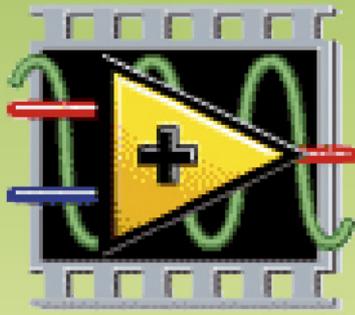


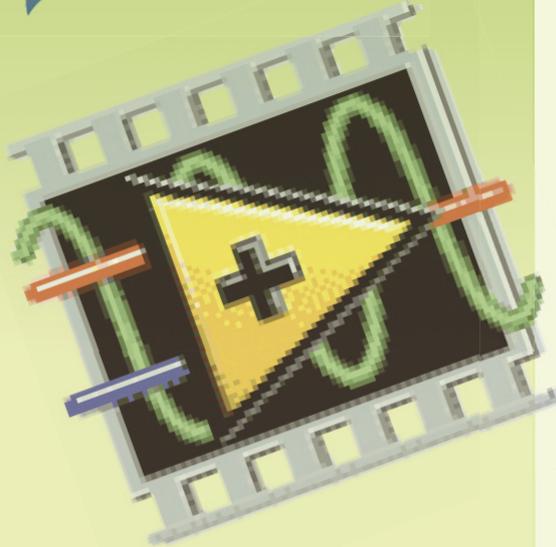
Fig. 13 - Esecuzione del programma di esempio.



Conoscere e usare LabVIEW



Continuiamo la descrizione delle strutture di controllo della piattaforma software LabVIEW, un innovativo ambiente di sviluppo software che, tramite il linguaggio-G, permette il rapido sviluppo di applicazioni per l'acquisizione dati ed il controllo remoto di strumentazione hardware.
Terza Puntata.



di
FRANCESCO
FICILI

LabVIEW

Nella scorsa puntata abbiamo iniziato a spiegare i principi di base della programmazione in LabVIEW, descrivendo le strutture dati ed iniziando ad esaminare le prime strutture di controllo. In questa terza puntata continueremo a descrivere le strutture di controllo disponibili in LabVIEW, che sono tra i principali strumenti a disposizione del programmatore per lo sviluppo degli algoritmi, e scriveremo nuovi software di esempio, in modo da comprendere meglio le potenzialità della programmazione grafica. A questo punto del corso supponiamo che abbiate acquisito un minimo di manualità con gli

strumenti di base dell'ambiente di sviluppo (li abbiamo già ampiamente descritti nelle puntate precedenti) quindi eviteremo di specificare nel dettaglio le operazioni, ma le descriveremo sommariamente.

ALTRE STRUTTURE DI CONTROLLO

Oltre alle strutture classiche, come la *for*, la *while* e la *case* descritte nella puntata precedente, LabVIEW introduce altre strutture di controllo, che permettono una migliore gestione del flusso di dati e consentono di superare alcuni limiti intrinseci derivanti dall'uso di un ambiente di programmazione grafico ad esecuzione parallela delle istruzioni.



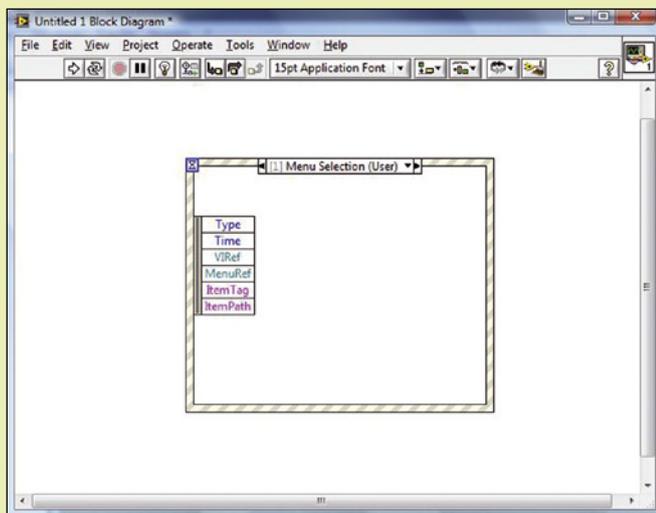


Fig. 1 - Struttura Event Case in LabVIEW.

Struttura Event Case

La event case è una struttura case gestita ad eventi, la cui icona è riportata nella Fig. 1; essa attende fino al verificarsi di un determinato evento e non appena questo si verifica lo gestisce con il caso appropriato. In sostanza si tratta di un case gestito ad interrupt. L'uso di questa struttura permette di modificare l'esecuzione dei programmi LabVIEW, introducendo la possibilità di generare algoritmi event driven, come avviene, ad esempio, nel Visual Basic.

Nella struttura Event Case è sempre presente, per impostazione predefinita, il caso *Timeout*, che viene gestito appena trascorre un tempo indicato dal programmatore senza alcun tipo di attività sul pannello frontale. Il valore di attesa viene cablato, in millisecondi, sul simbolo della clessidra posto in alto a sinistra della struttura stessa.

Vediamo a questo punto come funziona la struttura Event Case, attraverso un semplice esempio. Apriamo un VI vuoto e salviamolo, nominandolo come *esempio 2*. Al suo interno posizioniamo un Knob (*Numeric/Knob*), un controllo numerico (*Numeric/Numeric Control*), due indicatori numerici (*Numeric/Numeric Indicator*) e tre LED (*Boolean/Round LED*). Posizioniamo questi oggetti sul pannello frontale come mostrato nella Fig. 2. Vi ricordiamo che tutti i programmi presentati come esempi all'interno del corso sono scaricabili liberamente dal nostro sito Internet www.elettronica.in.it.

Come si può notare, nel pannello frontale di questo secondo esempio sono stati introdotti alcuni elementi decorativi. LabVIEW dispone di una vasta gamma di elementi decorativi,

tutti accessibili dalla control palette del pannello frontale sotto la sezione *decorations*. Non hanno caratteristiche di tipo funzionale, ma possono servire a fornire al front panel un aspetto più professionale (o un'estetica più accattivante) e risultano utili per rendere più leggibili e strutturati front panel con un grosso numero di controlli, indicatori e grafici. Nel caso in esempio è stata usata una serie di *recessed frame* per separare idealmente controlli e indicatori di tipo diverso.

Vediamo adesso come è stata gestita la programmazione grafica. Lo scopo del programma è quello di evidenziare come sia possibile associare un determinato comportamento del pannello frontale con le azioni che vengono effettuate su di esso dall'operatore, mentre in contemporanea viene eseguito un ciclo. Nel caso specifico, l'operatore può intervenire sui due controlli (la Knob e il controllo numerico) usando il mouse. Noi vogliamo che il LED Knob si accenda ad ogni cambiamento di valore del controllo associato, ed ancora, che il LED Num si accenda appena il puntatore del mouse passa sopra il controllo Numeric. Faremo in modo che durante l'esecuzione del programma i valori dei due indicatori numerici seguano quelli impostati dai controlli. Inoltre, in assenza di attività sul pannello frontale per dieci secondi, vogliamo che il programma vada in stato di timeout (accedendo il LED corrispondente); Infine, una volta entrato in questo stato vogliamo che si arresti definitivamente dopo un secondo. Nella Fig. 3 è riportato il block diagram relativo al programma *esempio 2*.

Come si vede è stato usato un ciclo while (che simula l'esecuzione di un programma principale) all'interno del quale è stata posizionata la struttura event case. Il codice racchiuso all'interno del riquadro colorato è l'applicazione principale, che viene eseguita di continuo all'interno del while (si tratta di un semplice collegamento diretto tra i comandi numeric e knob ed i relativi indicatori, per fare in modo che il valore visualizzato sugli indicatori inseguia quello impostato tramite i comandi). La struttura event case intraprende una serie di operazioni in funzione di eventi esterni, che sono asincroni rispetto al ciclo di esecuzione principale. Vediamo adesso come si inserisce un nuovo evento all'interno della struttura:

supponiamo, ad esempio, di voler fare accendere uno dei LED nel caso di variazione del valore del controllo Knob. Per aggiungere un nuovo evento, posizionarsi con il puntatore del mouse sul bordo della struttura, premere il tasto destro, e tra le varie opzioni disponibili, selezionare *Add Event Case...*, come illustrato nella Fig. 4.

Comparirà la finestra di dialogo visibile nella Fig. 5. Per aggiungere un nuovo evento a quelli già esistenti, selezionare l'evento desiderato tra quelli disponibili dalla combinazione dei contenuti delle due sottofinestre *event source* ed *events*. Come si può evincere dai nomi, la finestra *event source* permette di selezionare la sorgente dell'evento (che può essere un qualsiasi elemento del pannello), mentre la finestra *events* permette di selezionare l'evento specifico associato a tale sorgente (come ad esempio un cambiamento di valore per un controllo, o il posizionamento del mouse sopra l'elemento stesso). Ad esempio, per associare un evento al cambiamento di valore del controllo knob, basta selezionare come sorgente il controllo stesso e come evento, l'evento denominato *Value Change*, come è possibile vedere sempre nell'esempio di Fig. 5. Terminata quest'operazione, per aggiungere l'evento alla struttura basta premere il tasto *Add Event*.

Così come fatto per l'aggiunta dell'evento *Knob:Value Change* è possibile associare un evento al passaggio del mouse sopra il controllo numeric, selezionando come sorgente il controllo stesso e come evento *Mouse Move*, presente all'interno della finestra *events* sotto la shortcuts *Mouse*. Una volta aggiunto questo terzo evento (ricordiamo che è presente di default l'evento *timeout*) possiamo procedere. Chiudiamo la finestra *edit events* e torniamo alla finestra di programmazione. Come si può vedere, ora la struttura *event case* contiene i casi aggiunti in precedenza. Adesso, per completare il programma è sufficiente agire sui LED in funzione dei casi della struttura *event case* in modo che il comportamento del programma rispetti le specifiche iniziali e gestire il caso *timeout*. Per prima cosa vediamo come gestire i LED. Vogliamo sempre che in relazione al cambiamento di valore del knob si accenda il LED knob e quando il mouse passa sopra il controllo numeric si accenda

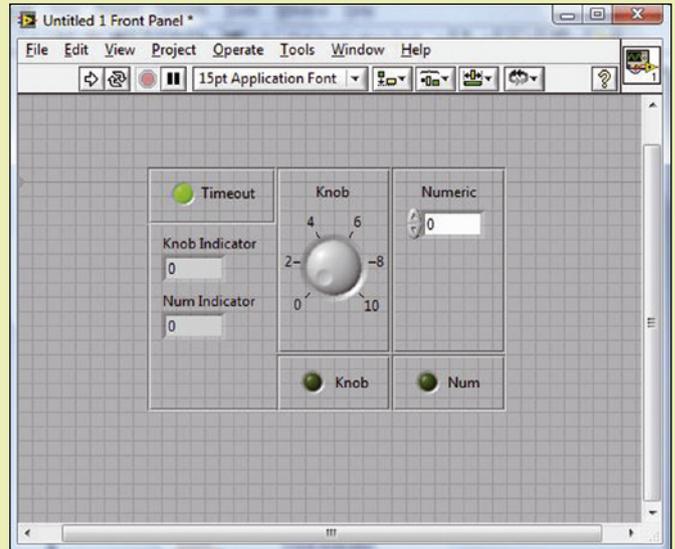


Fig. 2 - Posizionamento di controlli ed indicatori per il programma esempio 2.

il LED Numeric. Naturalmente, oltre all'accensione dei due LED bisogna gestire anche lo spegnimento, altrimenti il programma funzionerebbe bene solo la prima volta e poi i LED resterebbero sempre accesi. Quindi ogni volta che si accende uno dei due LED bisognerà spegnere l'altro. Per accendere e spegnere i LED usiamo due costanti booleane inserite all'interno degli event case generati e portate fuori tramite dei tunnel. Nelle figure 6a e 6b è evidenziato il codice inserito all'interno dei due event case. Vediamo ora come è stato gestito il caso del *timeout*. Come da specifica iniziale, vogliamo che trascorsi 10 secondi di inattività sul pannello frontale il programma vada in stato di *timeout*, arrestandosi in ma-

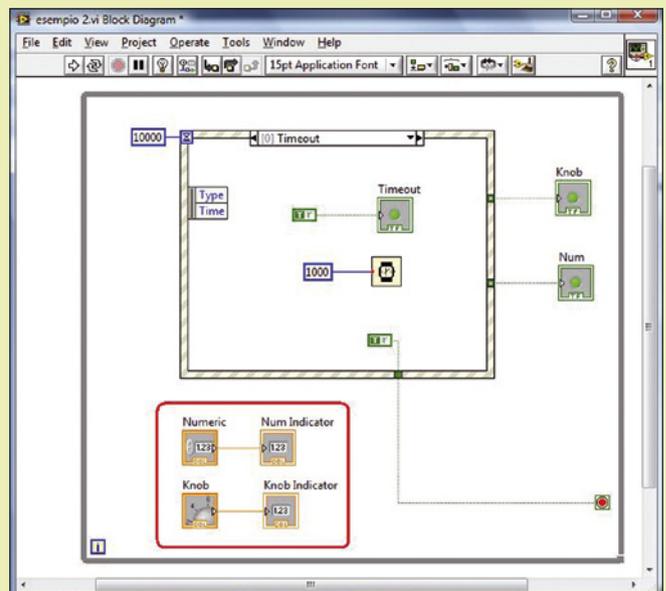


Fig. 3 - Block diagram del programma esempio 2.

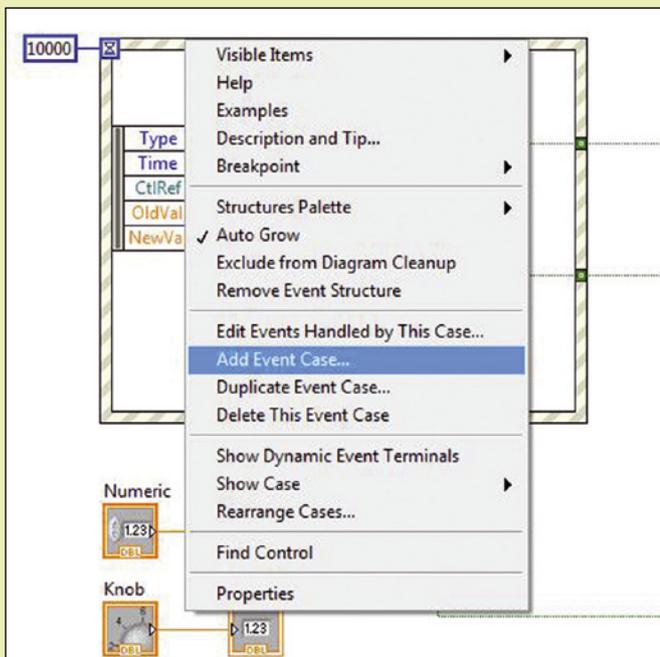


Fig. 4 - Inserimento di un nuovo caso.

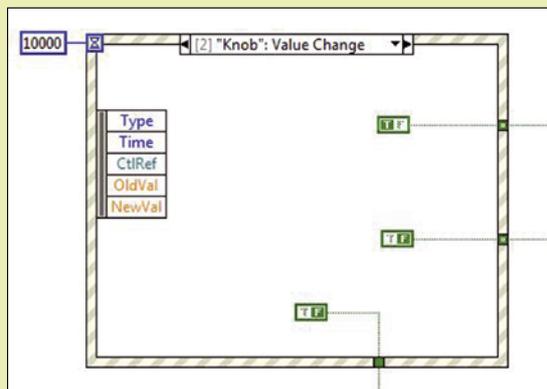


Fig. 4c

niera definitiva dopo un secondo. Per gestire il timeout è sufficiente collegare al simbolo della clessidra, posto nell'angolo in alto a sinistra, una costante numerica col valore relativo al periodo di timeout (in millisecondi). Collegiamo quindi una costante numerica di valore pari a 10.000. Questo fa sì che al trascorrere del tempo indicato, senza attività di alcun tipo sul front panel, la struttura event case entri automaticamente all'interno del caso timeout. A questo punto inseriamo all'interno del caso un ritardo software di 1 secondo (*Programming/Timing/Wait (ms)*), accendiamo (sempre tramite variabile booleana) il LED Timeout, e arrestiamo il programma, collegando una costante True alla Loop Condition di arresto (icona con il simbolo dello stop) del ciclo while principale, come illustrato nella Fig. 7. Per il collegamento al di fuori della event structure ci serviamo del solito tunnel. A questo punto possiamo mandare in esecuzione il programma e verificare l'accensione dei vari LED in corrispondenza delle azioni specifiche. Infine proviamo ad attendere 10 secondi senza compiere nessuna operazione, per verificare che il programma vada in timeout, accendendo il relativo LED ed arrestandosi automaticamente.

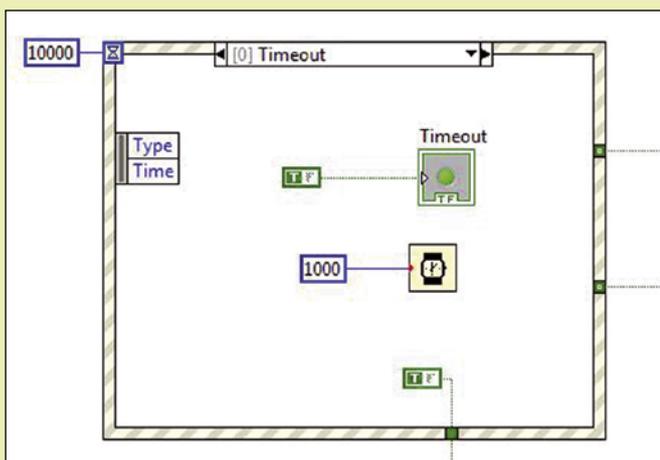


Fig. 4a

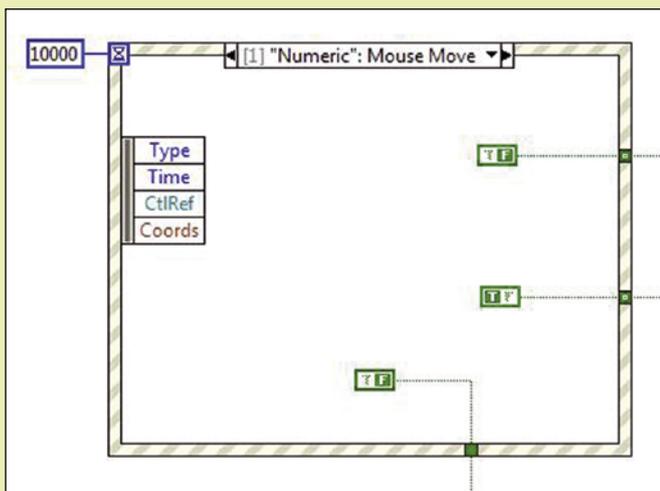


Fig. 4b

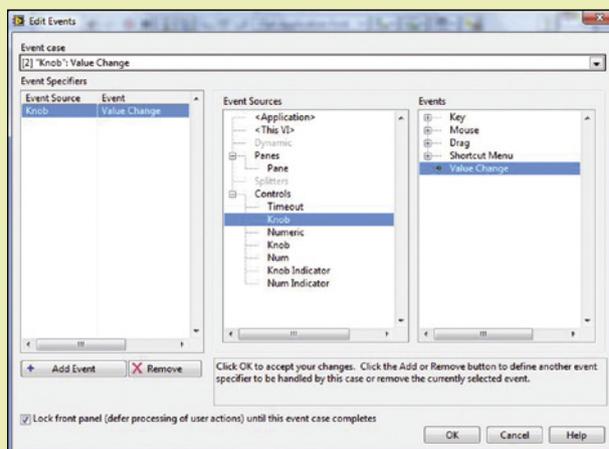


Fig. 5 - Inserimento dell'evento Knob: Value Change.

STRUTTURA SEQUENCE

Un altro esempio di struttura che permette di modificare il comportamento nativo di LabVIEW è la struttura *Sequence*. Come già anticipato nella prima puntata, LabVIEW è un ambiente a flusso di dati e non ad esecuzione sequenziale, come è invece nella maggior parte dei linguaggi di programmazione text-based.

Ciò vuol dire che l'esecuzione del codice di un VI avviene in maniera parallela, all'interno di uno stesso foglio di lavoro.

Un tale modo di operare può essere un vantaggio (si pensi infatti che in questo modo può venire realizzato spontaneamente il multithreading, cosa che invece è molto più difficoltosa da ottenere nei linguaggi classici), ma può essere anche un limite, nel momento in cui si desidera che una certa porzione di codice venga eseguita in maniera sequenziale.

Per ovviare a questo problema si usa la struttura *Sequence*. Di questa tipologia di struttura, in LabVIEW, esistono due versioni: la *stacked sequence* e la *flat sequence*.

Ognuna di queste strutture è frazionata in uno o più frame.

Nella Fig. 8 sono rappresentate entrambe le tipologie.

Una *sequence* (sia essa di tipo *flat* o *stacked*) permette di forzare un'esecuzione sequenziale del codice, ossia il codice contenuto all'interno di un frame viene eseguito per intero prima di passare a quello contenuto nel frame successivo. Per aggiungere una *sequence* all'interno di un block diagram bisogna aprire la control palette e selezionare la *sequence* desiderata all'interno del menù *Structures*.

Quando una *sequence* viene aggiunta è costituita da un singolo frame, per aggiungere altri frame bisogna posizionare il puntatore del mouse sul bordo a forma di pellicola cinematografica e cliccare con il tasto destro del mouse.

Dal menu a tendina che appare a questo punto, bisogna selezionare una delle due voci *Add Frame After* o *Add Frame Before*.

Si noti come i frame della *flat sequence* sono sempre tutti visibili, mentre per visualizzare i frame della *stacked sequence* bisogna utilizzare l'indice di scorrimento posizionato sul bordo in alto al centro, come si faceva per le strutture di tipo *case*.

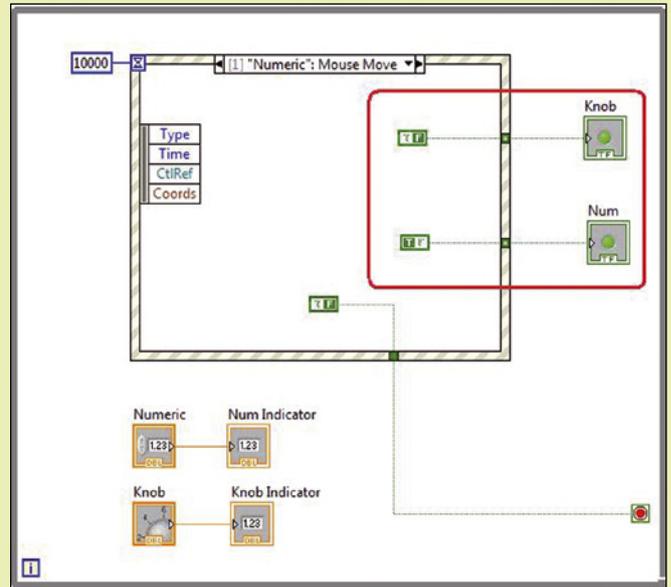


Fig. 6a - Gestione dell'evento Numeric: Mouse Move.

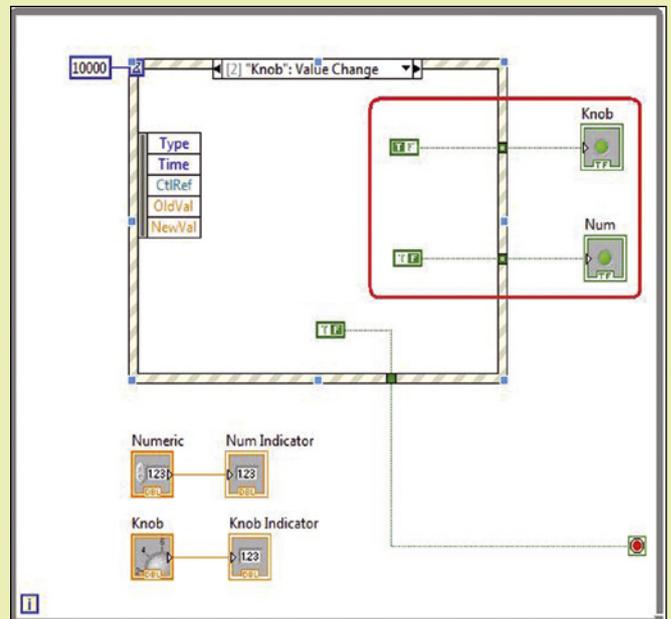


Fig. 6b - Gestione dell'evento Knob: Mouse Move.

FORMULA NODE

Oltre alle strutture di controllo descritte nelle ultime due puntate LabVIEW dispone di un tipo particolare di struttura chiamato *Formula Node*. Il *Formula Node* è un oggetto grafico che funge da contenitore per l'introduzione all'interno di un block diagram di equazioni e codice testuale. Questo tipo di struttura evidenzia in maniera particolare la flessibilità di LabVIEW, infatti permette di introdurre all'interno di un programma sviluppato tramite linguaggio-G, dei frammenti di codice testuale. Nella Fig. 9 è rappresentata l'icona della struttura così come appare sul foglio di lavoro.

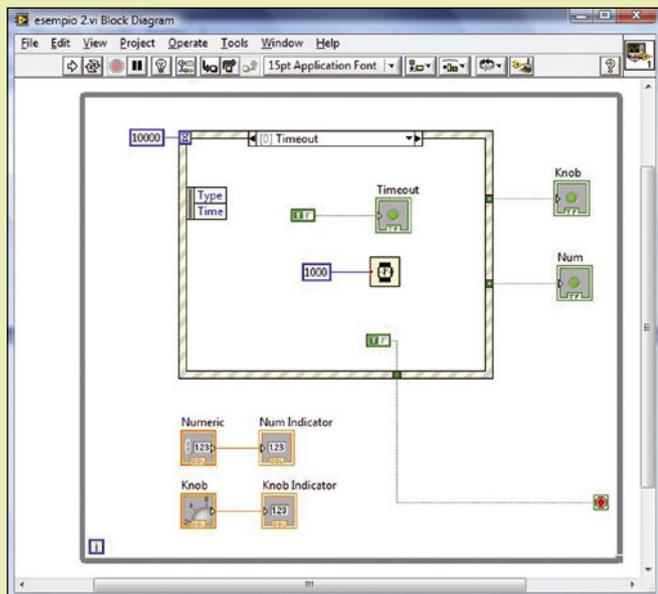


Fig. 7 - Gestione del timeout.

Analizziamo il funzionamento di questa nuova struttura (che può tornare molto utile in diverse situazioni) servendoci di un semplice programma di esempio. Come già fatto in precedenza, creiamo un nuovo VI dal menu principale e salviamolo con il nome *esempio 3*. Posizioniamo sul front panel un chart (*Graph/Waveform Chart*) e due controlli numerici (*Numeric/Numeric Control*), chiamandoli 'm' e 'q'. Ora clicchiamo con il tasto destro sul bordo del grafico e selezioniamo l'opzione *Visible items/Plot legend*, per rendere visibile la legenda del grafico. Posizionandoci sul bordo della Plot Legend, trasciniamo il puntatore del mouse tenendo premuto il tasto sinistro finché le tracce elencate non diventano due. Dopo aver svolto le operazioni descritte, dovreste ottenere un front panel come quello visibile nella **Fig. 10** (la seconda traccia è stata

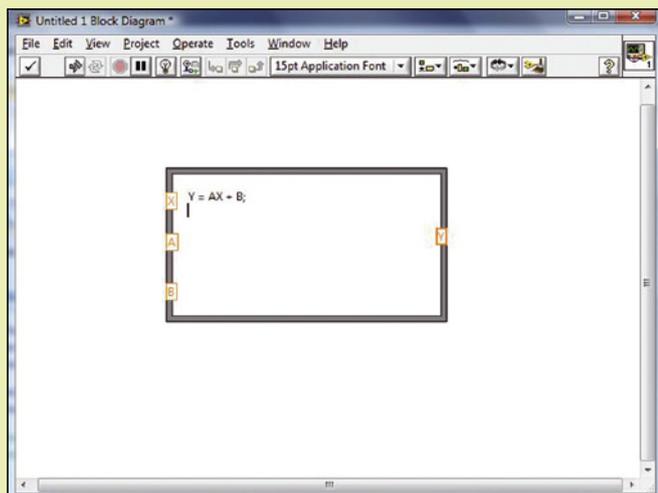


Fig. 9 Icona della struttura Formula Node.

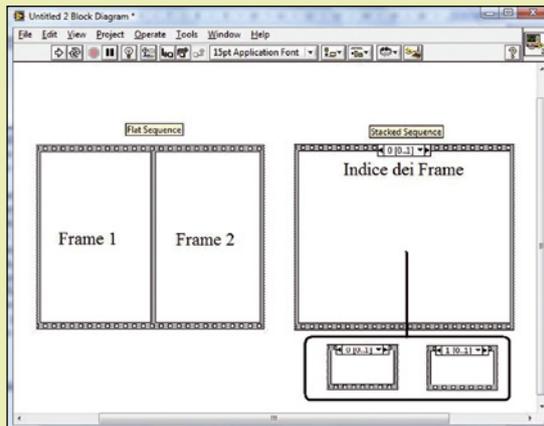


Fig. 8 - Flat Sequence (a sinistra) e Stacked Sequence (a destra).

modificata in modo che sia una linea tratteggiata anziché continua, usando l'opzione relativa alla *Plot Legend*, presente sotto *Line Style*). A questo punto apriamo il block diagram. Sul block diagram riproduciamo lo schema rappresentato nella **Fig. 11** (a questo punto il lettore dovrebbe essere in grado di muoversi più o meno agevolmente all'interno di un foglio di lavoro LabVIEW e dovrebbe aver imparato ad utilizzare la control palette). Ciò che vogliamo ottenere è la stampa a video di un grafico rappresentante una curva ed una retta che la intercetta, con la possibilità di modificare il coefficiente angolare (m) e l'intercetta (q) della retta. Per farlo inseriamo all'interno del formula node l'equazione di una parabola per l'origine ($y = x^2$, che in LabVIEW si esprime usando il doppio asterisco "*" tra base ed esponente) e l'equazione della retta in forma esplicita ($y = m \cdot x + q$). Nei formula node, ogni espressione (sia essa matematica o logica) va terminata con un punto e virgola ";", altrimenti il compilatore restituirà errore. Le due equazioni inserite all'interno del formula node, a questo punto

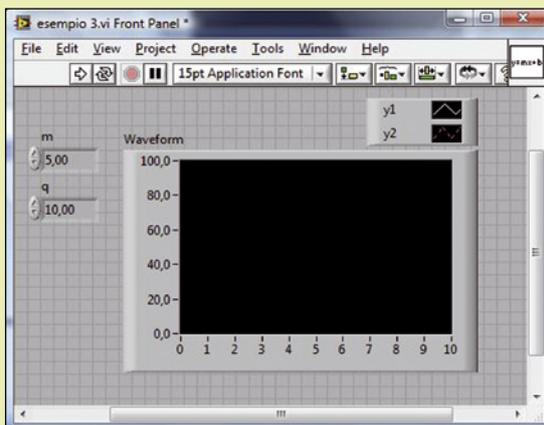


Fig. 10 Front panel del programma esempio 3.

sono le seguenti:

$$y1 = x^{**2};$$

$$y2 = m*x + q;$$

Scrivendole abbiamo implicitamente scelto i nomi della variabili, che sono y1, y2, m, x e q. A questo punto, per proseguire bisogna permettere alle variabili del formula node di comunicare con il resto del VI; in altre parole, bisogna rendere tali variabili, ingressi e uscite. Nel caso specifico, y1 e y2 sono le due uscite, mentre m, x e q sono i tre ingressi.

Per aggiungere ingressi ed uscite al formula node, basta cliccare con il tasto destro del mouse sul bordo della struttura e selezionare una della due opzioni *Add Input* o *Add Output*, come illustrato nella Fig. 12.

Fatto ciò, occorre aggiungere tre ingressi chiamandoli rispettivamente x, m e q e due uscite (y1 e y2). La corrispondenza del nome è sufficiente a correlare ingressi ed uscite dall'interno all'esterno del Formula Node.

A questo punto colleghiamo i due controlli m e q alle rispettive variabili e alla variabile x l'indice del ciclo for, in questo modo cambiando l'indice si avranno retta e curva definite su un diverso numero di punti (nell'esempio inizialmente abbiamo 11, ma questo valore può essere cambiato a piacimento).

Ora non resta che collegare le due uscite y1 e y2 al grafico.

Poiché abbiamo realizzato un grafico multitraccia, per connettere entrambe le uscite all'unico ingresso del grafico usiamo il blocco *Build Array (Array/Build Array)*, che serve a concatenare due o più array monodimensionali o ad appendere array o matrici ad array n-dimensionali. Fatto ciò basta collegare l'uscita del blocco build array al grafico ed il programma è completo.

Lanciate l'esecuzione tramite la freccia bianca e osservate il risultato. Se tutto è stato eseguito correttamente dovrete ottenere un grafico come quello illustrato nella Fig. 13.

Questo semplice esempio mostra le grandi potenzialità di LabVIEW nel generare codice per l'acquisizione e la visualizzazione di dati, in pochi minuti e senza nessuna difficoltà è stato generato un VI in grado di riprodurre un grafico multitraccia, un risultato impensabile usando un ambiente

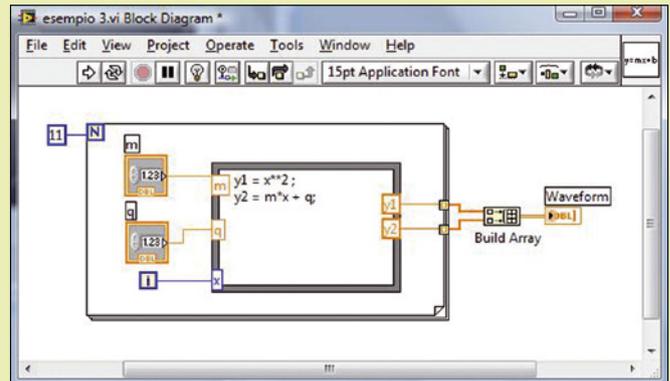


Fig. 11 - Block diagram del programma esempio 3.

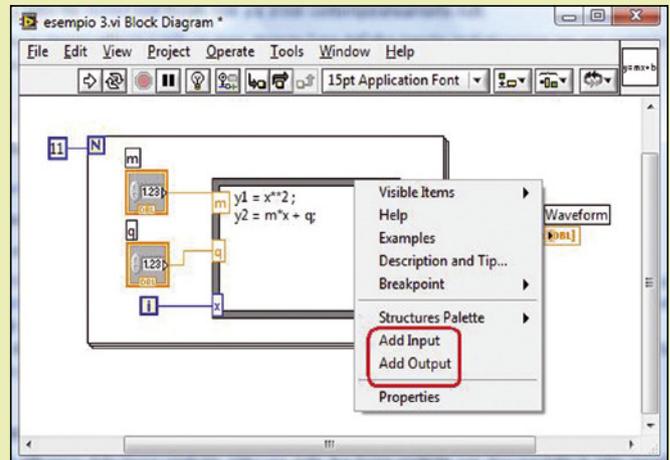


Fig. 12 - Block diagram del programma esempio 3.

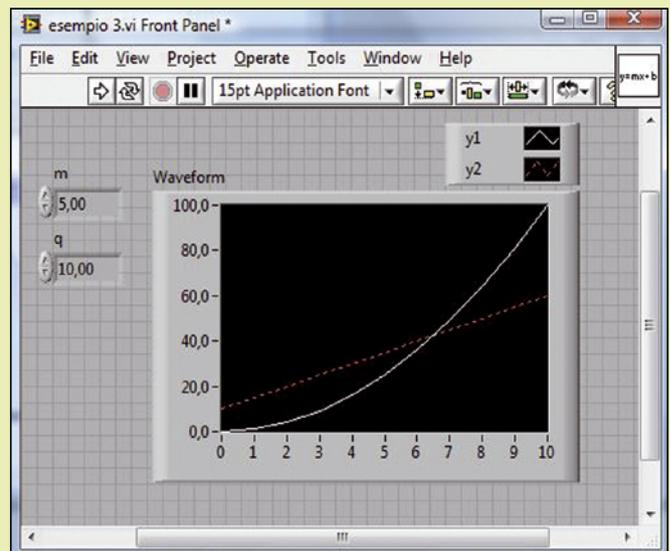
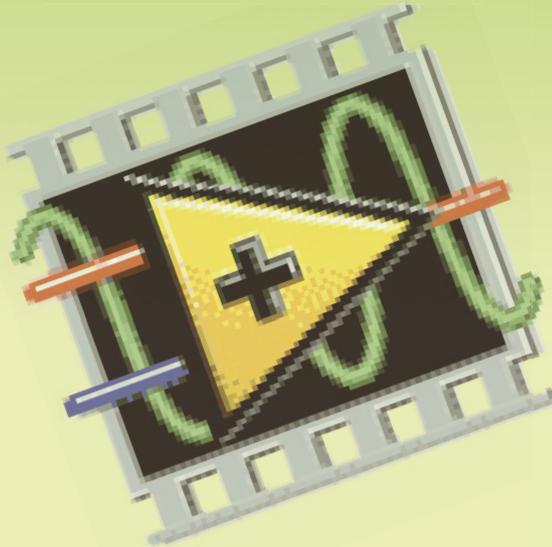
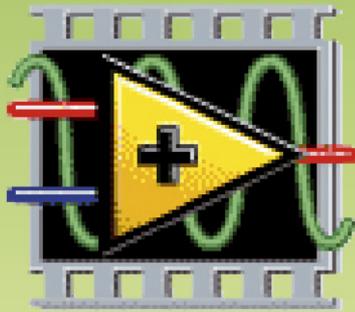


Fig. 13 - Esecuzione del programma esempio 3.

di tipo testuale. Con questa puntata terminiamo la descrizione delle strutture di controllo di LabVIEW. Nella prossima puntata analizzeremo più in dettaglio gli oggetti grafici e ci occuperemo della gestione del File I/O in LabVIEW. ■

Conoscere e usare LabVIEW



Proseguiamo la descrizione della piattaforma software LabVIEW, analizzando gli oggetti grafici, il front panel, oltre al salvataggio e alla lettura dei file.
Quarta Puntata.

di
FRANCESCO
FICILI

LabVIEW

Nelle precedenti puntate del corso abbiamo introdotto l'ambiente di sviluppo, analizzato l'interfaccia, presentato le strutture dati ed esaminato nel dettaglio le strutture di controllo, prendendo sempre maggiore confidenza con la programmazione grafica ed il linguaggio-G. In questa quarta puntata andremo a studiare in profondità altri aspetti dell'ambiente di sviluppo che possono essere significativamente utili nelle nostre applicazioni. Qui verranno analizzati i più importanti oggetti grafici che compongono la relativa control-palette, naturalmente, ed i front panel, che costituiscono l'interfaccia grafica delle nostre ap-

plicazioni. Successivamente analizzeremo in dettaglio come vengono gestiti, in LabVIEW, il salvataggio e la lettura dei file, attraverso alcuni programmi di esempio.

OGGETTI GRAFICI

Nelle precedenti puntate sono stati già introdotti alcuni oggetti grafici (che abbiamo già utilizzato nei primi esempi di programmazione) come LED, pulsanti e grafici. LabVIEW dispone di una notevole varietà di questi oggetti, che permettono di personalizzare a piacimento i pannelli frontali. Vediamo quali sono i più importanti, quali le caratteristiche principali e come



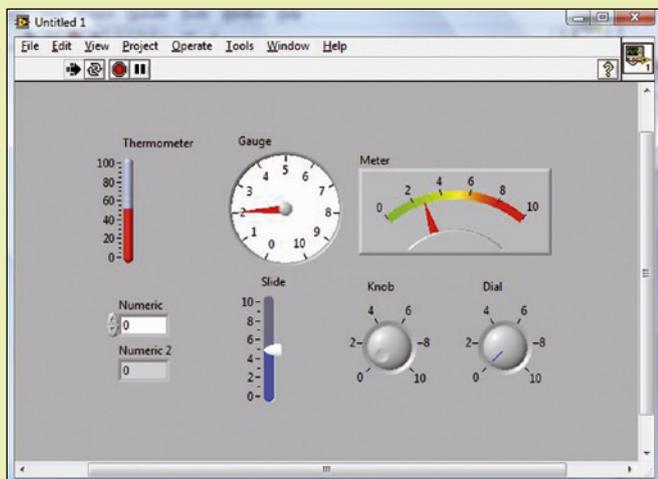


Fig. 1 - Controlli ed indicatori di tipo *Numeric*.

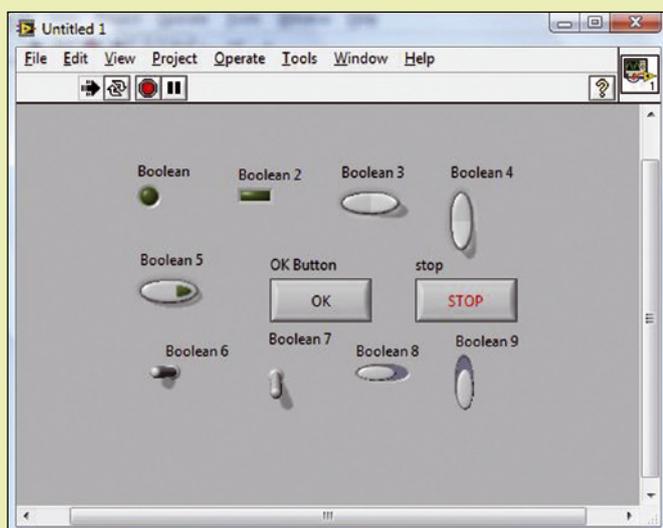


Fig. 2 - Controlli ed indicatori di tipo *Boolean*.

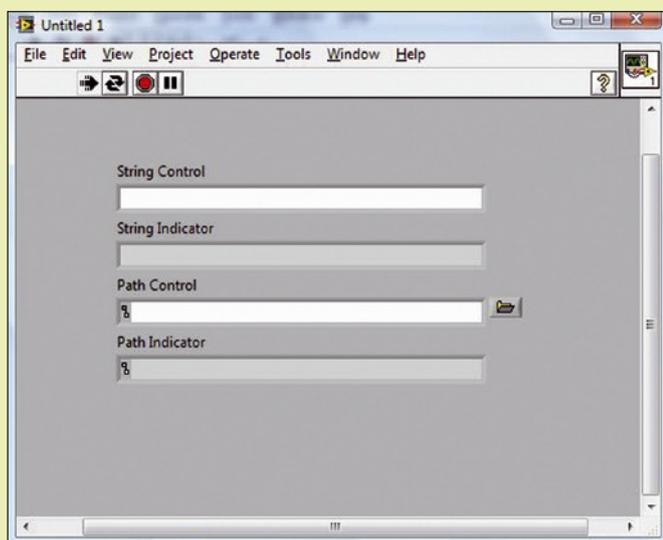


Fig. 3 - Controlli ed indicatori di tipo *String* e di tipo *Path*.

utilizzarli nelle nostre applicazioni. Gli oggetti grafici di LabVIEW si dividono fondamentalmente in due categorie: controlli ed indicatori. Un *controllo* è un oggetto che prende un input dall'operatore che agisce sul pannello e lo trasferisce all'interno del VI, mentre un *indicatore* prende in ingresso un valore dal VI (una variabile o una costante) e lo rende visibile sul front panel. A loro volta, controlli ed indicatori esistono in varie tipologie e sono raggruppati in base al tipo di variabile che possono rappresentare. Vediamo qui di seguito quali sono i più utilizzati in LabVIEW.

Numeric

Questo gruppo contiene controlli ed indicatori numerici; ne fanno parte i controlli e gli indicatori a display, gli slider (barre), knob e dial (manopole), meter e gauge (indicatori a lancetta) ed altri ancora. Sono sempre associati a variabili numeriche, di qualsiasi tipo, quindi numeri sia puri che in virgola mobile. Alcuni dei controlli e indicatori appartenenti a questo gruppo sono riprodotti nella **Fig. 1**.

Boolean

Fanno parte di questo gruppo i controlli e gli indicatori booleani (quindi con solo due valori: vero/falso). Essenzialmente si tratta di vari tipi di LED e di pulsanti, alcuni esempi dei quali sono i LED tondi e quadrati, i pushbutton, gli switch, ecc. La **Fig. 2** elenca alcuni controlli e indicatori appartenenti a questa categoria.

String e Path

Questo gruppo contiene controlli ed indicatori di tipo stringa, e controlli ed indicatori di tipo percorso file (path). I controlli di tipo stringa sono molto utili per l'introduzione dei dati di tipo testo nelle nostre applicazioni, mentre i relativi indicatori sono utili per la visualizzazione dei messaggi. I controlli e gli indicatori di tipo path sono utili per la gestione della lettura e scrittura dei file, anche se in molte circostanze possono essere rimpiazzati dalle finestre di dialogo per la scelta dei path di windows (come vedremo in uno degli esempi successivi). Alcuni controlli e indicatori di questo tipo sono visibili nella **Fig. 3**.

List e Table

In questa categoria troviamo controlli ed

indicatori di tipo tabellare, di varie tipologie: matrici, tabelle, alberi, listbox ed altro ancora. Sono usatissimi per presentare i dati in formato tabellare e anche per la visualizzazione di strutture tipo file system (alberi).

Nella Fig. 4 sono elencati alcuni controlli ed indicatori appartenenti a questa categoria.

Graph e Chart

Molto utilizzati nei programmi LV, i grafici non dispongono di controlli, ma soltanto di indicatori. Anche qui abbiamo una grossa varietà di grafici: chart, graph, XYgraph, digital waveform graph, intensity graph, ecc. Sono molto utilizzati in LabVIEW, perché permettono di presentare agevolmente i dati in forma grafica. Nella Fig. 5 sono riportate alcune tipologie di grafici di LabVIEW.

Gli oggetti descritti nei paragrafi precedenti costituiscono i blocchi fondamentali che vengono utilizzati dai programmatori LabVIEW per lo sviluppo delle loro applicazioni. Esistono ancora altri controlli ed indicatori grafici, ed oggetti grafici di altro tipo, come ad esempio i *tab*, i *container* e gli *elementi decorativi*, ma non ci dilungheremo nella descrizione di tali elementi, poiché quanto descritto in precedenza risulta più che sufficiente per la comprensione dei primi programmi.

FILE I/O IN LABVIEW

Vediamo adesso come è possibile, in LabVIEW, effettuare la scrittura e la lettura di file di testo; queste sono tra le operazioni più importanti lavorando con un sistema di sviluppo software, in quanto ci permettono di leggere e salvare informazioni in maniera permanente, sfruttando uno dei supporti di memorizzazione fissa del computer sul quale lavoriamo (dischi, memorie USB, memorie SD). Per iniziare vediamo un semplice, ma significativo esempio, che permette di salvare una stringa, inserita dall'utente attraverso un controllo di tipo stringa, all'interno di un comune file di testo, il quale può essere aperto, per esempio, con Blocco Note di Windows. Dal menu principale di LabVIEW, apriamo un nuovo VI, chiamandolo *esempio 4*; sul pannello frontale posizioniamo un pulsante generico (Boolean/OK button), nominandolo come *write*, ed un controllo di tipo stringa (String&Path/String control). Posizioniamo i

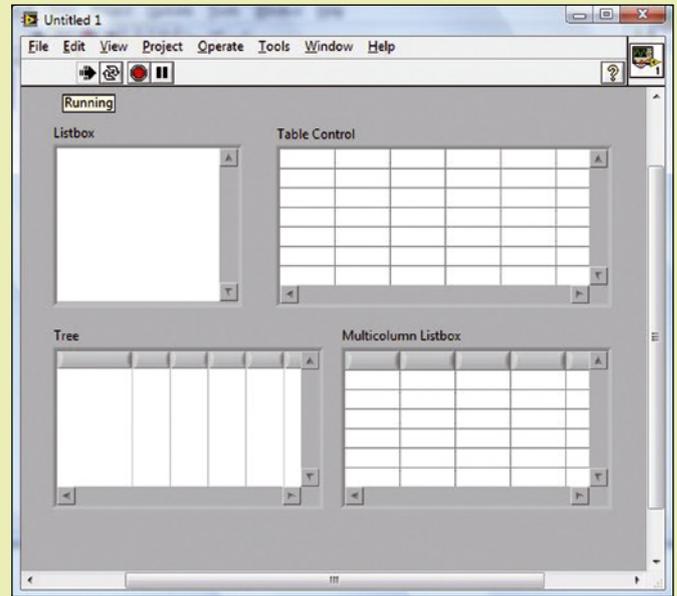


Fig. 4 - Controlli ed indicatori List e Table.

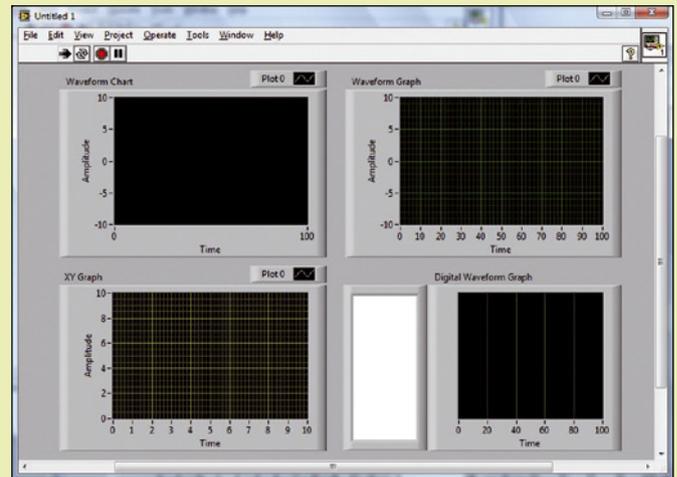


Fig. 5 - Controlli ed indicatori di tipo Graph.

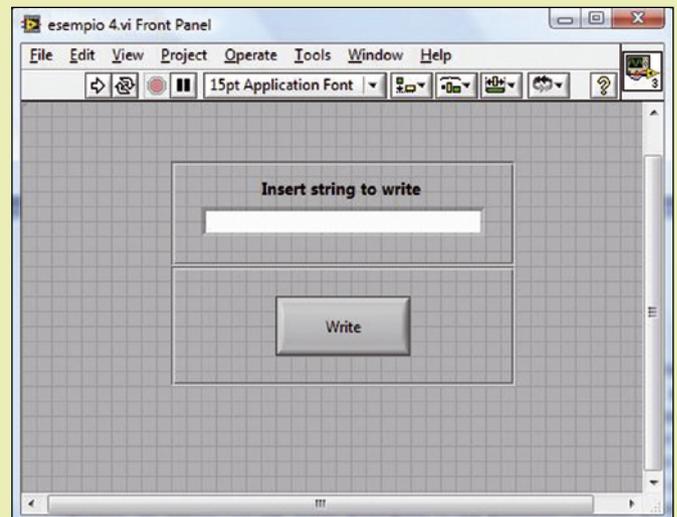


Fig. 6 - Front panel del programma esempio 4.

due elementi come illustrato nella Fig. 6 (sulla finestra sono stati aggiunti alcuni elementi decorativi e una label contenente un'indicazione). Ora spostiamoci sul block diagram. Creiamo una struttura case con, annidata al proprio interno, una struttura sequence con tre frame. La sequence può essere di tipo sia *flat* che *stacked*; nell'esempio in questione è stata usata la *stacked sequence*. Per la manipolazione dei file, LabVIEW dispone di una serie di blocchi specifici, che si trovano un'apposita sezione della control palette, denominata appunto *File I/O* e rappresentata nella Fig. 7. Alcune delle operazioni accessibili tramite i blocchi presenti in questa sezione sono:

- apertura e chiusura di un file;
- scrittura e lettura di un file di testo;
- scrittura e lettura di un file binario.

A questo punto inseriamo al di fuori della struttura case un blocco *Open/Create/Replace File*. Questo blocco permette di creare o di aprire un file creato in lettura o scrittura e necessita di una serie di parametri di configurazione collegati ai vari ingressi di cui è dotato; riassumiamo di seguito i più importanti.

- *Prompt*: è il messaggio che compare in cima alla finestra di dialogo di salvataggio. Nel caso specifico, è stato collegato ad una costante di tipo stringa contenente il messaggio "Scegliere il percorso di salvataggio del file".
- *File path*: indica il percorso di salvataggio del file; se lasciato vuoto, viene automaticamente richiamata una finestra di dialogo di Windows che permette di selezionare manualmente il percorso. Nel nostro programma di esempio, usiamo la finestra di dialogo in modo da poter selezionare il percorso desiderato.
- *Operation*: l'operazione che si desidera intraprendere sul file. Tutte le operazioni possibili sono elencate nella **Tabella 1**. Nel caso specifico è stata scelta l'operazione 3 (*open or create*).
- *Access*: specifica la tipologia di accesso, lettura, scrittura o entrambe, come elencato nella **Tabella 2**. Nel caso specifico è stato scelto l'accesso in lettura e scrittura, anche se sarebbe sufficiente il solo accesso in scrittura.
- *Disable Buffering*: questa opzione permette di disabilitare il buffering e quindi di effettuare un accesso veloce. Se la periferica hardware è

Fig. 7 - Sezione File I/O della control palette.

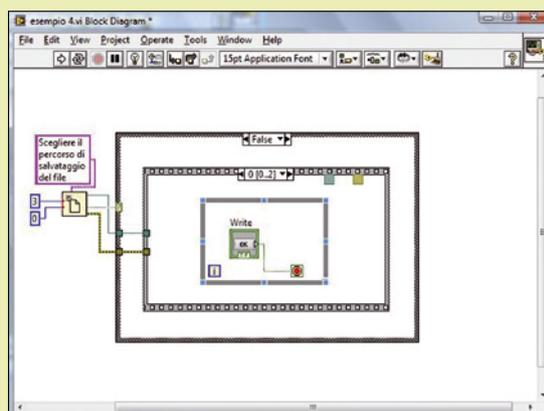
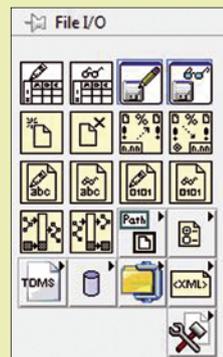


Fig. 8 - Block diagram del programma esempio 4.

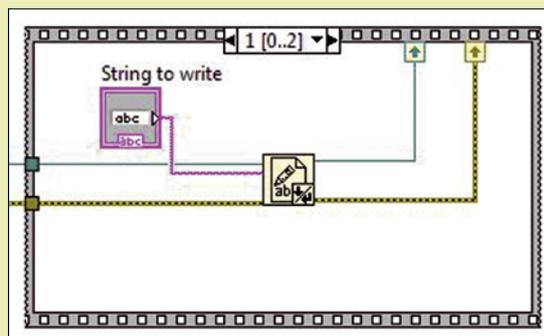


Fig. 9 - Secondo frame della struttura sequence del programma esempio 4.

particolarmente veloce, disabilitare il buffering permette di accelerare il processo di accesso al file. Per impostazione predefinita l'opzione è FALSE.

Oltre ai parametri in ingresso e alla linea di ingresso/uscita per il controllo di errore (che, sebbene colleghiamo, al momento non visualizziamo sul front panel) il blocco presenta due importanti uscite:

- *Refnum out*; è il reference number del file che viene aperto e va collegato agli altri blocchi che intervengono sul file (ad esempio i successivi blocchi di scrittura e chiusura file);
- *Cancelled*: questa uscita, di tipo booleano,

diventa TRUE nel caso in cui, nella finestra di dialogo, venga premuto il tasto *annulla*; è molto utile per gestire tale caso specifico.

Nella Fig. 8 è rappresentato il block diagram del programma *esempio 4*. Come si può vedere, i parametri di ingresso del blocco open file sono, una costante pari a 3 collegata all'ingresso operation (in modo da creare un ex-novo un file se non esistente o aprirne uno esistente, in caso venga selezionato) e una costante pari a 0 collegata all'ingresso access (in modo da garantire l'accesso in lettura e scrittura). Gli altri parametri di ingresso non vengono collegati, in modo da utilizzare i valori predefiniti (in particolare, non collegando alcun valore sull'ingresso file path, viene automaticamente richiamata la finestra di dialogo di Windows per selezionare il percorso del file). Poi, viene utilizzata l'uscita booleana *cancelled* come variabile di controllo della struttura case che contiene il resto del programma. All'interno del caso FALSE viene inserito il resto del codice, mentre il caso TRUE viene lasciato vuoto. Quest'operazione fa in modo che, nel caso in cui venga premuto il tasto annulla sulla finestra di dialogo, il pro-

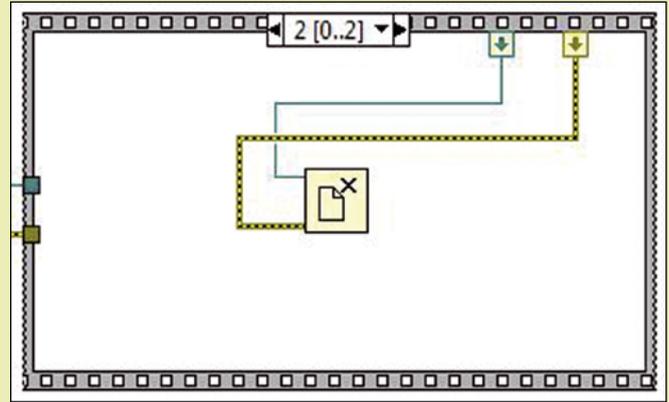


Fig. 10 - Terzo frame della struttura sequence del programma esempio 4.

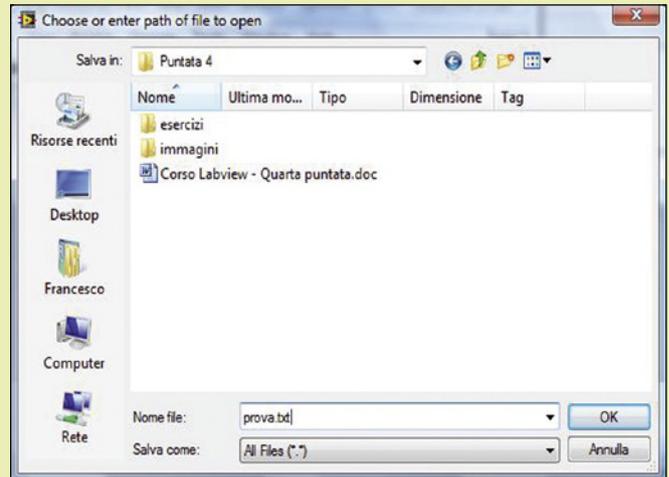


Fig. 11 - Finestra di dialogo per la selezione del percorso di salvataggio del file di testo.

0	open (default)—Opens an existing file. Error 7 occurs if the file cannot be found.
1	replace —Replaces an existing file by opening the file and setting its end of file to 0.
2	create —Creates a new file. Error 10 occurs if the file already exists.
3	open or create —Opens an existing file or creates a new file if one does not exist.
4	replace or create —Creates a new file or replaces a file if it exists. This VI replaces a file by opening the file and setting its end of file to 0.
5	replace or create with confirmation —Creates a new file or replaces a file if it exists and you give permission. This VI replaces a file by opening the file and setting its end of file to 0.

Tabella 1 - Opzioni relative all'ingresso operation del blocco Open/Create/Replace File. Questo ingresso permette stabilire come verrà aperto il file.

0	read/write
1	read-only
2	write-only

Tabella 2 - Opzioni relative all'ingresso access del blocco Open/Create/Replace File. Questo ingresso permette di configurare le opzioni di accesso al file: se in lettura, in scrittura o in entrambe.

gramma esca dall'esecuzione (caso TRUE della struttura case). Se così non fosse, il programma cercherebbe di scrivere su un file del quale non viene fornito il percorso, generando un errore. Diamo adesso uno sguardo al codice contenuto all'interno della struttura sequence, analizzando un frame alla volta. Il primo frame è un semplice ciclo while vincolato al comando booleano write; questo frammento di codice grafico realizza un ciclo di attesa (verrà eseguito un loop vuoto fino alla pressione del tasto write). Questo ciclo di attesa permette di inserire all'interno del controllo di tipo stringa la stringa che si desidera venga salvata sul file. Passiamo ora ad analizzare il secondo frame, riportato nella Fig. 9. In esso viene effettivamente eseguito il codice che salva la stringa su file. Per eseguire questa operazione si usa il blocco *write to text file*, presente sempre nella control palette *File I/O*. Il blocco presenta quattro ingressi e tre uscite. L'ingresso *file* individua il percorso del file da scrivere, ma avendo

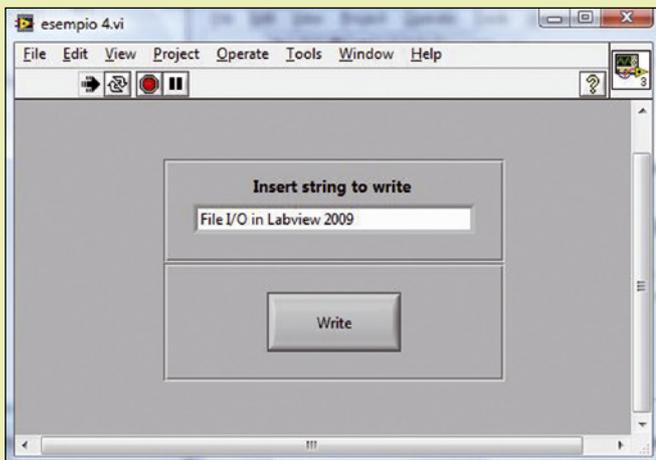


Fig. 12 - Esecuzione del programma esempio 4.

già aperto il file tramite il precedente blocco *open file*, può essere semplicemente passato il reference number del file (portato all'interno della sequence tramite un tunnel). Se l'ingresso viene lasciato libero, anche in questo caso viene richiamata la finestra di dialogo di Windows, quindi in alcuni casi il blocco può funzionare anche in stand-alone, senza che sia necessario il blocco *open file*. L'ingresso *prompt* serve solo nel caso venga usata la finestra di dialogo, quindi in questo esempio perde di significato. L'ingresso *text* costituisce l'insieme di dati che verranno scritti su file dal blocco e deve necessariamente essere di tipo string, quindi se si desidera salvare dei valori numerici questi dovranno essere opportunamente convertiti. Infine, vengono cablate anche le due linee di errore. Si noti che, per trasferire il refnum e la linea di errore di uscita al blocco presente nel frame successivo, non può essere utilizzato un tunnel come è stato fatto per i dati in ingresso, ma bisogna necessariamente usare un *sequence local*. Il *sequence local* è un apposito oggetto

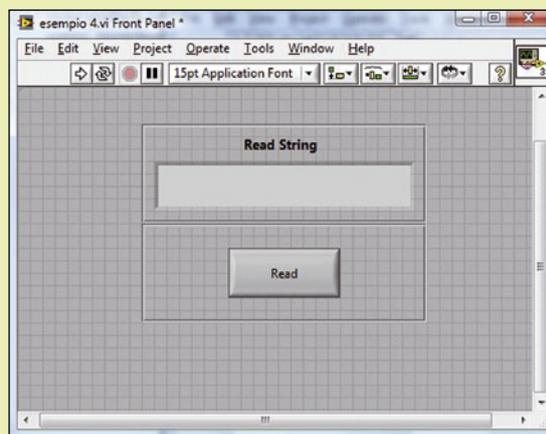


Fig. 14 - Pannello frontale del programma esempio 5

che consente il passaggio di dati tra i frame di una struttura *stacked sequence*. Per inserire un *sequence local* bisogna posizionarsi sul bordo della struttura (quello a forma di pellicola cinematografica), premere il tasto destro del mouse e selezionare l'opzione *add sequence local*. Passiamo adesso al terzo ed ultimo frame, rappresentato nella Fig. 10, nel quale viene chiuso il file. Per la chiusura del file si usa il blocco *close file*, che semplicemente prende in ingresso il refnum del file da chiudere e la linea di errore, chiude il file e restituisce il path associato al file che è stato chiuso.

A questo punto il programma è completo: possiamo lanciarlo in debug (premendo la freccia bianca) ed osservare cosa succede. Appena viene lanciato il programma, compare immediatamente la finestra di dialogo (Fig. 11) che permette di selezionare il percorso di salvataggio. Qui dovete selezionare un percorso qualsiasi ed un nome file (non importa se il file non esiste, nel caso ne verrà creato uno ex-novo), con estensione .txt. A questo punto potete inse-

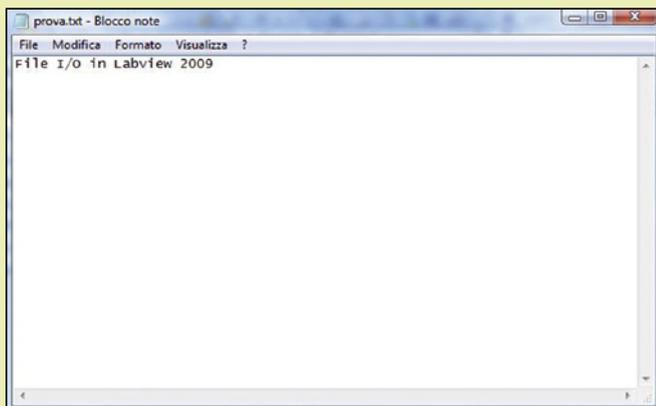


Fig. 13 - File di testo salvato dal programma esempio 4.

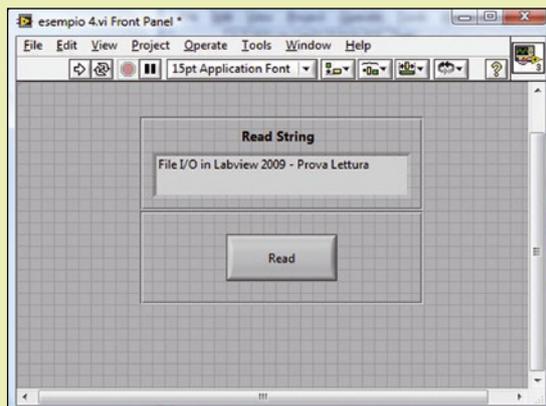


Fig. 15 - Risultato dell'esecuzione del programma esempio 5.

rire una stringa all'interno del controllo string, come illustrato nella Fig. 12, e premere il tasto write, per far sì che la stringa venga salvata sul file precedentemente selezionato. Il risultato sarà un file di testo come quello visibile nella Fig. 13. Questo semplice esempio mostra come viene gestito il file I/O in LabVIEW per il salvataggio di file di testo. Sfruttando la stessa logica è possibile salvare dati numerici (opportunitamente convertiti) provenienti, ad esempio, da sensori acquisiti da opportune schede di acquisizione dati. Si noti come non sia stato necessario utilizzare un *common dialog control* (ad esempio come viene fatto in Visual Basic) per la gestione delle finestre di dialogo; qui è stato l'ambiente a gestire tale aspetto (generalmente piuttosto convenzionale) per noi, velocizzando la stesura del codice.

Lo stesso programma, con qualche piccola modifica, può essere utilizzato per dimostrare come viene gestita la lettura dei file di testo in LabVIEW; ad esempio si può arrivare a realizzare un programma che visualizza su un indicatore il contenuto di un file di testo precedentemente salvato. Adesso salviamo nuovamente il programma appena scritto, rinominandolo *esempio 5*. Effettuiamo le seguenti modifiche sul pannello frontale:

- cambiamo il nome del pulsante da *write* a *read*;
- sostituiamo il controllo di tipo string in uno string indicator;
- modifichiamo i commenti sul pannello frontale come riportato nella Fig. 14.

A questo punto spostiamoci sul block diagram ed andiamo a modificare il blocco presente nel frame numero 2 della stacked sequence. Cancelliamo il blocco write text file e sostituiamolo con un read text file. Ripristiniamo i precedenti collegamenti delle linee di refnum e di errore e colleghiamo l'uscita text all'indicatore di tipo string. Fatto ciò, il programma è pronto per l'esecuzione. Il risultato è visualizzato nella Fig. 15.

Concludendo, in questa quarta puntata vi abbiamo descritto alcune nuove potenzialità dell'ambiente LabVIEW, che utilizzeremo nelle puntate successive per lo sviluppo di un software di acquisizione dati, facente uso di un multifunction DAQ della NI come scheda di acquisizione.

Il combo-box

Un interessante controllo, appartenente alla famiglia dei controlli di tipo stringa, è il combo-box: esso permette di realizzare menu a scelta multipla, che trovano uso in diverse occasioni, quando vengono realizzati software di interfaccia uomo-macchina. Nella Fig. 16 è riportato l'aspetto grafico del controllo, così come appare sul front panel.

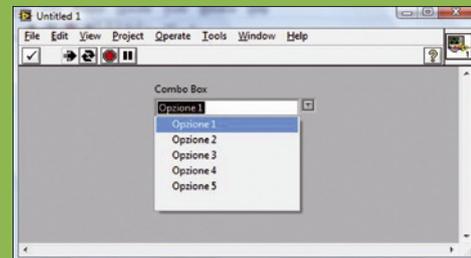


Fig. 16 - Aspetto del controllo combo-box.

Come si vede dalla figura, premendo la freccia collocata alla destra del controllo, si apre un drop-down menu che elenca tutte le opzioni disponibili, e che possono essere selezionate semplicemente con un click del mouse. Per aggiungere elementi all'interno del drop-down menu è sufficiente cliccare con il tasto destro del mouse sul controllo, e selezionare l'opzione "edit items...". Comparirà la finestra visibile nella Fig. 17. Come si può vedere, è possibile inserire, togliere, spostare ed editare elementi a piacimento. Gli elementi sono sempre di tipo stringa, e possono essere utilizzati, sul block diagram, come elementi di selezione per una struttura di tipo case, come evidenziato nella Fig. 18.

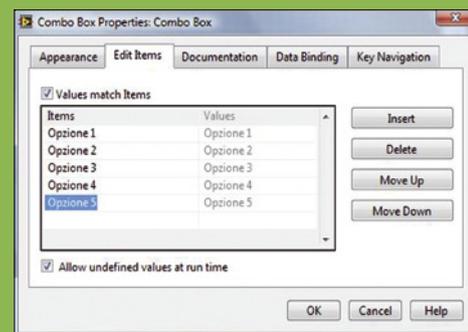


Fig. 17 - Finestra edit items di un generico combo box.

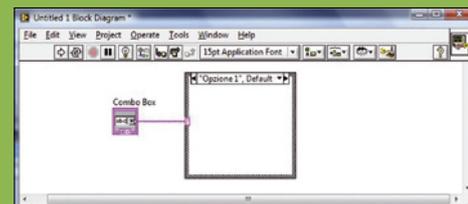
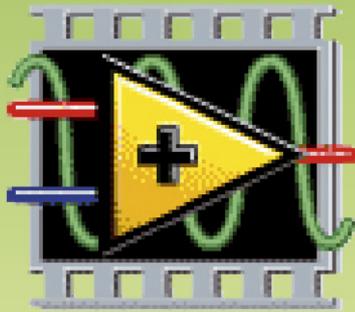
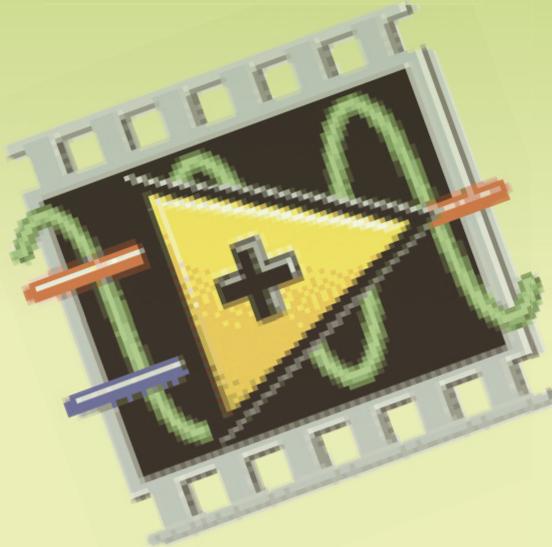


Fig. 18 - Uso di un combo-box come elemento di selezione di una struttura case.

Conoscere e usare LabVIEW



Studiamo una delle caratteristiche più apprezzate della piattaforma LabVIEW: l'interfacciamento ed il controllo di strumentazione esterna. Scopriremo il DAQ Assistant ed altro ancora. Quinta Puntata.



di
FRANCESCO
FICILI

LabVIEW

Nelle scorse puntate abbiamo iniziato a muovere i primi passi in LabVIEW, presentando le strutture dati e le strutture di controllo. In esse è stata descritta l'interfaccia dell'ambiente ed abbiamo analizzato in dettaglio alcuni aspetti della programmazione grafica tramite il linguaggio-G; in particolare, nelle ultime puntate abbiamo visto anche come sia possibile utilizzare i grafici ed interagire con il file system del PC, scrivendo e leggendo file di testo. Tutte le informazioni apprese saranno utilizzate per la scrittura di un programma di acquisizione dati, che fa uso di una scheda esterna per leggere segnali provenienti dal

mondo fisico. Questo ci consentirà di esplorare una delle caratteristiche più apprezzate di LabVIEW, ossia l'interfacciamento ed il controllo di strumentazione esterna.

PROTOCOLLI DI COMUNICAZIONE IN LABVIEW

Iniziamo con l'analizzare quali sono le tipologie di interfacce disponibili per il collegamento di un device esterno (una scheda di acquisizione dati) al PC, nel caso si usi LabVIEW come ambiente di sviluppo del software. Naturalmente ogni bus ha le sue caratteristiche e la scelta di quello da utilizzare va considerata attentamente in funzione dell'ap-

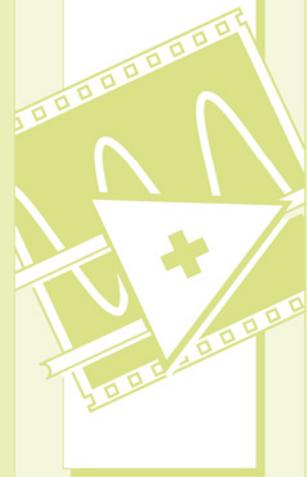




Fig. 1 - L'Instruments I/O di LabVIEW: fornisce un efficace strumento per il rapido sviluppo di applicazioni di misura.



Fig. 2 - Cartuccia con ingressi analogici e carrier USB, NI USB-9162.



Fig. 3 - Scheda NI ENET-9206.

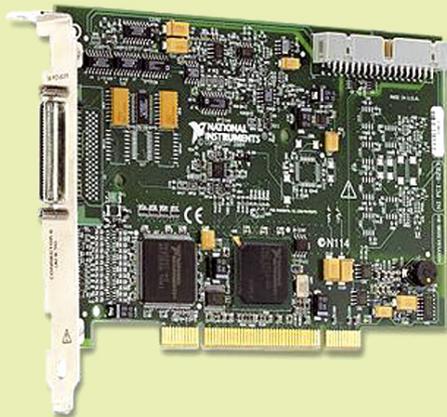


Fig. 4 - Scheda NI PCI-6221.

plicazione finale. LabVIEW consente di gestire diversi protocolli di comunicazione, sia cablati che wireless, sia point-to-point che relativi a configurazioni più complesse (bus, reti, alberi). Oltre ai protocolli di comunicazione tipici delle applicazioni industriali, vengono forniti driver per la gestione di protocolli più evoluti come ad esempio l'SMTP, che è quello utilizzato per l'invio di e-mail.

Di seguito elenchiamo i principali protocolli di comunicazione gestiti in LabVIEW, a cominciare da quelli "wired" (a collegamento cablato).

- **Seriale:** si tratta del protocollo usato dalle porte seriali del PC (la più comune è la RS-232, usata per le vecchie stampanti e i modem, prima dell'introduzione dell'USB). Questo tipo di interfaccia non è più molto utilizzato e tende sempre più a scomparire dai PC. Ciononostante i driver utilizzati per la sua gestione sono ancora attuali, in quanto utilizzati (per via della loro semplicità) per la gestione delle periferiche USB usate in emulazione seriale (classe USB CDC).
- **GPIO (IEEE-488):** il General Purpose Interface Bus (noto anche come IEEE-488) è un bus per l'interconnessione ed il controllo di strumentazione elettronica. Discende da uno standard proprietario della Hewlett-Packard, che lo aveva implementato per consentire l'interfacciamento verso i PC dei propri strumenti di misura. La sua caratteristica è quella di utilizzare un collegamento daisy-chain (ossia in serie, con il segnale passato da un dispositivo all'altro) per collegare fino ad un massimo di 15 dispositivi ad un'unica porta su PC. Avendo avuto una buona diffusione come standard intorno agli anni '80 del secolo scorso, è ancora piuttosto utilizzato.
- **USB:** l'Universal Serial Bus è un protocollo seriale usato per l'interfacciamento di periferiche di vario tipo al PC. Pur non essendo nato per applicazioni di misura, le sue caratteristiche di economicità, semplicità e larghezza di banda, hanno creato una certa diffusione anche in quest'ambito, principalmente per dispositivi low-cost e di prestazioni non elevate. National Instruments produce diversi dispositivi USB (denominati NI USB) che spaziano dalle schede low-cost (ad esempio la famiglia NI USB-60xx) fino a quelle a prestazioni medio-alte.
- **Ethernet:** lo standard nato nel 1973 per

l'implementazione di reti LAN ha trovato un notevole campo di applicazione anche nel settore delle misure elettroniche e del controllo industriale. NI produce una gamma di DAQ ethernet (NI ENET) che permettono l'acquisizione di segnali di vario tipo. Le schede sono costituite da un carrier (che ospita solo l'interfaccia) sul quale si possono connettere diversi tipi di cartucce. La modularità del sistema permette di riutilizzare i carrier e, nel caso sorgano nuove esigenze di misura, cambiare solo la cartuccia.

- PCI e PCI Express: le due ben note interfacce per l'interconnessione delle più svariate periferiche sulle motherboard dei PC sono utilizzate da National Instruments per l'implementazione di schede sia low-cost che ad elevate prestazioni.
- PC Card: generalmente indicato come PCMCIA (Personal Computer Memory Card Interface Association) è uno standard di interconnessione per PC portatili, il cui scopo all'atto della sua creazione era quello di estendere le funzionalità dei PC portatili tramite l'interconnessione di periferiche esterne. Al giorno d'oggi lo standard soffre molto la concorrenza dell'USB.
- PXI: è un bus proprietario NI nato nel 1997. Deriva dal più noto standard PCI (il suo acronimo vuol dire infatti Pci eXtensions for Instrumentation) e si distingue per le elevate prestazioni e la grande robustezza ed affidabilità, che lo hanno reso estremamente adatto all'impiego nel settore dell'automazione industriale. NI commercializza una vasta gamma di schede PXI, ed ha anche creato un'estensione basata su PCI Express, denominata PXI Express.

Di seguito riportiamo, adesso, i principali protocolli wireless gestiti da LabVIEW.

- Bluetooth: tramite LabVIEW è possibile gestire il protocollo Bluetooth sia come server che come client, sfruttando le interfacce commerciali che comunemente si trovano nei moderni Personal Computer.
- Wi-Fi: il protocollo, noto come IEEE 802.11, è gestito all'interno dell'ambiente LabVIEW e per esso è disponibile una serie di driver. Inoltre NI commercializza una famiglia di schede di acquisizione dati (per vari tipi di sensori) note come NI WLS.

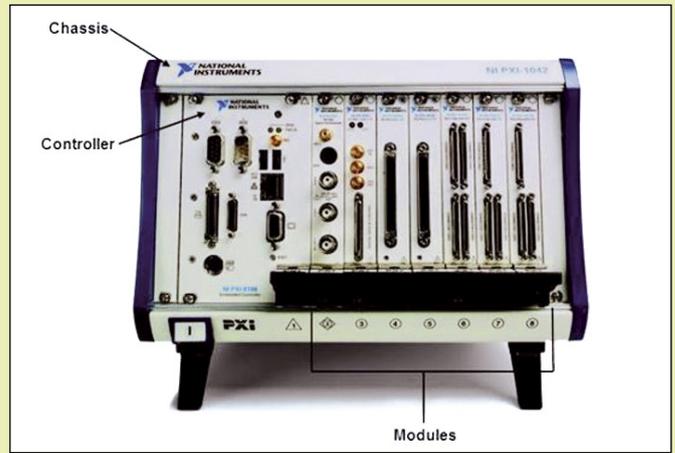


Fig. 5 - Cestello PXI con schede di diverso tipo. Questa soluzione proprietaria NI è quella che offre la banda più larga (si arriva ad alcuni GS/s).



Fig. 6 - LabVIEW è in grado di gestire una grande varietà di bus di comunicazione, dai più economici e commerciali (USB, RS232) fino a quelli proprietari ed altamente professionali (PXI, PXI Express).

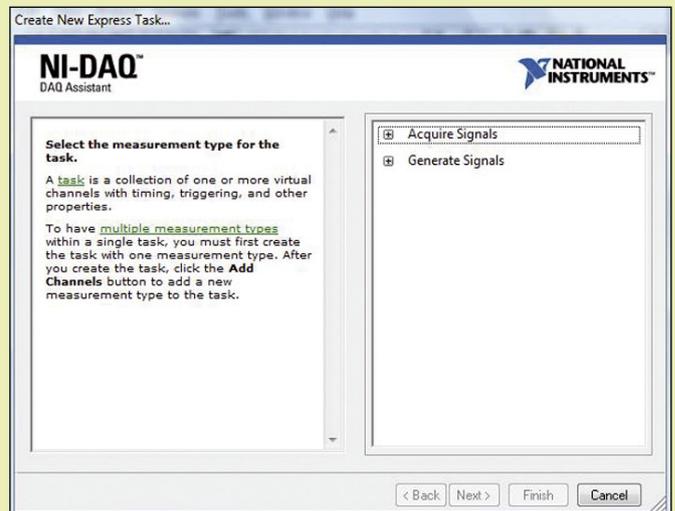


Fig. 7 - Generazione di un task tramite DAQ Assistant.

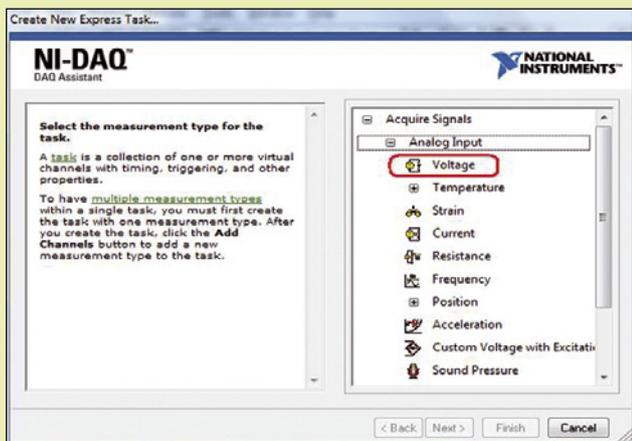


Fig. 8 - Selezione della tipologia di ingresso.

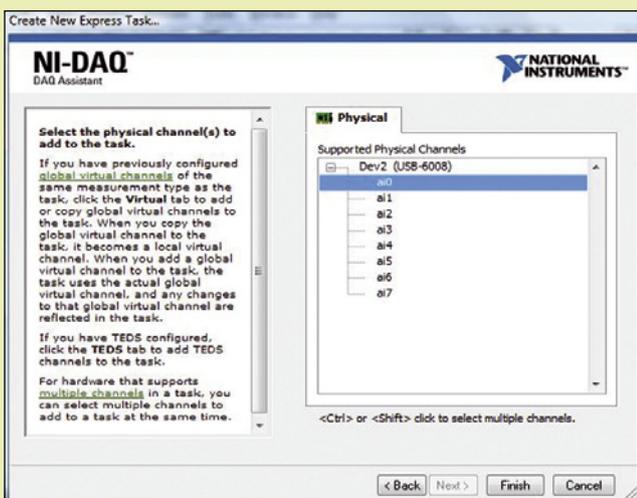


Fig. 9 - Scelta dei canali fisici.

Naturalmente questa lista è molto generale, dato che la quantità di driver presenti all'interno dell'ambiente è notevole ed in costante aggiornamento. National Instruments ha uniformato i pacchetti di driver forniti a supporto dei suoi dispositivi, creando i DAQ Assistant, i quali sono una collezione di driver e wizard che semplificano in maniera notevole l'interfaciamento delle varie schede (dotate di interfacce diverse) al PC, permettendo allo sviluppatore di concentrarsi sull'applicazione in sé.

IL DAQ ASSISTANT

National Instruments, con più di 50 milioni di canali di I/O venduti negli ultimi 10 anni, si pone in una posizione di leadership nel campo dell'acquisizione dati PC-based. L'enorme varietà di bus di interfaccia tra PC e scheda può, però, rendere complesso lo sviluppo della parte di software che deve gestire il dialogo tra

l'applicativo e la scheda; ecco perché National Instruments ha sviluppato una serie di strumenti software che semplificano notevolmente il lavoro del programmatore. Uno di questi è, appunto, il DAQ Assistant: si tratta di un modulo che permette di creare, editare ed infine eseguire, un task, sfruttando la collezione di driver contenuti all'interno del pacchetto NI-DAQmx.

A questo punto è importante chiarire il concetto di task in LabVIEW: un task è una collezione di uno o più canali virtuali caratterizzati da una serie di parametri specifici (frequenza di campionamento, triggering ed altro). In sostanza, il task è l'interfaccia tra la scheda di acquisizione ed il software di alto livello che gira sul PC.

La caratteristica dei task è di poter operare simultaneamente, consentendo quindi di acquisire dati da più periferiche in contemporanea, senza essere vincolati ad utilizzarle una alla volta. La configurazione dei task può avvenire anche utilizzando strumenti diversi dal DAQ Assistant, che però rimane il metodo più veloce per chi ha iniziato da poco ad usare LabVIEW. Da notare che affinché il DAQ Assistant funzioni correttamente è necessario installare il pacchetto di driver NI-DAQmx (<http://joule.ni.com/nidu/cds/view/p/id/1614/lang/en>). Questo pacchetto viene costantemente aggiornato dalla National Instruments (siamo alla versione 9.1) e contiene una grande varietà di driver per schede di acquisizione dati e controllo NI. Una volta installati tutti i driver sarà possibile accedere alle funzionalità del modulo DAQ Assistant. Procediamo, a questo punto, con un semplice esempio di configurazione di un multifunction DAQ operante su bus USB: la scheda NI USB-6008, che verrà utilizzata anche come hardware per il prossimo progetto pratico. Creiamo un nuovo VI, dalla schermata principale di LabVIEW e salviamolo nominandolo esempio 6.vi.

Spostiamoci sul block diagram, nel quale inseriremo un DAQ Assistant.

Per configurare un task di acquisizione tramite DAQ Assistant bisogna posizionare il VI all'interno di un foglio di lavoro LabVIEW. Il VI (da notare che si tratta di un VI di tipo express, riconoscibile dalla colorazione azzurrina) si trova all'interno del percorso *Express/Input/DAQ Assist*. Una volta posizionato, si aprirà un

wizard di configurazione come quello riportato nella Fig. 7.

Come si può vedere, è possibile effettuare due scelte: acquisizione di segnali o generazione di un output. Creiamo un task che ci permetta di acquisire un segnale analogico in tensione, da utilizzare come task di acquisizione generico. Espandiamo quindi la sezione *Acquire Signal* e a seguire la sezione *Analog Input*. A questo punto ci troviamo di fronte ad una serie di possibili ingressi, come evidenziato nella Fig. 8: si va dalle termocoppie agli accelerometri, passando attraverso molte altre tipologie di sensori, come gli *strain gauge* e i sensori di pressione sonora. Scegliamo come tipologia di input un ingresso in tensione generico (indicato con il termine *Voltage*).

Premendo il tasto *Next* si passa alla schermata successiva del wizard. A questo punto il DAQ Assistant passa alla rilevazione dell'hardware collegato al PC.

Come si può vedere dalla Fig. 9, viene rilevata la scheda NI USB-6008 (che è anche automaticamente battezzata con il nome simbolico "Dev2") e vengono elencati tutti gli 8 canali analogici d'ingresso disponibili. A questo punto è possibile selezionare 1 o più canali (per selezionare più canali può essere usato il mouse tenendo premuto il tasto sinistro e trascinando il cursore) e premendo il tasto *Finish* si passerà a definire gli ultimi parametri per la generazione del task.

Si noti che, avendo più interfacce collegate al PC comparirebbero più schede e canali selezionabili, tuttavia in questa puntata non ci occuperemo ancora della gestione multipla di schede di acquisizione, dato che le nostre attuali necessità sono pienamente soddisfatte dall'uso di una singola scheda.

Selezioniamo il canale analogico *ai0* (come illustrato sempre nella Fig. 9) e procediamo. A questo il wizard farà comparire automaticamente la schermata di configurazione del VI Express (si noti che tale schermata è sempre accessibile con il doppio click sul VI, in modo da poter cambiare in ogni momento i parametri di configurazione). La schermata è riportata nella Fig. 10. È visibile una serie di parametri che devono essere impostati per ottenere un determinato comportamento del VI. Tali parametri sono:

- range d'ingresso del segnale (*Signal Input*

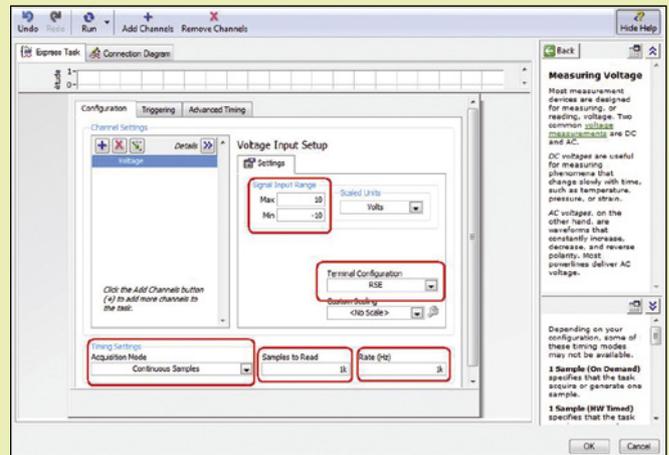


Fig. 10 - Schermata di configurazione.

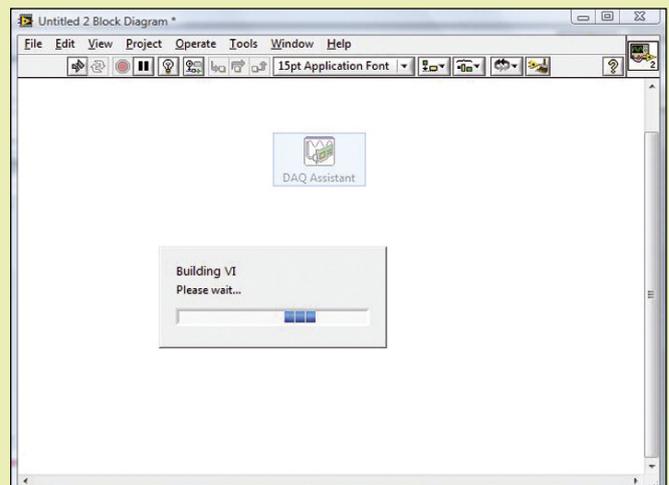


Fig. 11 - Creazione dell'Express VI.

- range);
- configurazione dell'ingresso (*Terminal Configuration*);
- frequenza di campionamento (*Rate*);
- numero di campioni visualizzati (*Samples to Read*);
- modalità di acquisizione (*Acquisition Mode*).

Esaminiamoli in dettaglio, in modo da comprendere bene come configurarli correttamente. Ci proponiamo di acquisire un segnale in modalità continua, con range da -10 a +10 volt, configurazione d'ingresso single-ended (ingressi riferiti a massa) con frequenza di campionamento 1 kS/s. Passiamo quindi all'inserimento dei parametri per ottenere la configurazione desiderata. Il parametro *Signal Input Range* verrà settato con estremi -10 e +10 volt, allo scopo di ottenere il range d'ingresso desiderato.

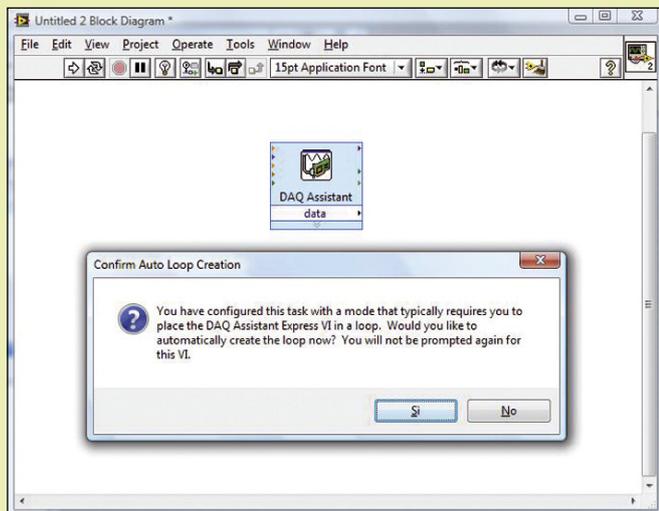


Fig. 12 - Auto Loop Creation.

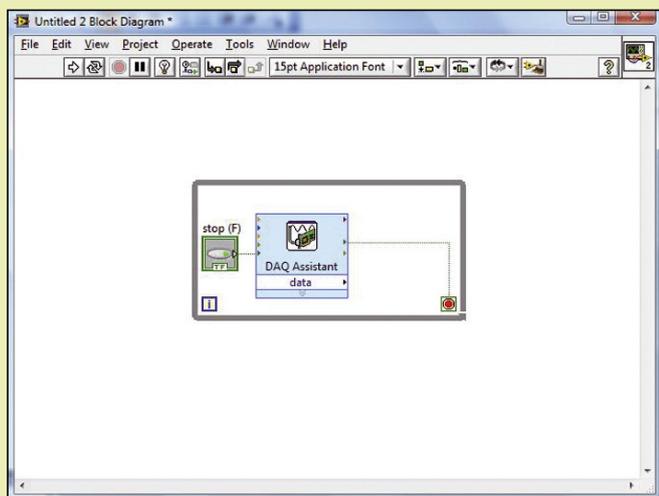


Fig. 13 - Generazione automatica del ciclo while e del pulsante di arresto.

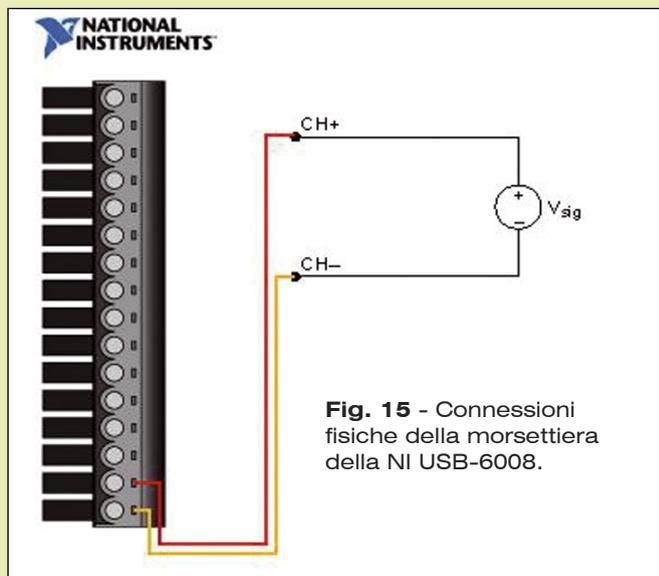


Fig. 15 - Connessioni fisiche della morsettieria della NI USB-6008.

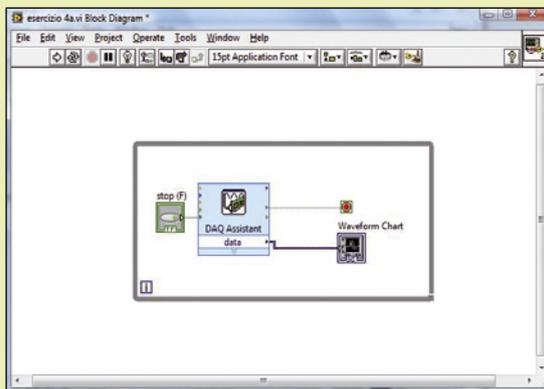


Fig. 14 - Inserimento di un waveform chart per la visualizzazione dei segnali.

La scheda NI USB-6008 dispone di una serie di amplificatori a guadagno programmabile, inseriti all'interno di ogni stadio d'ingresso analogico; ciò permette di settare opportunamente i guadagni per ottenere il range desiderato. Le possibili configurazioni d'ingresso sono due: ingressi differenziali o RSE (single-ended). Gli ingressi differenziali permettono di avere una maggiore risoluzione ed una più efficace immunità ai disturbi, ma questa configurazione impiega due canali d'ingresso, mentre la RSE permette di utilizzare tutti e 8 gli ingressi disponibili.

La scelta va fatta in funzione dell'applicazione; nel nostro caso utilizziamo la seconda opzione, quindi il parametro *Terminal Configuration* verrà settato al valore RSE.

Scegliamo adesso la modalità di acquisizione, tenendo presente che esistono diverse possibilità: si può acquisire un singolo campione su richiesta (con diversi timing) oppure ottenere un'acquisizione continua fino all'arresto del VI. Scegliamo la modalità continua, in accordo con le specifiche, quindi impostiamo il parametro *Acquisition Mode* come Continuous Samples. A questo punto non ci resta che impostare la frequenza di campionamento (Rate) ed il numero di campioni visualizzati.

La frequenza di campionamento si imposta tramite il parametro *Rate* che, nel caso specifico della 6008, può arrivare fino a 10 kS/s. Nell'esempio specifico selezioniamo una frequenza più bassa, 1 kS/s. Infine impostiamo il numero di campioni da visualizzare (parametro *Samples to Read*), inserendo 1K, che significa che verranno visualizzati 1000 campioni alla volta. Questo parametro incide sulla velocità di aggiornamento di un eventuale grafico e non può essere superiore alla frequenza di campionamento.

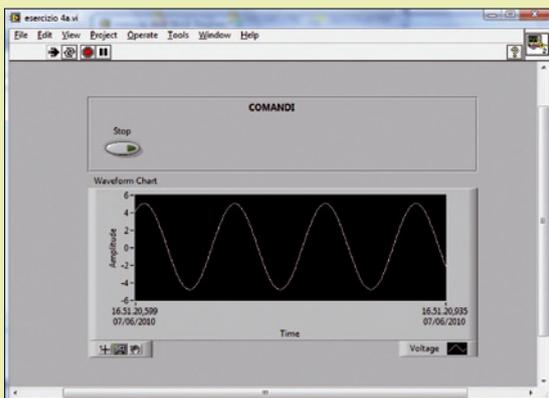


Fig. 16 - Acquisizione di un segnale sinusoidale con NI USB-6008.

A questo punto possiamo procedere: premendo il tasto OK il wizard inizierà la procedura di creazione del VI, come illustrato nella Fig. 11. Dato che abbiamo scelto la modalità di acquisizione continua, il wizard chiede automaticamente se si desidera inserire il VI all'interno di un ciclo (Auto Loop Creation, Fig. 12) e inserisce automaticamente il pulsante di arresto, per evitare di entrare in un ciclo infinito. Il ciclo che viene generato automaticamente è visibile in Fig. 13. Ora il task è stato generato ed è pronto all'esecuzione. Possiamo verificare il funzionamento del programma collegando un waveform chart all'uscita, come illustrato nella Fig. 14. Possiamo a questo punto testare quanto fatto finora, collegando il generatore di segnali alla scheda, generando un segnale sinusoidale che acquisiamo e visualizziamo su grafico. Le connessioni fisiche nel caso specifico sono riportate in Fig. 15 (ricavate direttamente dal wizard di configurazione del DAQ Assistant): basta collegare il riferimento del segnale sul terminale 1 della morsettieria ed il segnale sul terminale 2.

TEST DEL PROGRAMMA DI ACQUISIZIONE

In Fig. 16 è riportato il pannello frontale del programma esempio 6 (nel quale sono stati aggiunti i soliti elementi decorativi). Nel caso specifico abbiamo generato, tramite il generatore di funzioni, un'onda sinusoidale con frequenza pari a 10 Hz e dinamica compresa tra -5 e +5 V. Il programma acquisisce correttamente la forma d'onda e tramite le funzioni integrate all'interno dell'oggetto waveform chart è possibile effettuare zoom di vario tipo, cambiare i parametri di visualizzazione a molto altro ancora. Come si vede, con un esiguo dispendio di tempo (pochi minuti, in realtà...) è stato realizzato un completo programma di

Il multifunction DAQ NI USB-6008



Fig. 17 - Multifunction DAQ NI USB-6008.

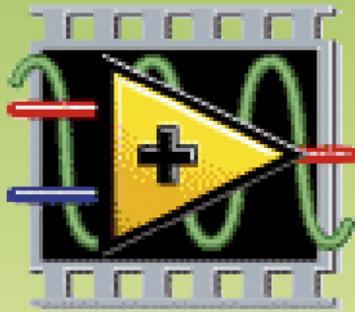
I multifunction DAQ (acronimo di Data Acquisition) della serie 60xx, sono dei DAQ USB prodotti dalla National Instruments. Tali dispositivi sono delle schede di acquisizione dati low-cost, caratterizzate da basso fattore di forma e semplicità d'uso, ideali per applicazioni di data-logging, laboratorio e sviluppo di prodotti OEM (per cui esiste una versione apposita costituita dalla sola scheda, senza case protettivo). Il dispositivo ha dimensioni contenute e dispone di interfaccia USB 2.0 e di terminali a vite sui lati per il collegamento degli I/O. Nella Fig. 17 è visibile un'immagine del DAQ.

Caratteristiche:

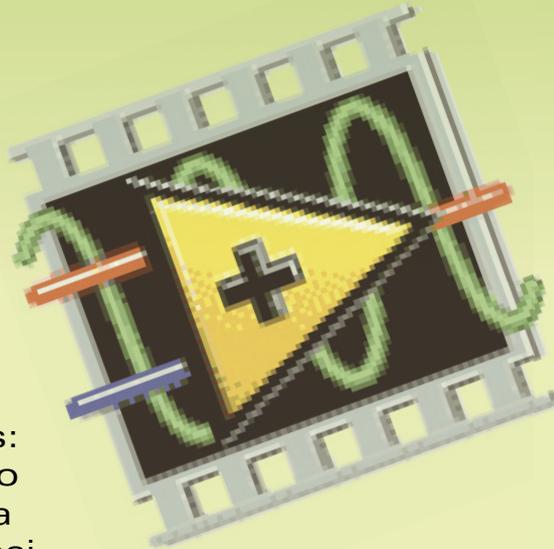
- 8 ingressi analogici a 12-bit;
- Frequenza di campionamento fino a 10 kS/s;
- 12 I/O digitali;
- 2 uscite analogiche a 12-bit, fino a 150 S/s;
- 1 Timer/Counter a 32-bit;
- alimentazione dal bus USB;
- compatibile con LabVIEW, Labwindows CVI, Measurement Studio per Visual Basic.NET.

acquisizione, che visualizza anche i risultati su grafico. Nella prossima puntata, sfruttando le conoscenze acquisite durante il corso, esanderemo le funzioni di questo programma, in modo da ottenere un completo SW di acquisizione dati e controllo di I/O. ■

Conoscere e usare LabVIEW



Completiamo il programma di esempio iniziato il mese scorso, riguardante l'acquisizione di segnali mediante la scheda USB-6008 di National Instruments: stavolta aggiungiamo un Tab Control per la gestione degli ingressi analogici e delle uscite digitali. Sesta Puntata.



di
FRANCESCO
FICILI

LabVIEW

Nella precedente puntata ci siamo lasciati dopo aver realizzato insieme un primo esempio di software di acquisizione dati in LabVIEW facente uso di una scheda NI USB-6008. In questa sesta puntata completeremo la scrittura di tale software, in modo da avere una visione globale sulla stesura di un programma di acquisizione dati completo, che sfrutti appieno le potenzialità offerte dall'hardware NI. La NI USB-6008 è uno dei prodotti entry-level; indubbiamente National Instruments propone prodotti e dispositivi molto più performanti, tuttavia i concetti applicati alla realizzazione del nostro

software applicativo possono essere facilmente estesi anche ad altre schede, sfruttando la portabilità del DAQ-Assistant. Quindi, con poche modifiche, sarà possibile adattare il programma scritto per la USB-6008 ad unità DAQ più potenti e operanti su bus differenti dall'USB. Dunque, mettiamoci al lavoro riprendendo da dove eravamo rimasti: riapriamo il programma esempio *6.vi* sul quale avevamo iniziato a lavorare nella scorsa puntata. Il front panel del VI è riportato nella **Fig. 1**. Ad esso vogliamo effettuare alcune modifiche, in modo da rendere più funzionale il programma; ma prima di procedere salviamo nuovamen-



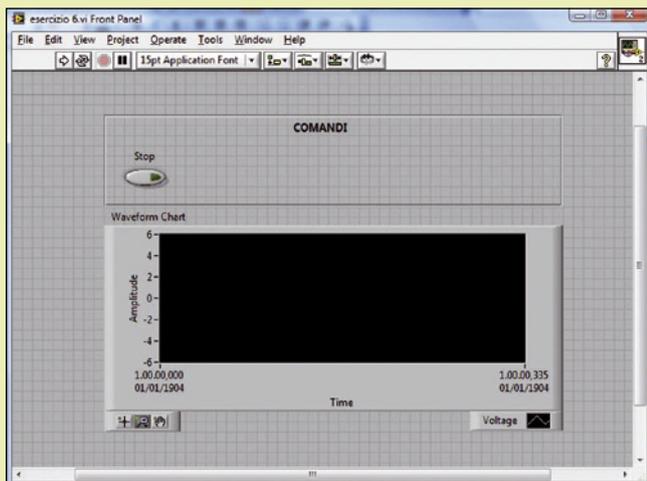


Fig. 1 - Pannello frontale del programma esempio 6.

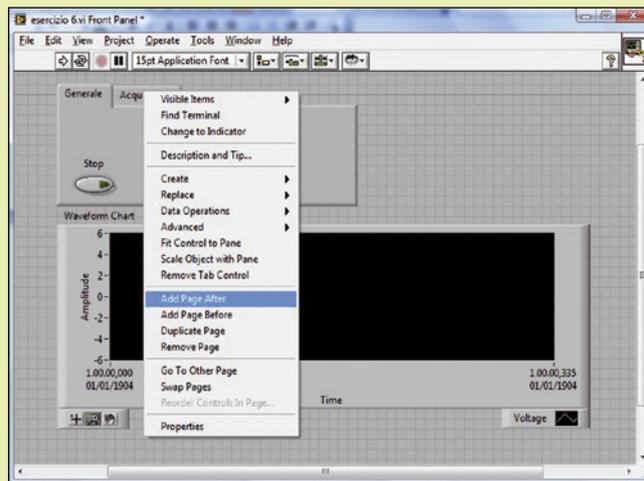


Fig. 2 - Inserimento del Tab Control.

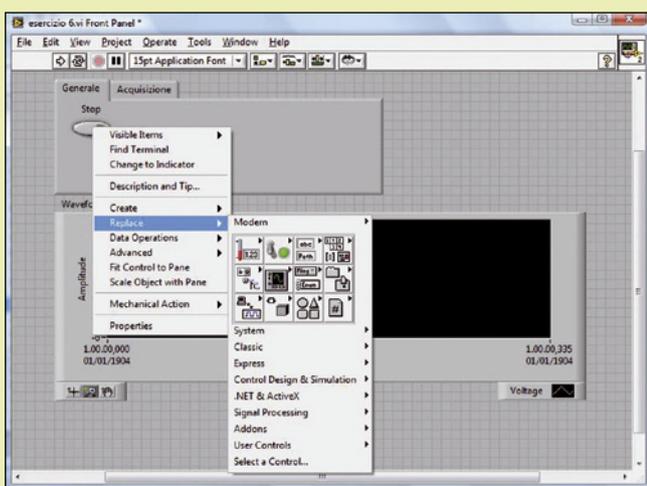


Fig. 3 - Replace del comando stop.

te il VI e cambiamone il nome in *esempio 7*. Partiamo naturalmente dal front panel, per poi passare al foglio di lavoro, come di consueto. Per prima cosa inseriamo nella zona relativa ai comandi un Tab Control (*Containers/Tab Control*) che sarà destinato a contenere i vari controlli, creando inoltre una separazione ideale tra le varie macrofunzioni. Il Tab Control permette di risparmiare spazio sul pannello frontale, consentendo di racchiudere pagine (Tab) all'interno della stessa area, in modo che ogni pagina sia accessibile premendo la relativa linguetta posta nella parte superiore del Tab. Rinominiamo le due label presenti per impostazione predefinita, rispettivamente come "Generale" e "Acquisizione", ed aggiungiamone una terza (basta fare clic con il tasto destro del mouse su una delle label e poi selezionare il comando *add page after*, come illustrato nella Fig. 2) nominandola come "I/O

Digitali". A questo punto modifichiamo il pulsante di stop e sostituiamo il pulsante dotato di LED con uno generico (Boolean/Stop Button). Per rimpiazzare un comando (o indicatore) esistente con uno di altro tipo, senza perdere tutte le proprietà associate a quel comando o indicatore (label, proprietà meccaniche, estremi, ecc.) è sufficiente cliccare con il tasto destro del mouse sopra l'oggetto da sostituire e selezionare l'opzione *Replace*, come illustrato nella Fig. 3. Ora si aprirà una nuova controlpalette che permetterà di selezionare il nuovo oggetto grafico da inserire sul pannello. Adesso inseriamo un altro pulsante generico (Boolean/OK Button) nominandolo "START".

L'avvio del programma verrà vincolato al tasto start tramite un ciclo while, in modo da creare un loop di attesa all'avvio. A questo punto vogliamo creare altre due pagine sul Tab Control, che permettano di accedere ad una serie di funzioni della scheda; per l'esattezza, intendiamo creare una pagina per il controllo del canale di ingresso analogico ed una per il controllo delle uscite digitali. Sulla pagina relativa all'ingresso analogico posizioniamo due controlli numerici, nominandoli, rispettivamente, *Frequenza di campionamento* e *Numero di campioni per ciclo*. Questi controlli numerici permetteranno di controllare tali parametri attraverso il DAQ-Assistant. Sulla pagina di controllo delle linee digitali, invece, posizioniamo otto pulsanti generici ed altrettanti LED; nelle figure 4, 5 e 6 abbiamo riportato le pagine del Tab Control, in modo da illustrare la disposizione degli elementi grafici. Vediamo adesso come è stato modificato il codice relativo all'esempio in questione. Per prima cosa abbiamo inserito, collegato al tasto

START, un ciclo di attesa software che arresta l'esecuzione del programma fino alla pressione del tasto; il codice grafico che implementa quanto detto è visibile nella Fig. 7.

CONTROLLO DEGLI I/O DIGITALI

La scheda NI USB-6008 dispone di due porte di I/O digitali, denominate rispettivamente port0 e port1, la cui piedinatura è riportata nella Fig. 8. Tutte le porte, come i contatori e gli ingressi analogici, sono accessibili tramite dei blocchi con morsettiere a vite forniti in dotazione con la scheda.

Il DAQ-Assistant, che abbiamo già utilizzato per la generazione del task di acquisizione, può essere utilizzato anche per creare il task deputato al controllo degli I/O digitali. Esattamente come era stato fatto per il task di acquisizione, inseriamo all'interno del foglio di lavoro un *express task* di tipo DAQ Assistant (*Express/ Input/DAQAssist*). Comparirà quindi il wizard che già conosciamo, visibile nella Fig. 9. Stavolta, anziché scegliere la prima opzione (Acquire Signal), optiamo per Generate Signal. Come si vede è possibile generare diversi tipi di segnali in uscita, sia analogici (sfruttando i DAC) che digitali (sfruttando le porte di uscita).

Scegliamo l'opzione Digital Output e all'interno di essa Line Output. L'opzione line output permette di controllare una o più linee di uscita di una data porta.

Una volta selezionata l'opzione, comparirà la schermata visibile nella Fig. 10. Come detto in precedenza, la NI USB-6008 dispone di due porte digitali, utilizzabili come porte GPIO (General Purpose Input Output), la port0 (8-bit wide) e la port1 (4-bit wide). Per il nostro

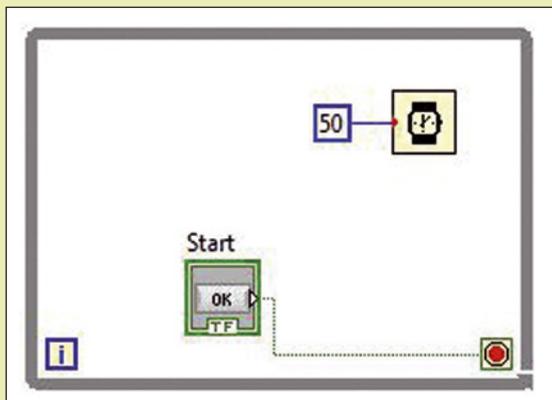


Fig. 7
Ciclo di attesa sul tasto Start.

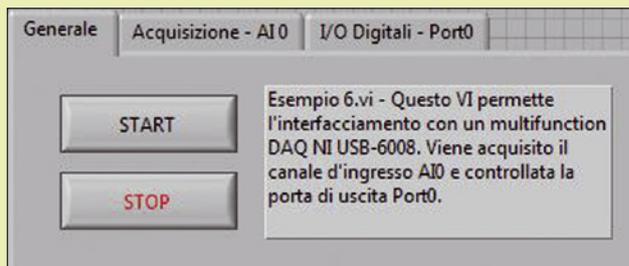


Fig. 4 - Tab Control Generale.

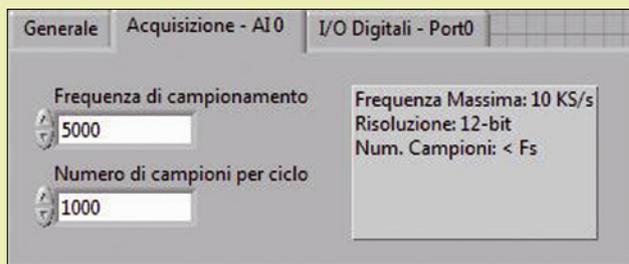


Fig. 5 - Tab Control Acquisizione.



Fig. 6 - Tab Control I/O Digitali.

Fig. 8 - Pin-out della scheda NI USB-6008.

Terminal	Signal
17	P0.0
18	P0.1
19	P0.2
20	P0.3
21	P0.4
22	P0.5
23	P0.6
24	P0.7
25	P1.0
26	P1.1
27	P1.2
28	P1.3
29	PFIO
30	+2.5V
31	+5V
32	GND

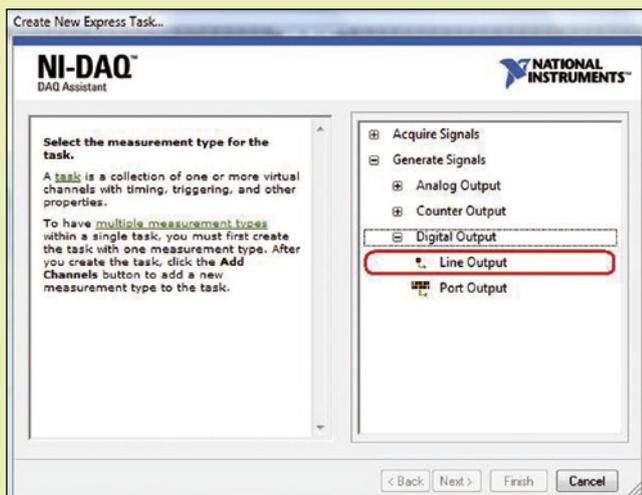


Fig. 9 - Nel wizard di configurazione del task scegliamo l'opzione Line Output, che si trova all'interno del percorso Generate Signal / Digital Output.

esempio utilizzeremo per intero la port0. Per selezionare tutti gli otto canali di quest'ultima, possiamo utilizzare il mouse ed il tasto shift per effettuare una selezione multipla, come visibile nella Fig. 11. Premendo il tasto finish apparirà la schermata illustrata nella Fig. 12. Dato che la generazione dei segnali di output avverrà on-demand, non dobbiamo cambiare nulla relativamente alle opzioni di time settings (*Generation mode: 1 sample (on demand)*). Possiamo verificare il funzionamento del task utilizzando l'apposito strumento di debug posto nella parte alta della schermata, come visibile nella Fig. 13.

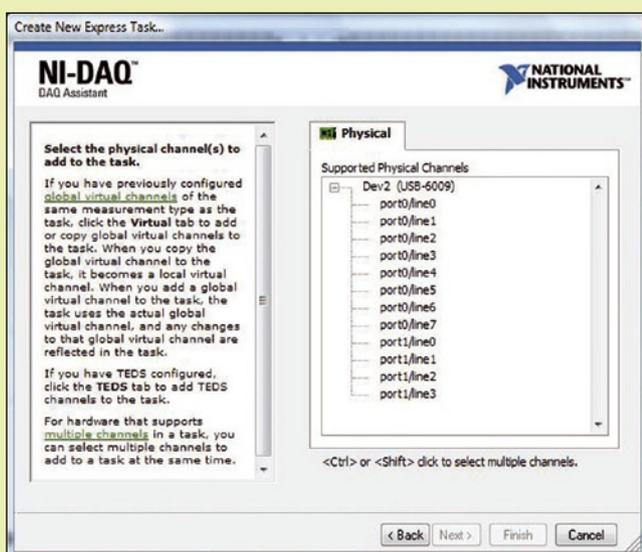


Fig. 10 - Elenco delle linee disponibili.

Sono poi presenti alcune opzioni, come la possibilità di invertire le linee ed altro ancora. Per visualizzare l'output delle linee digitali potete servirvi di una bread board e di 8 LED, collegati alla scheda sfruttando i connettori a morsettiera forniti in dotazione. Una volta verificato che il task funziona è sufficiente fare clic sul tasto OK: il task verrà generato ed il VI sarà posizionato sul foglio di lavoro. Ora il task per la gestione degli I/O digitali è pronto; ciò che bisogna fare è inserirlo all'interno del ciclo principale, in modo che lavori insieme al task di acquisizione. Abbiamo due possibilità: i due task possono lavorare in parallelo (si tratta di un multitasking puro) oppure in sequenza. Le due situazioni sono schematizzate nella Fig. 14.

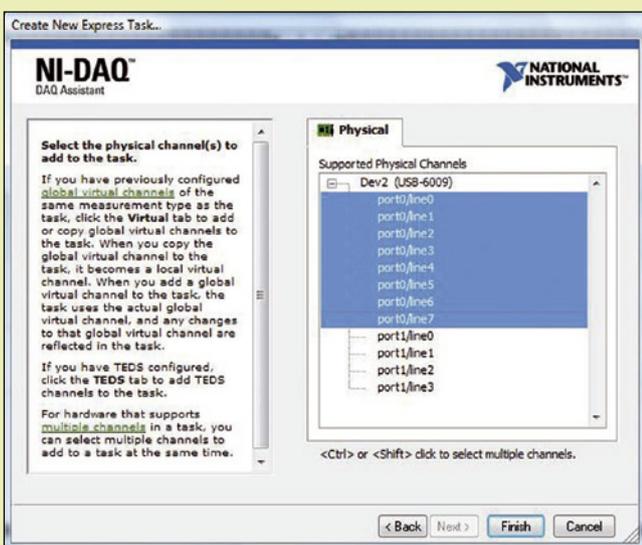


Fig. 11 - Selezione delle 8 linee di port0.

Nel programma di esempio si è scelta un'esecuzione di tipo sequenziale, ma solo per una questione di leggibilità del programma. Naturalmente il fatto di disporre in maniera sequenziale i due task penalizza leggermente la velocità di attuazione delle uscite digitali (in quanto bisogna attendere la fine del task di acquisizione). Per programmi con tempistiche non troppo stringenti, questa soluzione può anche andar bene, mentre in situazioni *time-critical* si dovrà necessariamente ricorrere ad un'esecuzione di tipo parallelo.

Vediamo a questo punto come è stato modificato il codice grafico del software per implementare quanto illustrato logicamente prima: l'ultima volta avevamo scritto il codice grafico appena sufficiente a far girare il task di acquisizione, che riportiamo, per comodità, in Fig. 15. Aggiungiamo a tale codice due strutture di tipo stacked sequence, la prima (che chiameremo *sequence 2*) immediatamente

Fig. 12 - Settaggio dei parametri del task.

all'interno del while e l'altra (che chiameremo *sequence 1*) a parte. Entrambe le strutture devono avere due frame. Ora inseriamo nel primo frame della struttura esterna il codice visibile nella Fig. 7, che vincola l'esecuzione del programma alla pressione del tasto START sul pannello frontale. Nel secondo frame trasportiamo il vecchio codice, con la nuova struttura *sequence* all'interno. La struttura del programma sarà quindi la seguente: la struttura *sequence 1* contiene nel primo frame il ciclo while bloccante, vincolato al tasto start, e nel secondo frame ha il ciclo while contenente il resto del programma.

Passiamo ora al secondo frame della struttura *sequence 1*: il codice all'interno del ciclo while deve essere modificato come illustrato nella Fig. 16, dove si vede che il task di acquisizione è stato espanso (per farlo basta posizionarsi con il puntatore del mouse sul bordo inferiore del VI, cliccare e tenere premuto il tasto sinistro e trascinare il puntatore verso il basso) per mostrare tutti gli input e gli output del blocco. Collegiamo i due controlli numerici inseriti precedentemente nel front panel (Numero di campioni per ciclo e Frequenza di campionamento) rispettivamente agli ingressi *number of sample* e *rate* del VI. Ciò ci consentirà di variare a piacimento, ad ogni avvio del programma, questi parametri. A riguardo, si tengano presenti i seguenti aspetti: il parametro *rate* (controllato dal controllo numerico frequenza di campionamento) per la 6008 non può essere superiore alla massima frequenza di campionamento (10 kS/s), ed il numero di campioni per ciclo non può essere mai superiore al valore

Fig. 14 - Esecuzione parallela o sequenziale dei due task.

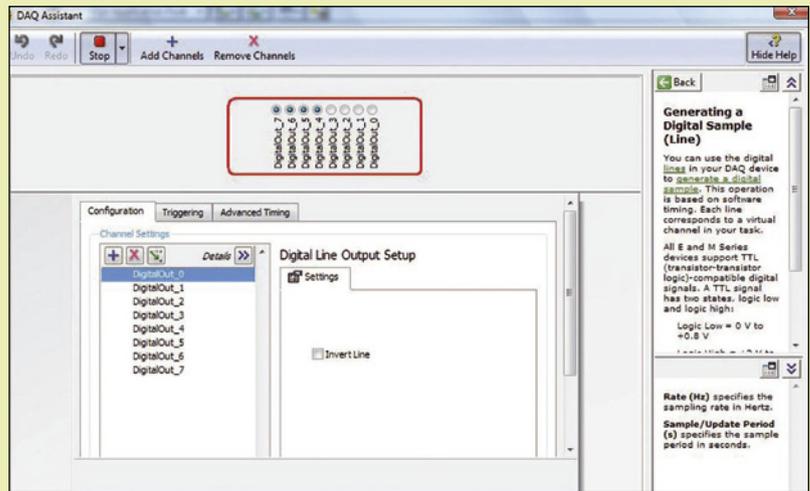
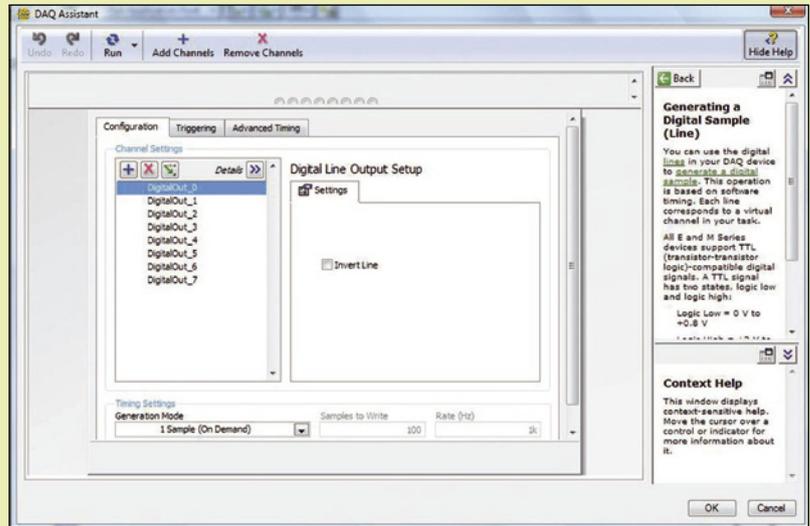
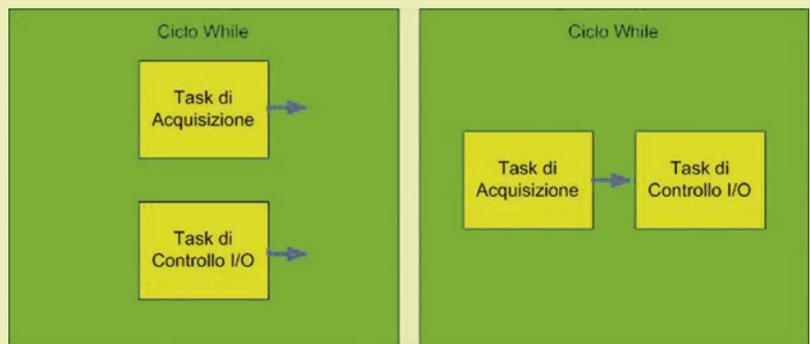


Fig. 13 - Uso dello strumento di debug del task.

impostato del parametro *rate*. Il numero di campioni per ciclo incide esclusivamente sulla velocità con cui i dati vengono presentati sul grafico. Ad esempio, la seguente impostazione:

$$\text{Numero di campioni per ciclo} = \text{Frequenza di campionamento}$$



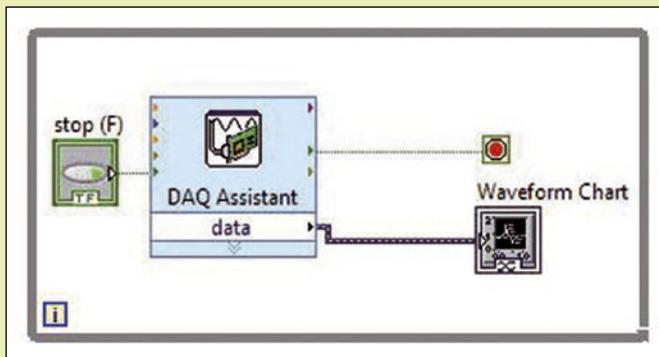


Fig. 15 - Codice grafico relativo al programma esempio 6, scritto nella puntata precedente.

Genera un aggiornamento dei dati per ciclo, quindi un aggiornamento al secondo. Valori inferiori di questo parametro, relativamente al parametro *rate*, determinano aggiornamenti più veloci, ma caricano di lavoro maggiormente il processore grafico. Con i valori massimi di frequenza di campionamento garantiti dalla 6008 è difficile notare qualche differenza anche con macchine poco performanti, ma con valori più elevati e con programmi più complessi, bisogna iniziare a tenere in considerazione il numero di campioni per ciclo.

Come si può vedere sempre nella Fig. 16, la linea di tipo boolean che agisce sull'ingresso di stop del ciclo while è stata trasportata fuori dalla sequenza tramite un tunnel; quest'operazione è stata resa necessaria dal fatto che non è possibile portare l'ingresso di stop del while all'interno di un'altra struttura. Si noti che è stato usato un tunnel e non un sequence local, dato che il sequence local serve a spostare segnali da un frame all'altro di una struttura stacked sequence, e non al di fuori di essa.

Passiamo ora al secondo frame della struttura sequence 2, che contiene il codice grafico relativo alla gestione delle linee di I/O digitale. Il codice utilizzato è stato riportato nella Fig. 17. In questo frame sono stati spostati, oltre al task di I/O, gli otto pulsanti e gli otto LED generati sul front panel, per la gestione di questo task. I LED non sono strettamente necessari in questo caso, dato che i pulsanti cambiano colore in base al loro stato e, volendo, possono anche esplicitarlo in maniera più chiara tramite un testo associato al loro stato booleano. Tuttavia l'inserimento dei LED permette di evidenziare una caratteristica dei controlli che non è stata ancora presentata all'interno del corso, e che vedremo successivamente.

Iniziamo a vedere come vengono utilizzati gli otto pulsanti per la gestione delle linee. Il task di I/O accetta una singola linea di ingresso, di tipo array di boolean. Se si provasse, quindi, a collegare un semplice ingresso di tipo booleano (anche ad un task che gestisce una singola linea di I/O) si genererebbe un errore. Ciò che bisogna fare è costruire un array a partire dalle singole variabili booleane associate ai tasti. Per farlo ci occorre un blocco specifico, in particolare il blocco build array (*array/build array*). Una volta posizionato il blocco sul foglio di lavoro occorre dimensionarlo, dato che, per impostazione predefinita, si presenta con un singolo ingresso. Per dimensionarlo in modo

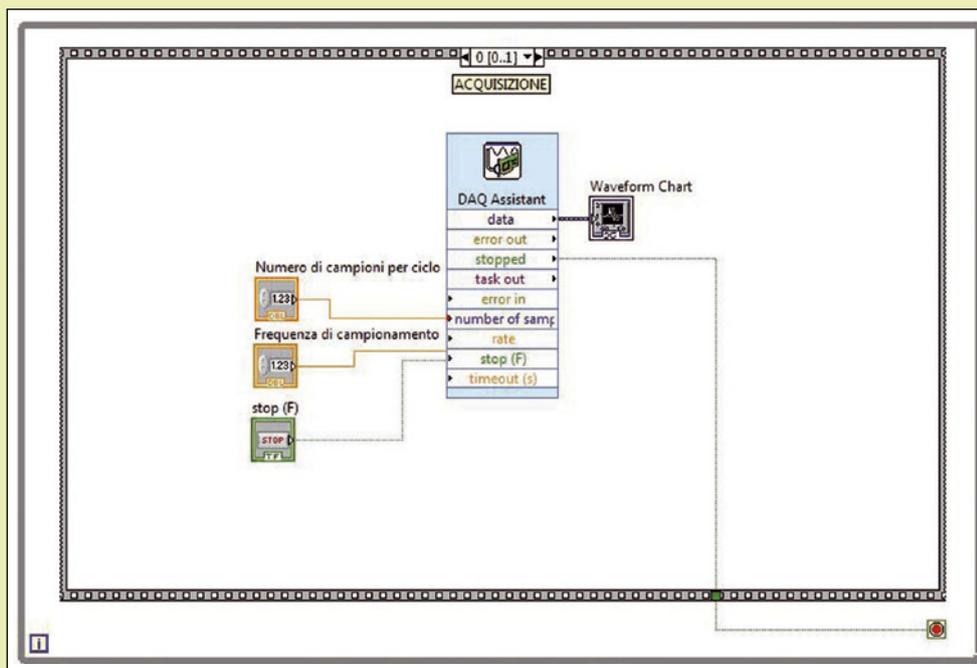


Fig. 16
Codice grafico relativo al task di acquisizione.

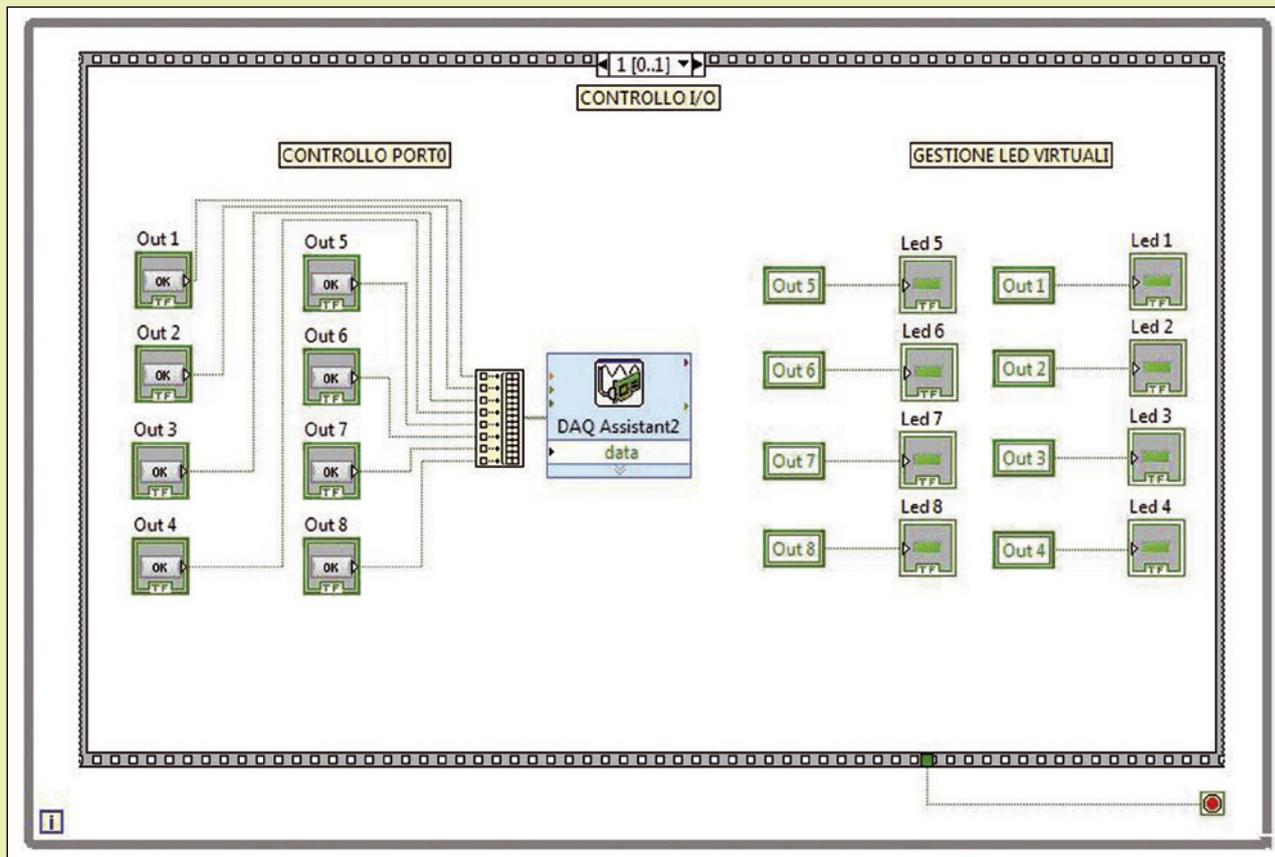


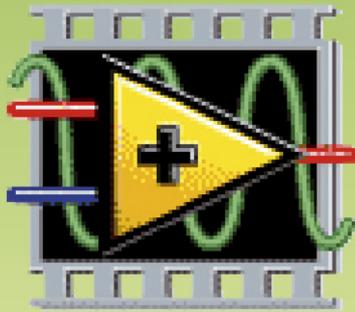
Fig. 17 - Codice grafico relativo al task di Output Digitale.

che ne possa accogliere otto, basta posizionarsi sul bordo inferiore del blocco con la freccia del mouse, premere e mantenere premuto il tasto sinistro e trascinare verso il basso, fino a quando gli ingressi del blocco non diventano otto. Ora colleghiamo una ad una le uscite booleane di tutti gli otto controlli di tipo pulsante che sono stati inseriti. Perché l'associazione tasto/linea di I/O sia corretta, il collegamento va fatto in base al nome del controllo, quindi il controllo Out1 andrà sul primo ingresso del blocco build array (quello più in alto), Out2 sul secondo e via discorrendo. Terminata quest'operazione bisogna collegare l'uscita del blocco build array all'ingresso data del task. Una volta fatto, le linee gestite dal task passano sotto il diretto controllo dei pulsanti virtuali del front panel. In ultimo vediamo cosa è stato fatto per la gestione dei LED. Naturalmente, volendo associare direttamente lo stato di un pulsante ad un LED virtuale basta collegare la linea booleana del tasto direttamente al LED. Questo va bene con programmi piccoli, ma con grossi schemi a blocchi, con centinaia di controlli ed indicatori può diventare un problema e la quantità di linee può rendere poco leggibile il programma. Inoltre, nel caso in cui si desideri collegare

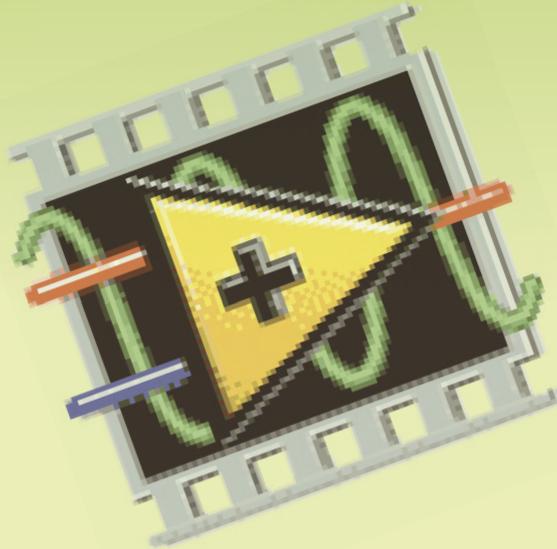
un controllo ad un indicatore in programmi composti da molte strutture annidate, bisognerebbe usare una grossa quantità di sequence local e tunnel. Anche questo tipo di operazione può rendere poco leggibile un programma. Per ovviare a tale problema, LabVIEW permette di generare delle variabili locali associate ad un qualsivoglia controllo o indicatore; allo scopo, basta cliccare con il tasto destro del mouse sul controllo/indicatore e selezionare l'opzione *Create/Local Variable*. Il blocchetto generato è un riferimento alla variabile di partenza che può essere trasportato in un punto qualsiasi del foglio di lavoro del VI. In questo modo, nel nostro programma sono state generate otto variabili locali, riferite agli otto tasti e collegate agli otto LED. L'effetto è lo stesso che si ottiene collegando le linee logiche dei pulsanti ai LED, a beneficio, però, di una migliore leggibilità del programma.

Bene, a questo punto la versione 2 del nostro programma di esempio è completa. Nella prossima puntata vedremo come sfruttare le conoscenze acquisite finora per salvare su file i risultati dell'acquisizione e processare i dati, tramite i potenti blocchi di post-processing forniti da LabVIEW. ■

Conoscere e usare LabVIEW



Come dotare il programma che abbiamo creato per acquisire i dati di un'interfaccia per salvare in un file di testo i risultati dell'acquisizione mediante l'unità DAQ. Settima Puntata.



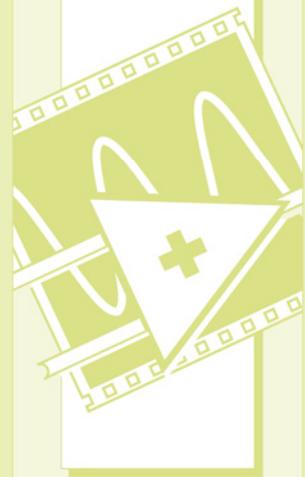
di
FRANCESCO
FICILI

Nella precedente puntata abbiamo continuato la scrittura del software di acquisizione dati per la scheda NI USB-6008, la quale è un multifunction-DAQ prodotto dalla NI che offre interessanti prestazioni ad un prezzo ragionevole; nella stessa occasione abbiamo spiegato come ampliare il software, aggiungendo il supporto per la gestione degli I/O digitali. Ora vogliamo dare una veste definitiva alla nostra interfaccia, sfruttando le conoscenze acquisite durante il resto del corso. Aggiungeremo quindi al nostro programma la possibilità di salvare i dati acquisiti tramite la NI USB-6008 in un file di testo, sfruttando le conoscenze che

abbiamo acquisito sull'argomento nella quarta puntata. Per prima cosa aggiungiamo i controlli e gli indicatori necessari a realizzare quanto ci proponiamo. Allo scopo creiamo, sul pannello frontale, una sezione dedicata al salvataggio, come illustrato nella Fig. 1.

In questa sezione inseriamo un pulsante con LED (*Boolean/ Push Button*), denominato *Salva su File* (la cui funzione è appunto quella di fornire all'operatore la possibilità di scegliere se salvare o meno i dati su file di testo) ed un indicatore di tipo path (*String&Path/File Path Indicator*), denominato *Path del File*. L'indicatore di tipo path ci servirà a visualizzare

LabVIEW



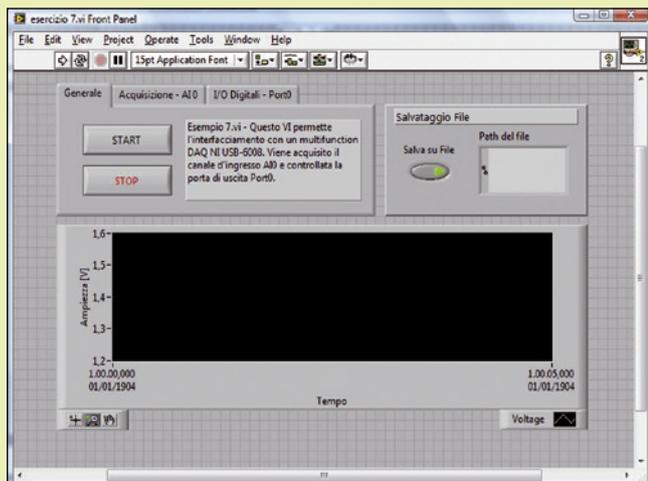


Fig. 1 - Pannello frontale definitivo del programma esempio 7.

Tabella 1 a

0	open (default) — Apre un file esistente. Se il file non viene trovato, si presenta l'errore 7.
1	replace — Sostituisce un file esistente aprendolo ed impostandone la fine a 0.
2	create — Crea un nuovo file; se il file da creare esiste già, viene segnalato l'errore 10.
3	open or create — Apre un file esistente o crea un nuovo file se non esiste.
4	replace or create — Crea un nuovo file o sostituisce quello indicato, se esiste. Questo VI sostituisce i file aprendolo e impostandone la fine a 0.
5	replace or create with confirmation — Crea un nuovo file o sostituisce un file esistente se gli date il consenso. Questo VI sostituisce un file aprendolo e impostandone la fine a 0.

Tabella 1 b

0	Read/write.
1	Read-only.
2	Write-only.

Tabella 1 a e 1 b - Valori assunti e significato degli ingressi operation (1 a) ed access (1 b) del blocco Open/Create/Replace File.

Si consideri che il valore assunto di default è sempre 0.

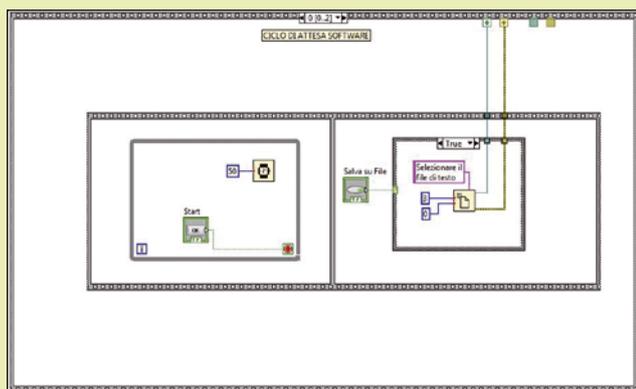


Fig. 2 - Block diagram del primo frame della struttura sequence principale.

il percorso all'interno del quale viene salvato il file di testo contenente i dati.

Ora possiamo modificare il codice grafico in modo da aggiungere le nuove funzionalità al programma. Iniziamo dal frame di apertura (frame 0) della struttura sequence, il quale inizialmente conteneva il ciclo di attesa software vincolato al tasto START, che aveva la funzione di impedire l'immediata esecuzione del programma all'apertura del VI. Questa porzione di codice va parzialmente modificata, aggiungendole una stacked-sequence con due frame: nel primo frame inseriamo il codice grafico del ciclo di attesa, mentre nel secondo realizziamo il codice che apre ed inizializza il file di testo che andremo a scrivere.

Dato che forniamo all'operatore la possibilità di scegliere se salvare o meno il file di testo, inseriamo una struttura di selezione (*case structure*) collegata al pulsante *Salva su File*. All'interno del caso *True* inseriamo il codice relativo all'apertura o creazione del file. Il blocco che ci interessa è *Open/Create/Replace File*. Questo sub-VI, a seconda dei parametri in ingresso, apre o crea ex-novo un file ed accoda o sovrascrive i dati. I parametri di input sono diversi: oltre alla linea di errore in ingresso, abbiamo un input di tipo stringa (*prompt*), un ingresso di tipo file path (*file path*), e due ingressi numerici, denominati rispettivamente *operation* e *access*. Come fatto anche in precedenza lasciamo libero l'ingresso file path, in modo che venga richiamata automaticamente la finestra di dialogo di windows per la selezione del percorso di salvataggio del file (vedere la quarta puntata). L'ingresso prompt (che costituisce il titolo della finestra di dialogo) viene collegato ad una costante di tipo stringa, come illustrato nella Fig. 2.

Particolare attenzione va portata agli ingressi *access* e *operation*: questi due stabiliscono le modalità di apertura del file e le operazioni che su di esso possono essere eseguite. Per comodità, nelle tabelle 1a ed 1b sono riportati i valori che possono essere assunti da tali ingressi ed il loro significato, così come illustrato nell'help di LabVIEW.

Impostiamo quindi le opzioni di accesso come read/write (valore 0) e le operazioni di apertura come open or create (valore 3), come visibile sempre nella Fig. 2. A questo punto creiamo due *sequence local* per portare al frame successivo il reference number relativo al path del file

e la classica linea di errore. Ricordiamo ancora una volta la differenza tra sequence local e tunnel: la sequence local serve a portare dati (wire) da un frame all'altro di una struttura sequenze, mentre i tunnel servono, genericamente, a portare dati al di fuori delle strutture (sequenze, case, while, ecc.). Detto ciò, passiamo ora al secondo frame della struttura principale, dove viene effettivamente inserito il codice che effettua il salvataggio dei dati su file. Il frame in oggetto è rappresentato nella Fig. 3: come si vede, prendiamo in considerazione solo la porzione di codice relativa all'acquisizione del canale d'ingresso analogico, dato che la gestione dei digitali non influisce su questo task. Ciò che bisogna fare è prelevare i dati in uscita dal DAQ Assistant, convertirli in maniera opportuna ed utilizzare un blocco apposito per la memorizzazione.

Il blocco da utilizzare è il blocco di scrittura su file di testo (*File I/O/Write to text file*); esso prende in ingresso, oltre al reference number ed alla linea di errore (che preleviamo dai relativi sequence local creati allo scopo in precedenza), i dati da scrivere su file, sotto forma di stringa di testo. Bisogna quindi convertire i dati di tipo DDT (*Dynamic Data Type*) in uscita dal DAQ Assistant, in un array di caratteri (*string type*).

Nella Fig. 4 è evidenziata la porzione di codice grafico che realizza la conversione. Per prima cosa si utilizza un blocco specifico per convertire il flusso di dati in uscita dal DAQ Assistant in un array di double: si tratta del blocco *from DDT (Express/Signal Manipulation/From DDT)*. Una volta ottenuto l'array, bisogna convertirlo in stringa. Per convertire un dato di tipo numerico in stringa (in formato ASCII) esistono dei blocchi appositi. Nel caso specifico, utilizziamo il blocco *Number to Fractional String (String/String to Number Conversion/Number to Fractional String)*.

A questo punto il flusso di dati numerici (array di double) viene convertito in un array di caratteri ASCII che rappresentano in maniera testuale i dati numerici acquisiti. L'uscita del blocco di conversione può essere collegata direttamente all'ingresso del blocco *Write to Text File*, come visibile sempre nella Fig. 4.

Tutta questa porzione di codice grafico viene racchiusa all'interno di una struttura case, il cui ingresso booleano è collegato ad una

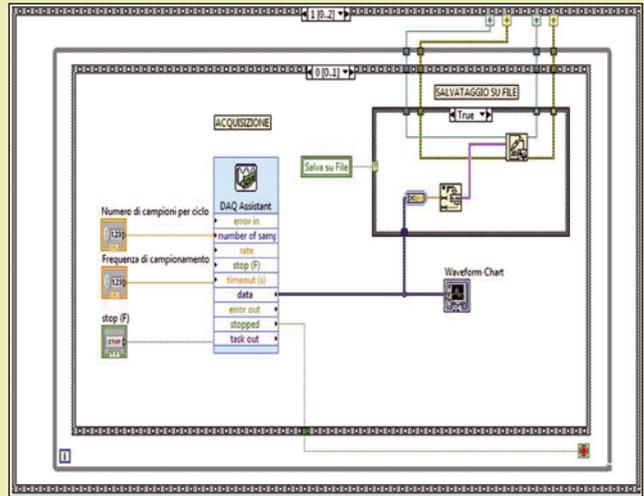


Fig. 3 - Block diagram del secondo frame della struttura sequenze principale.



Fig. 4 - Codice grafico che implementa la scrittura su file di testo.

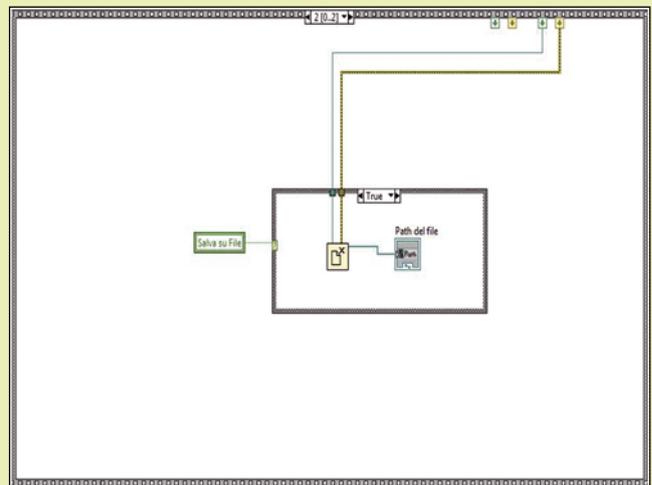


Fig. 5 - Codice grafico relativo alla chiusura del file di testo.

variabile locale ricavata dal controllo *Salva su File*. Questa operazione serve ad impedire che il programma provi a scrivere il file nel caso in cui questo non sia stato regolarmente aperto all'inizio (opzione *Salva su File* disabilitata). Infine notiamo come vengano generati altri due

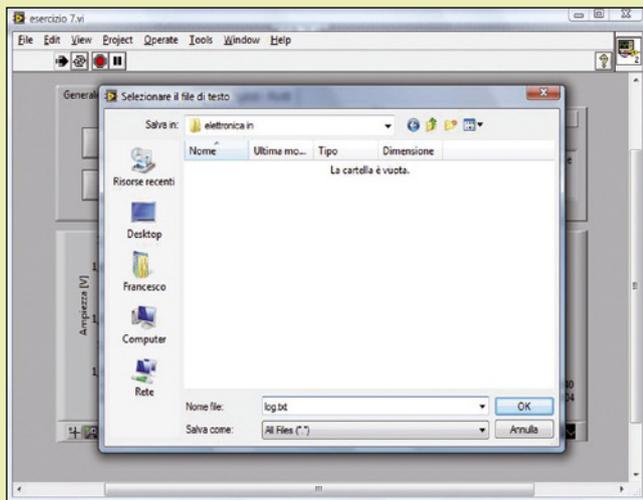


Fig. 6 - Finestra di dialogo per la selezione del percorso del file.

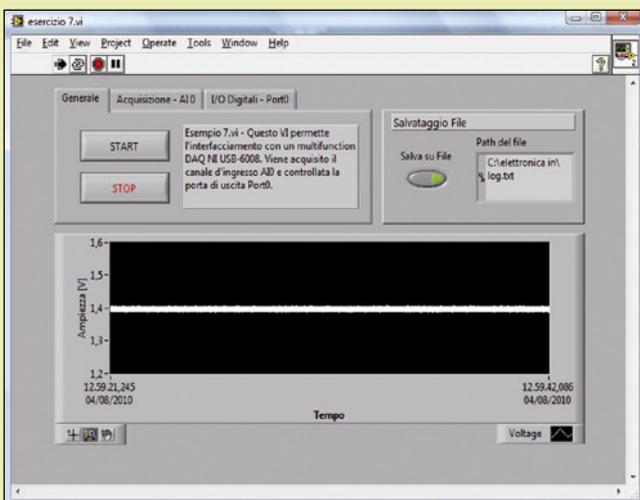


Fig. 7 - Acquisizione del canale AI0.

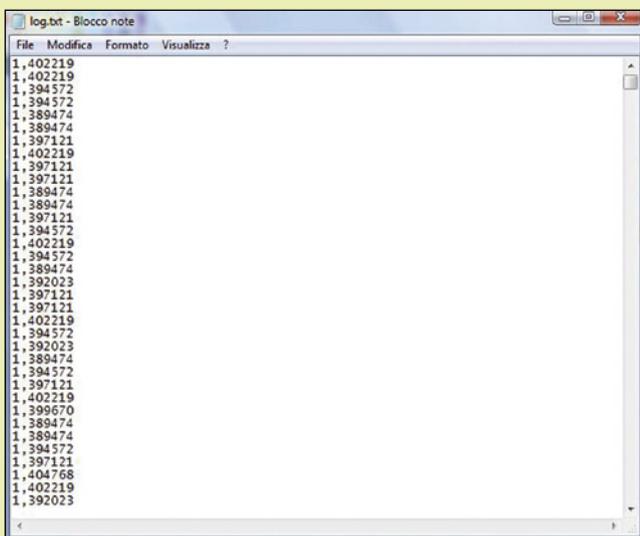


Fig. 8 - File di testo con i dati acquisiti.

sequence local per trasportare al frame successivo le linee di reference number del path e la linea per la gestione degli errori.

Passiamo adesso all'ultimo frame, dove viene gestita la chiusura del file. Viene naturalmente utilizzato il blocco File Close (*File\O/File Close*) al quale vengono collegate le linee ricavate dai sequence local. Anche la chiusura del file viene vincolata su una struttura case controllata dalla solita variabile locale relativa al pulsante *Salva su File*. Il codice grafico relativo a questa porzione di software è visibile nella **Fig. 5**.

Il nostro programma a questo punto è concluso, e possiamo provare ad eseguirlo in debug per verificarne il funzionamento. All'apertura, dopo la pressione del tasto START viene richiesto il percorso del file sul quale verranno memorizzati i dati, come visibile nella **Fig. 6**.

Il software che è stato scritto in queste ultime puntate può essere utilizzato, in congiunzione con la NI USB-6008, per realizzare dispositivi di data-logging PC based, con frequenze di campionamento fino a 10 kS/s, e consente inoltre il controllo delle linee di output e la visualizzazione dei dati in real-time. Il tutto viene realizzato con una manciata di blocchi e qualche struttura di controllo.

Nella **Fig. 7** è rappresentato un esempio di acquisizione. Nella **Fig. 8** è visibile il file di testo salvato durante l'acquisizione.

I PROPERTY NODE

Tutti gli elementi di tipo *control* o *indicator* di LabVIEW sono caratterizzati da una serie di proprietà. Questo concetto è familiare anche ad altri ambienti di programmazione; ad esempio, gli elementi della tool palette di Microsoft Visual Studio dispongono di diverse proprietà caratteristiche, che possono essere modificate dall'utente sia in fase di programmazione che a run-time. In LabVIEW il controllo di tali proprietà avviene tramite oggetti grafici specifici, denominati *property node*.

Un property node può essere associato ad un qualsiasi controllo o indicatore di LabVIEW e permette di modificare le proprietà di tale controllo o indicatore a run-time, ossia direttamente durante l'esecuzione del programma. In maniera più specifica, tramite un property node è possibile, ad esempio:

- modificare l'aspetto, i valori di scala o gli incrementi dei controlli numerici come slider,

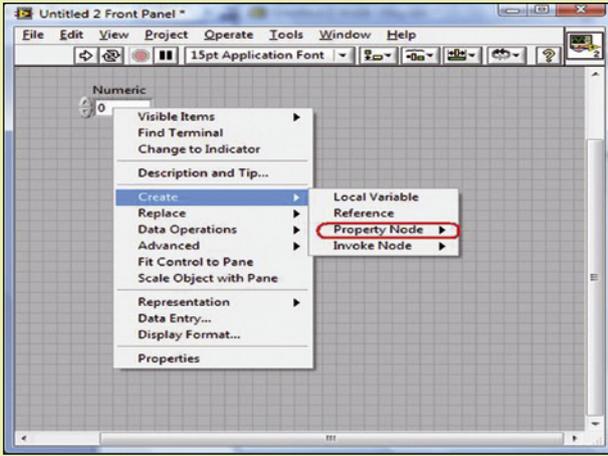


Fig. 9 - Creazione di un property node associato al controllo numerico.

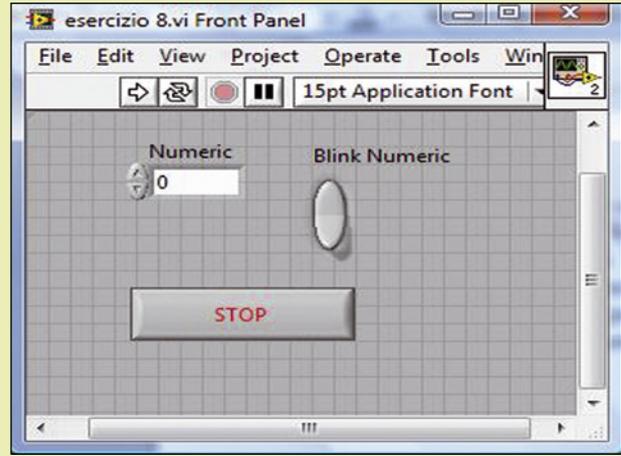


Fig. 10 - Pannello frontale del programma di esempio.

- knob, dial, tank ed altri;
- modificare le impostazioni di visualizzazione degli oggetti di tipo graph o chart (colori, scale, label, cursori, e altro);
- modificare la lista di opzioni di un controllo di tipo menu;
- abilitare/disabilitare qualsiasi tipo di controllo.

La procedura per l’inserimento (o, per meglio dire l’associazione) di un property node è illustrata nella Fig. 9. Nell’esempio è stato creato un generico controllo numerico denominato, appunto, *Numeric*. Per creare un property node associato a tale controllo bisogna cliccare con il tasto destro del mouse su tale controllo e selezionare l’opzione *Create/Property Node*, come illustrato nella Fig. 9.

Selezionando tale opzione comparirà un menu a tendina con elencate tutte le proprietà disponibili per l’oggetto selezionato.

Selezionando una delle proprietà, l’ambiente aprirà automaticamente il foglio di lavoro e creerà il property node selezionato, permettendo all’utente di posizionarlo all’interno del block diagram. Per illustrare meglio questo meccanismo possiamo fare un esempio puramente didattico, che però ci permetterà di comprendere in maniera più approfondita questo aspetto, il quale risulta fondamentale nella scrittura di software con maggiore livello di complessità.

Al controllo numerico creato precedentemente associamo un property node, linkato alla proprietà *blinking* del controllo stesso (*Create/Property Node/Blinking*). La *blinking* è una proprietà generica di controlli ed indicatori, e se attivata

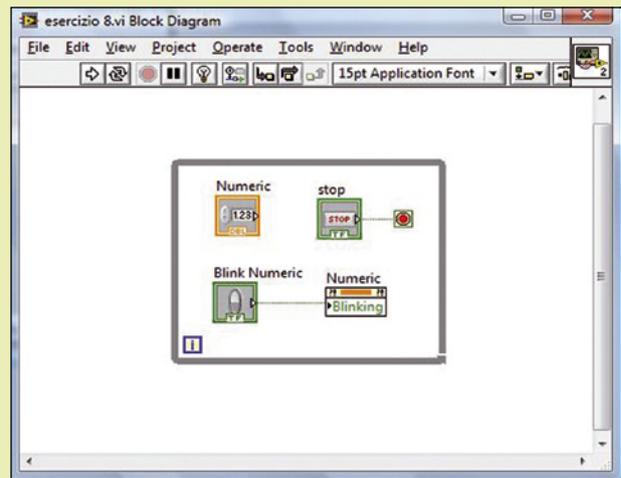


Fig. 11 - Schema a blocchi del programma di esempio.

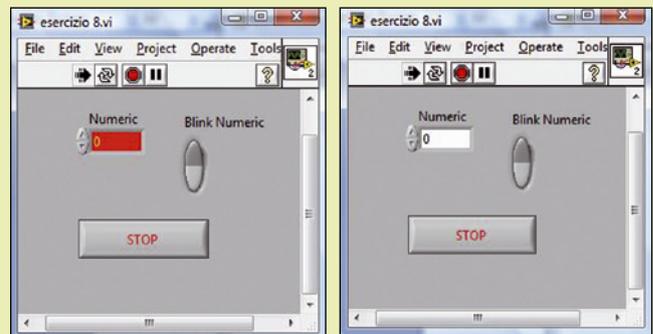


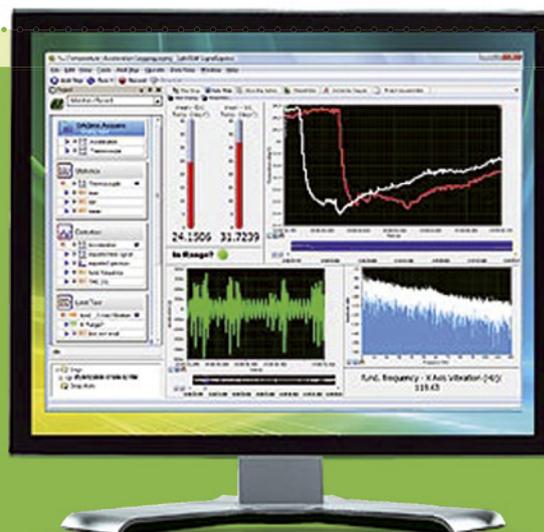
Fig. 12 - Esecuzione del programma dimostrativo.

(condizione d’ingresso pari a *true*) genera un lampeggio periodico del controllo o indicatore al quale è associata. A questo punto inseriamo nel pannello frontale un Vertical Rocker (*Boolean/Vert Rocker*), denominandolo “Blink Numeric” ed un pulsante di stop (*Boolean/*

Le librerie matematiche di LabVIEW

LabVIEW dispone di una ricca dotazione di librerie matematiche, che possono essere utilizzate all'interno della nostre applicazioni. La dotazione di librerie matematiche di LabVIEW comprende:

- **Elementary & Special Function;** blocchi con funzioni trigonometriche, esponenziali, ellittiche, più altre funzioni particolari;
- **Linear Algebra;** serie di blocchi per la gestione di operazioni con matrici;
- **Fitting;** blocchi con vari tipi di fit (fit lineare, esponenziale, logaritmico, etc.);
- **Interpolation & extrapolation;** blocchi di interpolazione polinomiale ed altro tipo;
- **Integration & differentiation;** questa sottofamiglia contiene alcuni blocchi per la realizzazione dell'integrazione numerica e della quadratura;
- **Probability & Statistics;** una grossa raccolta di blocchi di analisi statistica;
- **Optimization;** sono alcuni blocchi finalizzati all'ottimizzazione;
- **Differential equation;** blocchi per la soluzione delle ODE (Ordinary Differential Equation);



- **Geometry;** blocchi geometrici per la realizzazione di rotazioni e traslazioni nello spazio cartesiano;
- **Polynomial;** è una notevole raccolta di blocchi per il calcolo polinomiale.

Tutte queste librerie possono essere utilizzate per realizzare elaborazioni complesse dei dati. Chiaramente addentrarsi nella descrizione dettagliata di tutti questi blocchi esula dagli scopi di un corso di programmazione di base, essendo un argomento tipico dei corsi avanzati, ma è bene sapere che essi esistono e all'occorrenza possono essere utilizzati.

Stop Button), che chiameremo semplicemente "Stop". Nella Fig. 10 è rappresentato l'aspetto che dovrebbe assumere il pannello frontale dell'esempio proposto.

Ora passiamo sul foglio di lavoro. Per prima cosa cambiamo la proprietà di lettura/scrittura del property node, cliccando su di esso con il tasto destro e selezionando la prima opzione dall'alto (*Change All To Write*).

Questo consente di utilizzare il property node in scrittura, ossia di modificarne lo stato (e quindi lo stato della proprietà associata) tramite un ulteriore controllo. Ora colleghiamo il controllo booleano *Blink Numeric* al property node.

Questo collegamento farà sì che il valore assunto dal controllo *Blink Numeric* venga direttamente trasferito al nodo di proprietà, attivando o disattivando il lampeggio. Naturalmente, per vedere gli effetti di questo frammento di codice grafico occorre che il programma venga racchiuso in un ciclo, altrimenti terminerebbe subito la sua esecuzione. Creiamo quindi un ciclo while con condizione di uscita collegata al pulsante di stop inserito precedentemente. Il block diagram completo del VI dovrebbe appa-

rire come quello raffigurato nella Fig. 11.

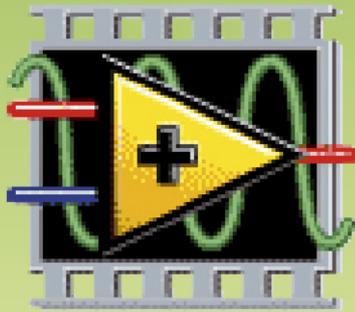
Possiamo a questo punto testare il funzionamento del programma, mandandolo in esecuzione e provando ad intervenire sul controllo *Blink Numeric*. Come si vede, premendo il *Rocker* il controllo numerico inizia a lampeggiare con un periodo di un secondo. Il risultato è visibile nella Fig. 12.

Questo semplice esempio evidenzia le caratteristiche dei property node, tramite i quali è possibile tenere sotto controllo le varie proprietà degli oggetti grafici inseriti nei pannelli frontali.

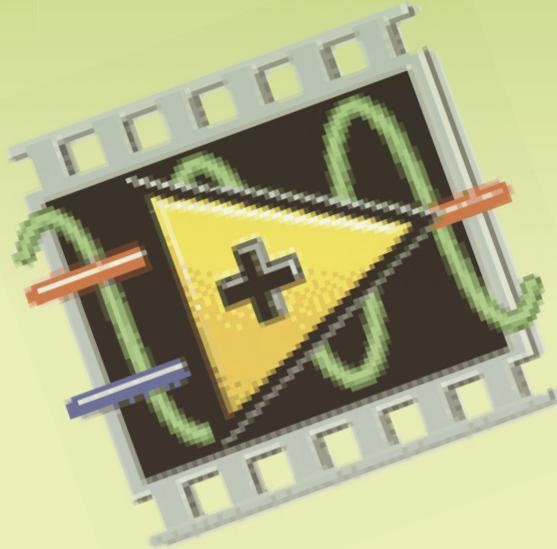
Si tratta di una caratteristica indispensabile, specialmente nei programmi di una certa complessità e che richiedono un certo livello di flessibilità.

Bene, giunti fin qui possiamo tirare le somme: in questa puntata abbiamo completato l'interfaccia di acquisizione per la NI USB-6008 e abbiamo esplorato un'altra caratteristica dell'ambiente di programmazione: i property node. Nella prossima puntata, ultima di questo corso, vedremo altre caratteristiche avanzate dell'ambiente e daremo uno sguardo anche ai toolkit aggiuntivi disponibili. ■

Conoscere e usare LabVIEW



Concludiamo il corso dedicato a LabVIEW spiegando come generare report testuali e grafici tramite l'uso del Linguaggio-G; faremo anche una rapida panoramica sui toolkit disponibili.



di
FRANCESCO
FICILI

Eccoci finalmente giunti alla puntata conclusiva del nostro corso dedicato a LabVIEW, il linguaggio di programmazione che National Instruments ha dedicato alla progettazione di applicazioni riguardanti la misura, l'analisi di segnali e la gestione automatizzata. In queste pagine approfondiremo il discorso sulla generazione automatica di report, tramite i blocchi di *report generation*, e daremo uno sguardo d'insieme ai toolkit aggiuntivi, che permettono di espandere le funzionalità di LabVIEW finora presentate.

REPORT GENERATION IN LABVIEW

Una delle caratteristiche fondamentali di un software per un sistema

automatico di misura è la sua capacità di fornire i risultati in maniera chiara e su supporti convenzionali; ecco perché gli sviluppatori di LabVIEW hanno pensato di dotare la suite di una serie di blocchi per il *report generation*, che consentono di effettuare report stampati o HTML ad un livello altamente professionale. Esiste inoltre un modulo aggiuntivo, chiamato *Report Generation Toolkit for MS Office*, che permette di generare report su MS Word ed MS Excel, offrendo una serie di blocchi aggiuntivi. In questo paragrafo non ci occuperemo del toolkit, del quale peraltro forniremo una descrizione di massima all'interno della panoramica sui toolkit nelle prossime

LabVIEW

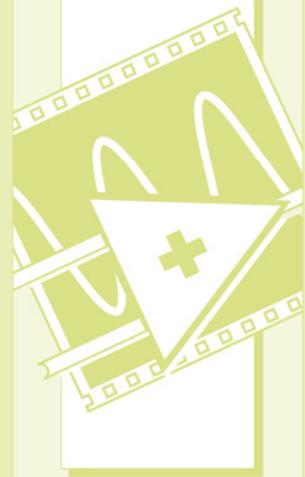
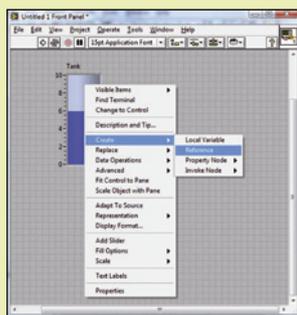




Fig. 1
Report
Generation
Palette.



Fig. 2
Creazione
del
reference.



pagine; piuttosto ci occuperemo dei blocchi base, che utilizzati correttamente e con qualche piccolo trucco permettono di generare report testuali e grafici molto professionali, senza la necessità di utilizzare alcun toolkit aggiuntivo.

I BLOCCHI PER LA GENERAZIONE DEI REPORT IN LABVIEW

LabVIEW dispone nativamente di una serie di blocchi dedicati al *report generation*, che si trovano nella control palette (relativa ai block diagram) all'interno del percorso *Report Generation*. Alcuni di questi blocchi sono accessibili solamente se si dispone del *report generation toolkit*, mentre altri sono attivi direttamente con la semplice suite di base. In pratica, con i blocchi di base è possibile generare report stampati (viene quindi gestito uno spooler di stampa) o HTML (pubblicabili su una pagina web). Nella Fig. 1 è riportata la palette relativa ai blocchi *report generation*.

Come si vede, i blocchi utilizzabili per la generazione dei report sono diversi, tuttavia solo alcuni possono essere utilizzati con la suite base, mentre per sfruttare gli altri è necessario disporre del *report generation toolkit* per Microsoft Office. Qui di seguito facciamo una carrellata sui blocchi più importanti che possono essere utilizzati senza questo toolkit.

- *New Report*: questo è il blocco di partenza, che crea il nuovo report e lo predispone per l'invio allo spooler di stampa o ad altri applicativi (per esempio una pagina di Internet Explorer o un documento di Microsoft Word). La destinazione del report viene decisa in questo blocco, tramite l'ingresso *report type*. Possono essere generati quattro tipi di output differenti, come elencato nella **Tabella 1**.
- *Set Report Font*: permette di settare font, colori, allineamenti e molte altre proprietà del testo del report.
- *Append Report Text*: inserisce delle stringhe di testo all'interno del report. Il report selezionato è quello collegato all'ingresso *report in*.
- *Append Table to Report*: inserisce un array bidimensionale (matrice) all'interno del report, sotto forma di tabella. La larghezza delle celle può essere definita dall'utente.
- *Append Control Image to Report*: inserisce l'immagine di un oggetto del front panel (passato con un reference) all'interno del report. L'oggetto da aggiungere viene selezionato creando una reference (vedere Fig. 2) e passandola all'ingresso *ctrl reference* del blocco.
- *Append Image to Report*: inserisce un'immagine generica all'interno del report. Il path dell'immagine da inserire viene passato tramite l'ingresso *path or URL of image*.

Report Type	Descrizione
0	Standard Report : crea un report solo stampabile.
1	HTML : crea un report HTML, che può essere salvato, pubblicato sul web o stampato.
2	Word : crea un report (nuovo o da un template) su un documento di Microsoft Word, che può essere salvato, modificato o stampato. Necessita del <i>report generation toolkit</i> .
3	Excel : crea un report (nuovo o da un template) su un documento di Microsoft Excel, che può essere salvato, modificato o stampato. Necessita del <i>report generation toolkit</i> .

Tabella 1 • Tipi di report generabili tramite il blocco new report.

- *Dispose Report*: chiude il report e rilascia tutta la memoria da esso impiegata.

GENERAZIONE DI REPORT TESTUALI IN LABVIEW

Vediamo adesso come è possibile generare un report testuale usando i blocchi di *report generation* di LabVIEW. Sfruttiamo, come al solito, un semplice programma di esempio per aiutarvi a comprendere meglio. La nostra idea è di creare un semplice programma che permetta di stampare una serie di stringhe di testo e di intervenire su alcuni aspetti della formattazione (una volta compreso come stampare il testo e controllare la formattazione, è relativamente semplice generare anche report più complessi). Creiamo, allora, un nuovo VI ed inseriamo nel front panel un controllo di tipo stringa, nominandolo "Testo del Report", dimensionato in modo da contenere cinque stringhe di testo. Aggiungiamo poi altrettanti controlli numerici standard (Numeric/Numeric Control), due stereo button (Boolean/Stereo Button), un controllo di tipo color (Numeric/Framed Color Box), tre controlli ring (Ring & Enum/Text Ring) ed infine un pulsante standard. Sistemiamo gli elementi come illustrato nella Fig. 4. I parametri della formattazione del testo che ci proponiamo di controllare sono:

- tipo di report (Standard o HTML);
- margini;
- orientamento della pagina (Portrait o Landscape);
- numero di copie;
- colore del testo;
- opzioni del testo (Grassetto, Corsivo, Sottolineatura).

Vediamo, adesso, come è stato strutturato il block diagram relativo all'esempio in questione. Nella Fig. 4 è riportato il codice in Linguaggio-G del programma corrispondente; il cuore del programma sono i cinque blocchi di *report generation* disposti in cascata all'interno del loop *while*. Le operazioni eseguite dai blocchi sono elencate qui di seguito.

1. Con il primo blocco viene aperto un nuovo report, e viene cablata la variabile che permette di scegliere tra report stampato ed HTML (*Tipo Report*).
2. Il secondo blocco serve a selezionare l'orientamento del testo; viene cablato sul selettore

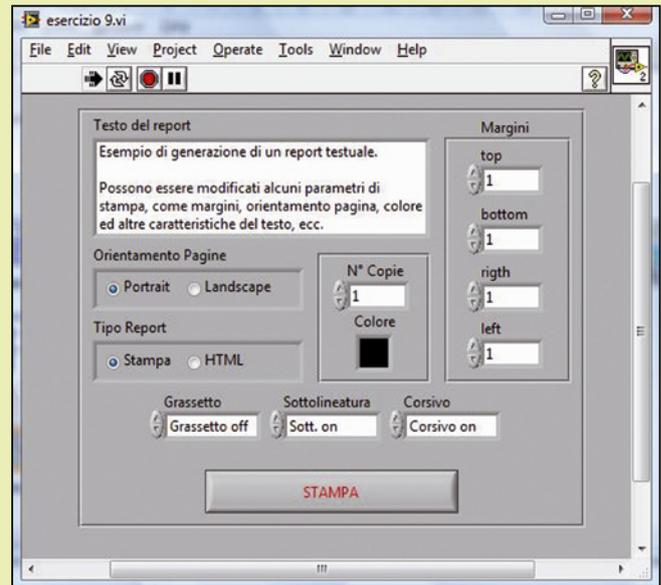


Fig. 3 • Programma di esempio per la generazione di report testuali. Come si può vedere viene generato un report standard, portrait, con margini di 1 cm su tutti i lati, di colore nero con testo in corsivo sottolineato.

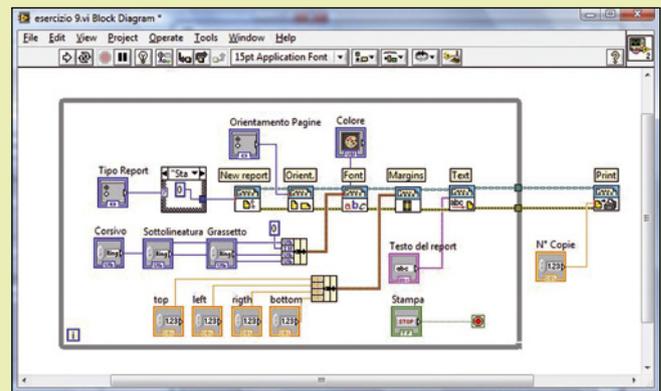


Fig. 4 • Block diagram del programma "esercizio 9".

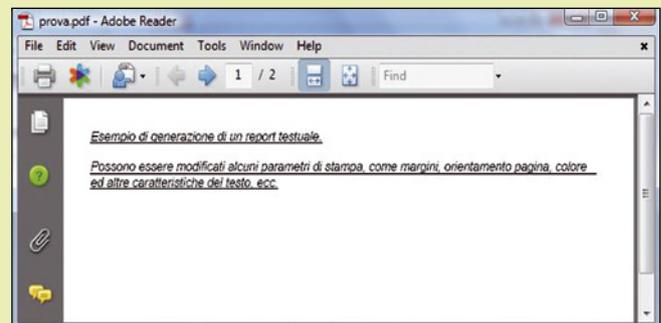


Fig. 5 • Output pdf del programma "esercizio 9".

Per la generazione del pdf si può usare una qualsiasi stampante virtuale "free" per pdf, come, ad esempio, PdfCreator o Win2pdf.

3. Il terzo blocco serve alla selezione del font.

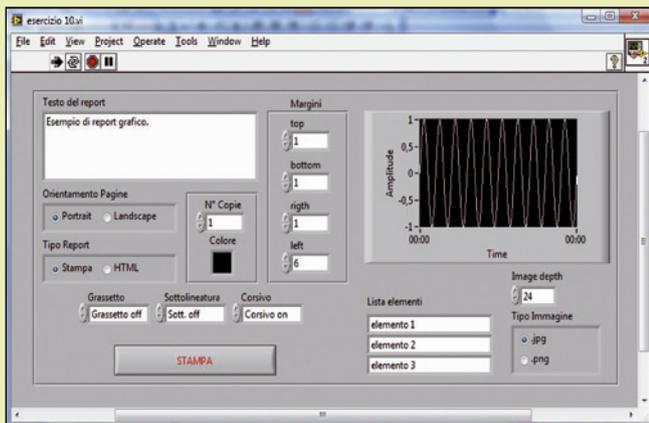


Fig. 6 • Front panel del programma esercizio 10.

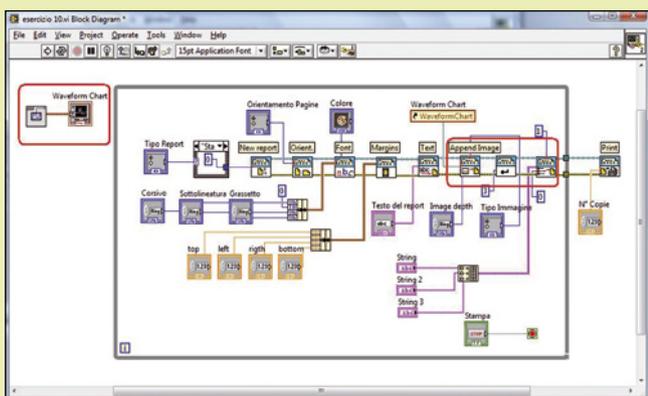


Fig. 7 • Block diagram del programma “esercizio 10”.

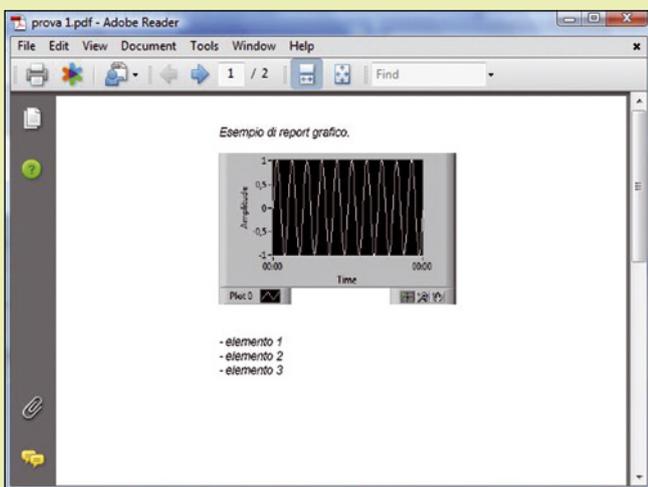


Fig. 8 • Output pdf del programma “esercizio 10”. Per la generazione del pdf si può usare una qualsiasi stampante virtuale per pdf free, come, ad esempio, PdfCreator o Win2pdf.

Su questo blocco vengono cablati i comandi di colore, oltre ai comandi per le opzioni del font (corsivo, sottolineatura, grassetto);

- questi ultimi tre vengono prima raggruppati in un cluster, tramite il blocco *bundle* (Cluster, Class&Variant/Bundle).
4. Il quarto blocco serve all'impostazione dei margini e riceve in input sempre un cluster che raggruppa i quattro selettori numerici che servono per impostare, appunto, i margini della pagina.
 5. Il quinto blocco è quello che effettivamente riceve in input il testo dalla textbox e lo aggiunge al report, secondo le indicazioni che gli provengono dai blocchi precedenti attraverso la linea *report in*. Questa linea deve essere sempre passata da blocco a blocco affinché tutte le impostazioni abbiano effetto.

I cinque blocchi appena descritti vengono racchiusi all'interno di un ciclo *while*, che ha come condizione di uscita la pressione del tasto *stampa*. Sarà così possibile modificare a piacimento i parametri prima di mandare in stampa il report. Appena fuori dal *while*, viene poi inserito il blocco *stampa report*, che prende in ingresso, oltre alle linee di *report in* e di gestione degli errori, anche il numero di copie da stampare.

A questo punto è possibile testare il programma, mandandolo in esecuzione e provando a stampare qualcosa.

Nella Fig. 5 è riportato un esempio di stampa, sotto forma di documento pdf. Per la generazione del pdf è stata usata una stampante virtuale di documenti pdf. Potete provare ad eseguire il programma variando a piacimento le impostazioni e vedere quello che succede. Come è facile intuire, sfruttando i blocchi di stampa e di salvataggio file descritti nelle puntate precedenti, sarebbe possibile anche creare un primitivo programma di elaborazione di testi "Office-like".

GENERAZIONE DI REPORT GRAFICI E LISTE IN LABVIEW

In questo successivo esempio vedremo invece come realizzare dei report con elementi grafici; una volta presa confidenza con i blocchi utilizzati, il procedimento non si discosterà di molto da quello dei report testuali.

Continuiamo a lavorare sull'esempio precedente, rinominandolo *esercizio 10*. Aggiungiamo al front panel un grafico e tre text entry box per simulare gli input di una lista. Al grafico

verrà passato un segnale generato internamente, in modo da simulare l'acquisizione di dati dall'esterno. Nella Fig. 6 è riportata un'immagine del front panel.

Vediamo ora come è stato modificato il block diagram: per prima cosa abbiamo aggiunto un blocco che simula un ingresso sinusoidale (*Signal Processing\Wfm Generation\Basic Function Generator*), in modo da avere dei dati da stampare su grafico; poi sono stati aggiunti alcuni blocchi in coda alla catena di generazione del report creata sull'esempio precedente. Innanzitutto viene aggiunto il grafico al report; un modo molto semplice per inserire un grafico all'interno del report consiste nell'utilizzare il blocco *Append Control Image*, che permette di aggiungere al report l'immagine di un qualsiasi controllo o indicatore (quindi inclusi i grafici). Tra i parametri che si possono passare abbiamo il tipo di immagine generata (jpeg o png) e la profondità dei pixel. Abbiamo poi aggiunto un blocco con una separazione di tre righe (*New Report Line*) ed infine in blocco *Append List to Report*, che prende in ingresso l'array generato dalla fusione delle tre text entry box inserite in precedenza. Nella Fig. 7 è riportato il block diagram con le modifiche descritte. Bene, con questi pochi e semplici passi abbiamo introdotto anche delle immagini al nostro report; il risultato è visibile nella Fig. 8.

PANORAMICA SUI MODULI AGGIUNTIVI

LabVIEW dispone di una serie di moduli e di toolkit aggiuntivi, acquistabili separatamente (ma sono comunque disponibili delle versioni di valutazione a scadenza) che permettono di espandere le funzionalità della suite di sviluppo. Vediamo qui di seguito quali sono i più importanti.

- *LabVIEW Mobile Module*: questo modulo è utilizzato per sviluppare applicazioni per i più recenti handheld device e smartphone. Con questo toolkit aggiuntivo è possibile creare sistemi di misura portatili che acquisiscono, analizzano e visualizzano dati, usando i dispositivi della linea DAQ NI. Il modulo consente inoltre di sfruttare le connessioni wireless (comunemente presenti su PDA e smartphone) per interfacciarsi con strumentazione esistente.
- *LabVIEW Real-Time Module*: il modulo real-time di LabVIEW, unito alla serie RT dei

Fig. 9
Esempio di handheld device sviluppato con il Mobile Module.



Fig. 10
Sviluppo applicazioni con LabVIEW Real-Time module.

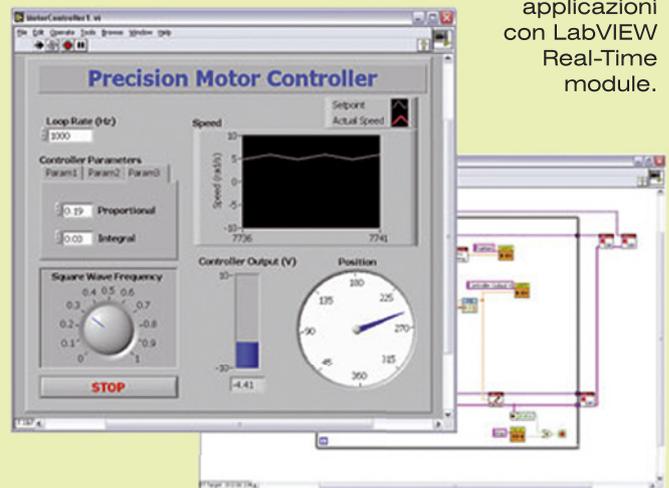


Fig. 11 • Evaluation kit NI per LabVIEW FPGA Module.

prodotti hardware NI, permette lo sviluppo di applicazioni deterministiche, con performance di tipo real-time, per applicazioni di



Fig. 12
Esempio di nodi WSN NI.

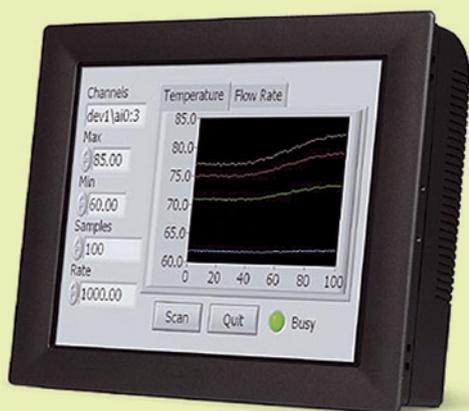


Fig. 13
Esempio di applicazione sviluppata con il Touch Panel Module.

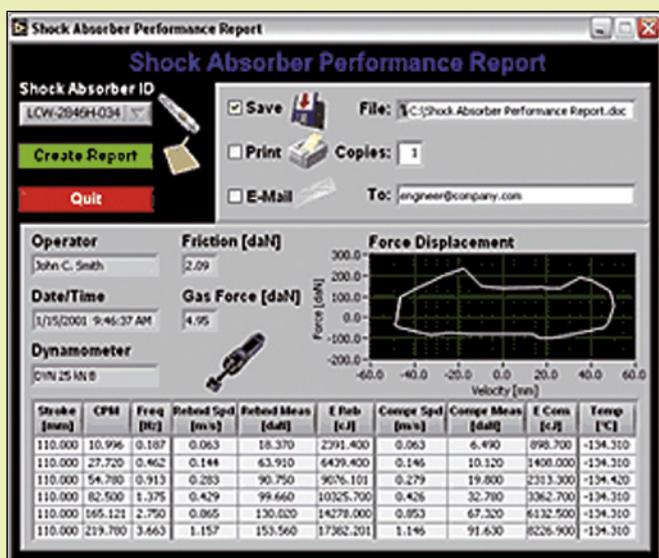


Fig. 14 • Esempio di VI che fa uso del Report Generation Toolkit for MS Office.

misura e controllo. Sfruttando le caratteristiche della programmazione grafica tramite il Linguaggio-G, è possibile sviluppare la

propria applicazione di controllo su una macchina desktop e trasferirla su un target indipendente. L'hardware real-time proposto da NI comprende un processore embedded, dotato di un suo RTOS, memoria onboard, local storage ed accesso a periferiche come porte seriali, ethernet, GPIB, USB, ecc.

- **LabVIEW FPGA Module:** questo modulo aggiuntivo permette di estendere le funzionalità di LabVIEW e della programmazione grafica alle FPGA (Field Programmable Gate Arrays) dei moduli NI RIO (Reconfigurable IO). La programmazione tramite LabVIEW è particolarmente indicata per le logiche programmabili, in quanto realizza in maniera nativa i parallelismi ed il flusso dati. Tramite questo modulo è possibile realizzare hardware customizzato per applicazioni di misura e controllo, senza dover utilizzare linguaggi di basso livello per la gestione dell'hardware. L'uso di FPGA permette di realizzare applicazioni ad altissima velocità, interfacciarsi con protocolli di comunicazione standard, gestire comunicazioni a radiofrequenza e molto altro ancora.
- **LabVIEW WSN Module:** le Wireless Sensor Network (WSN) costituiscono un interessantissimo campo di applicazione delle tecnologie elettroniche che in questi ultimi anni sono diventate di grande attualità. Il modulo NI WSN Pioneer estende le funzionalità del linguaggio alla programmazione dei nodi NI, permettendo lo sviluppo e la distribuzione di applicazioni basate su WSN. Usando il WSN Module è possibile estendere la vita delle batterie ottimizzando il funzionamento dei dispositivi, includere routine di analisi ed acquisizione dati, fornire intelligenza e facoltà decisionali ai singoli nodi. Il modulo fornisce, inoltre, la possibilità di aggiornare il software residente sui dispositivi in remoto, sfruttando il canale wireless.
- **LabVIEW Touch-Panel Module:** il modulo Touch Panel di LabVIEW permette lo sviluppo di applicazioni Human Machine Interface (HMI) in grado di comunicare con i moduli hardware real time di National Instruments, come ad esempio CompactFieldPoint, Compact RIO e CompactVisionSystem. I sistemi HMI costituiscono un'efficace sistema per la visualizzazione delle informazioni ed il controllo delle applicazioni.

- *LabVIEW Machine Vision Module*: il *Machine Vision Module* è una ricca libreria con centinaia di algoritmi di image-processing e computer vision. I blocchi presenti nella libreria coprono problematiche di computer vision come l'immagine enhancements, il controllo di presenza, l'identificazione di oggetti e la loro localizzazione nello spazio.
- *LabVIEW Report Generation Toolkit*: il Report generation Toolkit for MS Office di LabVIEW è una libreria di VI che permettono la realizzazione di report su MS Word e MS Excel. Oltre ai blocchi base, il toolkit è stato arricchito da una serie di blocchi express altamente specializzati che permettono lo sviluppo di report customizzati in un tempo ancora inferiore.
- *LabVIEW Robotics Module*: il Robotics Module è un toolkit aggiuntivo che permette la progettazione di software per il controllo robotico. Il modulo comprende una libreria per la gestione di sensori ed attuatori di uso comune in robotica, algoritmi di controllo e funzioni di movimento.

Alcune delle tipologie di robot di movimento realizzabili sono le seguenti:

- veicoli autonomi e semiautonomi, inclusi veicoli agricoli e militari;
 - piattaforme di soccorso;
 - veicoli sottomarini ed UAV (Ummanned Aerial Veicles);
 - robot di servizio;
 - dispositivi robotici per il settore medico.
- *LabVIEW Control Design and Simulation Module*: tramite questo modulo è possibile analizzare il comportamento in anello aperto di un determinato sistema, progettarne il controllore in anello chiuso, simulare il sistema e realizzarne l'implementazione fisica.
 - *LabVIEW Datalogging and Supervisory Control Module*: il modulo DSC (Datalogging and Supervisory Control) aggiuntivo di LabVIEW è lo strumento ideale per lo sviluppo di applicazioni HMI/SCADA o di datalogging con alto numero di canali in ingresso. Il modulo include tools per il datalogging, controllo real-time, controllo di allarmi ed eventi ed altro ancora.

I moduli descritti sono alcuni tra i più importanti messi a disposizione da NI per estendere

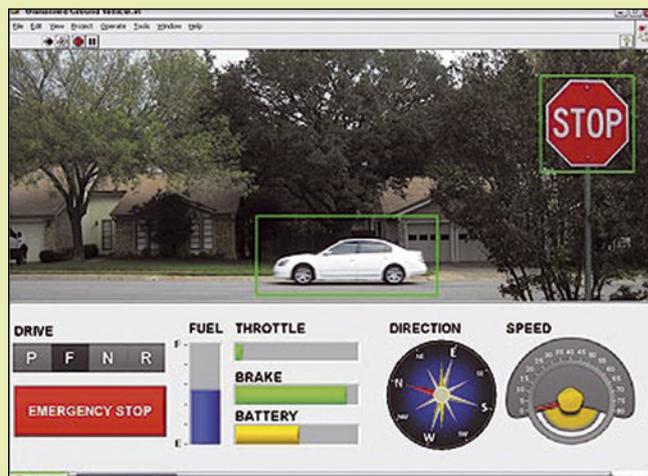


Fig. 15 • Esempio di applicazione sviluppata con il Robotics Module di LabVIEW.

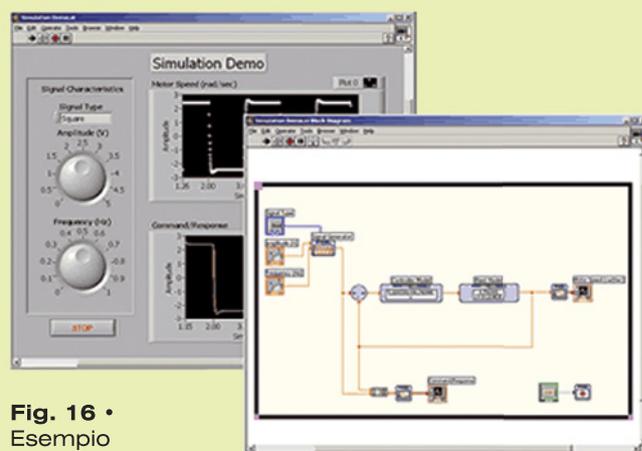
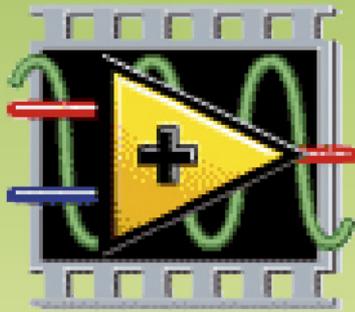


Fig. 16 • Esempio di progetto di un controllo tramite il CDS Module.

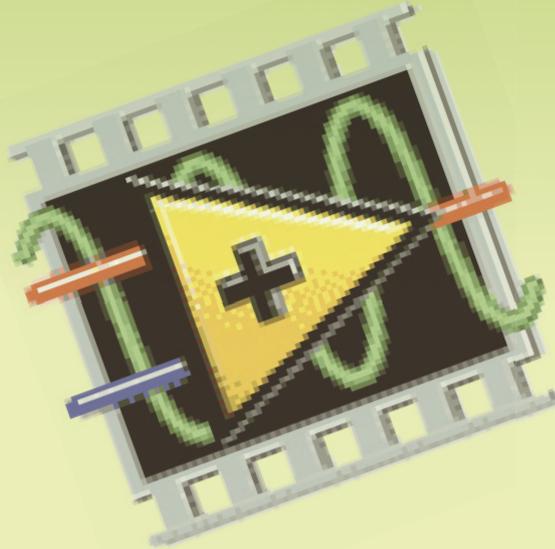
le funzionalità di LabVIEW. Come si è visto, si tratta di moduli altamente specializzati per facilitare lo sviluppo di specifiche applicazioni; questo consente di ridurre i costi, in quanto non è necessario acquisire una suite completa, ma si possono acquistare solo i moduli strettamente necessari.

Bene, con questa panoramica sui toolkit si conclude quest'ultima puntata del corso di programmazione LabVIEW; durante le otto puntate abbiamo cercato di introdurvi a questo innovativo approccio alla programmazione, supportandovi con le basi della programmazione grafica e guidandovi alla stesura di alcuni primi e semplici programmi di esempio. Inoltre abbiamo cercato di far luce sulle varie componenti aggiuntive di una suite, che, presa nella sua totalità, è decisamente ampia e mette nelle mani degli sviluppatori uno strumento estremamente completo. ■

Conoscere e usare LabVIEW



Concludiamo il nostro corso accennando a LabVIEW 2010, l'ultima release, che presenta interessanti novità rispetto al passato, come ad esempio migliorie apportate al compilatore, nuovi programmi per soddisfare ed allo stesso tempo utilizzare il contributo degli sviluppatori.



di
FRANCESCO
FICILI

Il corso di programmazione base in LabVIEW, che ha consentito ai nostri lettori di familiarizzare con questo innovativo ambiente di programmazione, apprendere tecniche di acquisizione dati e imparare a sfruttare le potenzialità della programmazione grafica per sviluppare rapidamente applicazioni anche complesse, è stato concepito e preparato prendendo come riferimento la versione 2009. Ciò perché come piattaforma per il corso abbiamo dovuto utilizzare l'ambiente che -alla data di pubblicazione della prima puntata- risultava essere l'ultima versione di LabVIEW rilasciata, ossia, appunto, LabVIEW 2009. Tuttavia

il reparto R&D di National Instruments non si stanca mai di sfornare nuovi prodotti software ed aggiornamenti, ed ecco che mentre noi stavamo giungendo alla fine di questo corso di programmazione, veniva resa disponibile una nuova release dell'ambiente di programmazione: LabVIEW 2010.

In questa puntata conclusiva vogliamo quindi porre in evidenza quali sono le novità ed i miglioramenti apportati all'ambiente di programmazione grafica di National Instruments con quest'ultima release.

Vediamo dunque le più significative innovazioni introdotte con la versione 2010.

LabVIEW



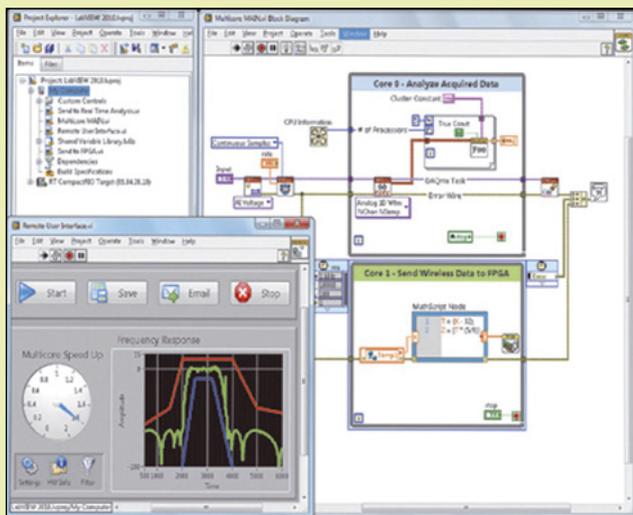


Fig. 1 - Sviluppo di un VI in LabVIEW.

COMPILATORE OTTIMIZZATO

Una delle novità più interessanti di LabVIEW 2010 risiede nel compilatore; per comprenderne la portata bisogna dire che LabVIEW è un linguaggio compilato, cosa che può sorprendere perché, durante il tipico sviluppo di un VI in linguaggio-G, non si nota una fase esplicita di compilazione. Al contrario, potete cambiare il vostro VI e premere semplicemente il pulsante *Run* per eseguirlo. Compilazione significa che il codice scritto in linguaggio-G viene tradotto in codice macchina nativo e che quindi può essere eseguito direttamente dal computer host. Un'alternativa a questo approccio è l'interpretazione, dove i programmi sono eseguiti direttamente da un altro programma software (chiamato interprete) anziché direttamente dal computer; le primissime versioni di LabVIEW non facevano uso di un compilatore ma di un interprete.

L'uso di un compilatore a scapito di un interprete presenta vantaggi e svantaggi.

Un interprete è tipicamente più facile da progettare e mantenere nel corso degli anni, ma presenta lo svantaggio di tempi di esecuzione più lenti. Al contrario, un compilatore è tipicamente più complesso da sviluppare, ma offre il vantaggio di una maggiore velocità di esecuzione del codice.

Si capisce, pertanto, come nel corso degli anni questa caratteristica dell'ambiente sia mutata, attraversando addirittura una fase di transizione durante la quale il linguaggio, da interpretato è diventato compilato. Passiamo quindi attraverso un breve excursus storico di LabVIEW, dal punto di vista del compilatore, per poi analizzare come si compone la catena

di compilazione di LabVIEW 2010. LabVIEW è stato introdotto nel 1986 e, come già detto, nella sua prima versione (LabVIEW 1.0) usava un interprete ed era eseguibile solo sui microprocessori Motorola 68000. A quel tempo, il nuovo linguaggio LabVIEW era molto più semplice, cosa che alleviava anche i requisiti del compilatore (al tempo, un interprete).

Per esempio, non c'era polimorfismo e l'unico tipo numerico era quello in virgola mobile a precisione estesa. La release LabVIEW 1.1 vide l'introduzione dell'algoritmo *inplaceness*, o dell'*inplacer*, che identifica le allocazioni dei dati che potete riutilizzare durante l'esecuzione, evitando inutili copie dei dati e, di conseguenza, velocizzando spesso nettamente le prestazioni di esecuzione.

In LabVIEW 2.0, l'interprete è stato sostituito con un effettivo compilatore. Pur rimanendo eseguibile ancora esclusivamente sui Motorola 68000, LabVIEW poteva generare codice macchina nativo. Nella versione 2.0 è stato aggiunto anche l'algoritmo di propagazione dei tipi di dato, che, fra le altre cose, gestisce il controllo della sintassi e la risoluzione dei tipi sul linguaggio LabVIEW in crescita. Un'altra grossa innovazione in LabVIEW 2.0 è stata l'introduzione del *clumper*. L'algoritmo di clumping identifica il parallelismo nel diagramma LabVIEW e raggruppa i nodi in "clump", che possono essere eseguiti in parallelo. Gli algoritmi di propagazione dei tipi, *inplaceness* e *clumping* continuano a essere componenti importanti del moderno compilatore LabVIEW e hanno visto numerosi miglioramenti incrementali nel tempo. L'infrastruttura del nuovo compilatore in LabVIEW 2.5 ha aggiunto supporto per back-end multipli, specificamente Intel x86 e SPARC. LabVIEW 2.5 ha inoltre introdotto il linker, che gestisce le dipendenze fra VI per tracciarli quando devono essere ricompilati. Due nuovi back-end, PowerPC e HP PA-RISC, sono stati aggiunti -insieme al folding delle costanti- nella versione 3.1 di LabVIEW.

LabVIEW 5.0 e 6.0 hanno ridato slancio al generatore di codice e hanno aggiunto la GenAPI, un'interfaccia comune ai molteplici back-end; la GenAPI permette la compilazione incrociata, cosa importante per lo sviluppo real-time. Gli sviluppatori real-time tipicamente scrivono i VI su un PC host ma li installano (e li compilano per) un target real-time. Inoltre, è stata inclusa

una forma limitata di spostamento del codice loop-invariante. Infine, il sistema di esecuzione multitasking di LabVIEW è stato esteso per supportare thread multipli.

LabVIEW 8.0 si è basato sull'infrastruttura GenAPI introdotta nella Version 5.0 per aggiungere un algoritmo di allocazione dei registri; prima dell'introduzione della GenAPI, i registri erano hard-coded nel codice generato per ogni nodo. Sono state introdotte anche forme limitate di eliminazione del codice irraggiungibile e del codice morto. LabVIEW 2009 ha apportato un novità LabVIEW a 64 bit e la Dataflow Intermediate Rappresentazione (DFIR). La DFIR è stata immediatamente usata per costruire forme più avanzate di spostamento del codice loop-invariante, folding delle costanti, eliminazione del codice morto e del codice irraggiungibile. Nuove caratteristiche del linguaggio introdotte in 2009, come il For Loop parallelo, sono state basate sulla DFIR.

IL COMPILATORE DI LABVIEW 2010

Dopo aver esaminato brevemente l'exkursus storico del compilatore, vediamo adesso come funziona, a grandi linee, il processo di compilazione di LabVIEW 2010.

Il primo passo della compilazione di un VI è l'algoritmo di propagazione dei tipi di dato, che ha il compito di riconoscere i tipi di dati implicati per i terminali che possono adattarsi ai tipi di dato; contestualmente vengono rilevati gli errori di sintassi. Se l'algoritmo determina che il VI è valido, la compilazione continua e si procede con il pass successivo. Dopo la propagazione dei tipi di dato il diagramma a blocchi dell'editor viene convertito nella DFIR usata da compilatore.

Dopo tale conversione, il compilatore esegue diverse trasformazioni sul grafico DFIR per decomporlo, ottimizzarlo e predisporlo alla generazione del codice. Alcune ottimizzazioni consentite dalla DFIR sono la riassociazione algebrica, l'eliminazione delle sottoespressioni comuni, il loop unrolling e l'inlining dei subVI. Nella Versione 2010 di LabVIEW è stata introdotta anche la LLVM (Low Level Virtual Machine), ossia Macchina Virtuale a Basso Livello.

La LLVM è una infrastruttura di compilazione open-source, che trova largo impiego nell'industria. L'introduzione della LLVM ha

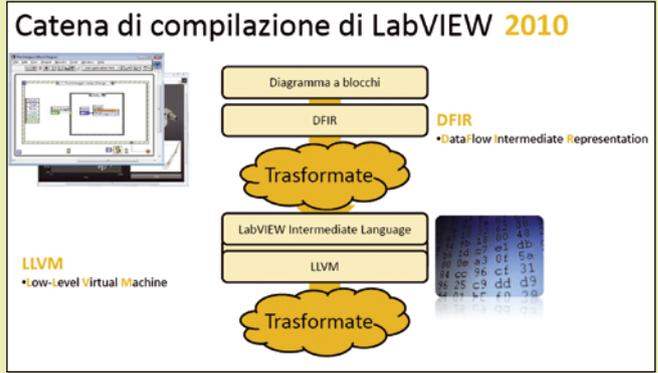


Fig. 2 - Catena di compilazione di LabVIEW 2010.

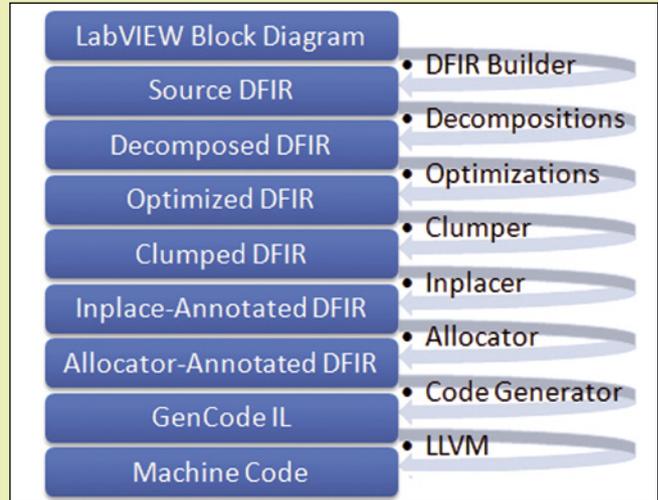


Fig. 3 - Dettaglio della procedura di compilazione.

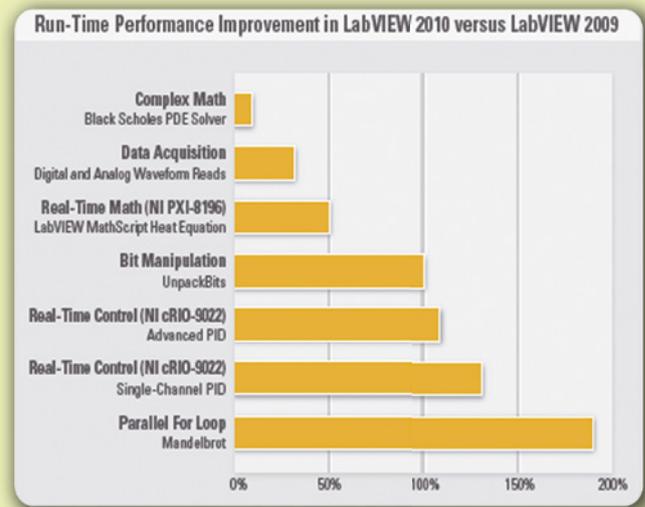


Fig. 4 - Performance improvement in LabVIEW 2010.

permesso l'aggiunta di nuove ottimizzazioni come la pianificazione delle istruzioni, il loop unswitching, la combinazione delle istruzioni, la propagazione condizionata ed un allocatore

di registri più sofisticato. Nella Fig. 2 è rappresentata la catena di compilazione di LabVIEW 2010: come si vede, il diagramma a blocchi di LV viene tradotto nella Dataflow Intermediate Representation (DFIR), la quale produce un "linguaggio intermedio" che diventa l'input della LLVM utilizzato per la creazione del codice eseguibile finale.

Nella Fig. 3 è riportata la procedura dettagliata di compilazione, mentre nella Fig. 4 è riportato un grafico che sintetizza i miglioramenti nelle performance dell'ambiente durante il passaggio tra la versione 2009 e la versione 2010.

SVILUPPO DI LABVIEW E PARTNERSHIP

Un'importantissima iniziativa che vede la luce con il rilascio di LabVIEW 2010, è il LabVIEW Add-On Developer Program, nato con lo scopo di fornire a migliaia di partner l'opportunità di espandere la piattaforma e introdurre funzionalità personalizzate in LabVIEW. Il programma consiste nella creazione di un mercato online costituente una parte evoluta del LabVIEW Tools Network per gli sviluppatori, con l'intento di offrire -gratuitamente e a pagamento- i propri toolkit, e una sorta di luogo di scambio per visionare, scaricare, valutare e acquistare degli add-on di LabVIEW. Sono disponibili oltre 50 add-on di sviluppatori National Instruments e di terze parti, incluse delle librerie per il riutilizzo del codice, template, controlli e connettori di interfacce utente per altri pacchet-

ti software. Tale iniziativa testimonia la sempre crescente attenzione di National Instruments verso il codice prodotto dai suoi sviluppatori e la volontà di favorire la crescita dei giovani programmatori che utilizzano questo innovativo ambiente di sviluppo.

LABVIEW IDEA EXCHANGE

LabVIEW Idea Exchange è un'iniziativa promossa da National Instruments, volta a raccogliere i vari feedback dei clienti e degli utilizzatori di LabVIEW per migliorare la progettazione dei nuovi ambienti. Durante la fase di sviluppo di LabVIEW 2010, il dipartimento R&D di NI si è avvalso di idee e suggerimenti raccolti tramite LabVIEW Idea Exchange (ni.com/ideas) e sono state implementate ben 14 proposte, alcune delle quali rivolte al miglioramento della documentazione ed all'ottimizzazione del codice.

In questo modo LabVIEW 2010 utilizza i feedback dei clienti per offrire nuove caratteristiche che ne facilitano i primi passi. Ad esempio, LabVIEW include ora un nuovo strumento di configurazione hardware che consente di accedere ai propri target LabVIEW Real Time e organizzarli in remoto, via web. Tra le altre caratteristiche sono presenti un installer intelligente che rileva automaticamente il software associato al numero seriale, per consentire una rapida installazione, ed un motore di ricerca ottimizzato per driver di strumenti, che offre esempi di progetti precompilati per strumenti specifici.

MAGGIORE FUNZIONALITÀ PER LO SVILUPPO DI APPLICAZIONI DI GRANDI DIMENSIONI

Per gli utenti e i gruppi di sviluppo più avanzati, LabVIEW 2010 include nuove caratteristiche, le quali ottimizzano le interfacce per il riutilizzo del codice, il raggruppamento di VI e le relative gerarchie che accelerano i tempi di realizzazione e separano il codice sorgente del VI dalla versione compilata per favorire la gestione del codice sorgente. Queste funzionalità sono ideali per

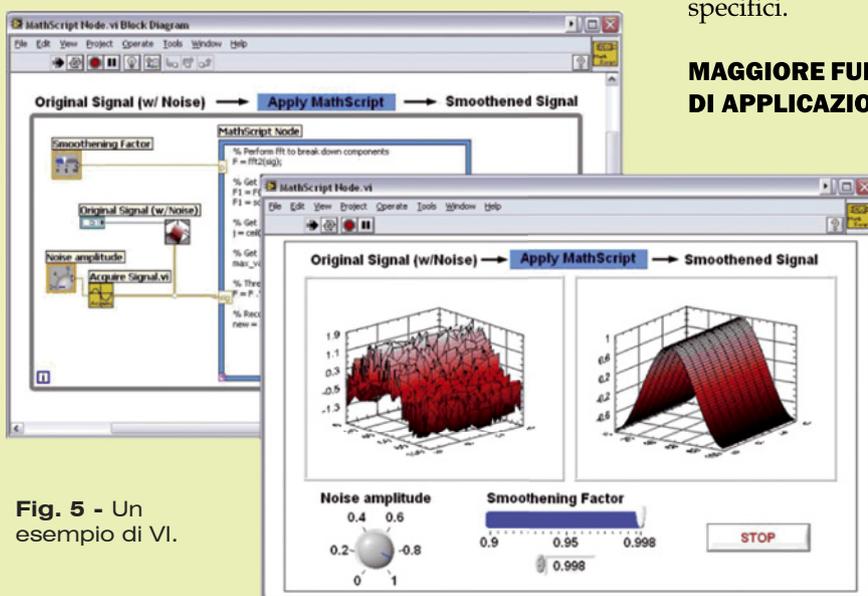


Fig. 5 - Un esempio di VI.

lo sviluppo di applicazioni che coinvolgono un elevato numero di persone, dove la manutenzione del codice attraverso utenti, versioni dei software e piattaforme informatiche diverse, è effettivamente critica.

Bene, in queste pagine abbiamo visto sinteticamente quali sono le novità apportate all'ambiente nella transizione tra la versione 2009 e la versione 2010. Da qualche mese è inoltre disponibile la Student Edition di LabVIEW, che consente anche a studenti ed hobbisti di utilizzare questo potentissimo ambiente di sviluppo per i propri esperimenti, ad un costo decisamente accessibile. Il pacchetto LabVIEW 2010 Student Edition comprende i seguenti software:

- LabVIEW 2010 Student Edition for Windows 7 / Vista / XP;
- LabVIEW Control Design and Simulation Module;
- LabVIEW MathScript RT Module;
- LabVIEW Digital Filter Design Toolkit;
- LabVIEW Modulation Toolkit;
- LabVIEW System Identification Toolkit;
- LabVIEW SignalExpress;
- NI Vision Development Module;
- NI Vision Builder for Automated Inspection (AI);
- NI Vision Acquisition Software.

Come vedete, la Student Edition non è limitata in quanto con essa è possibile fare pratica anche con i tool più complessi.

Bene, con questa puntata di aggiornamento termina il nostro corso "Conoscere ed Usare LabVIEW", ma vi diamo appuntamento a breve con i primi progetti pratici coinvolgenti questo ambiente. ■

Fig. 6 - Esempio di robot bipede sviluppato sfruttando le potenzialità software di LabVIEW.

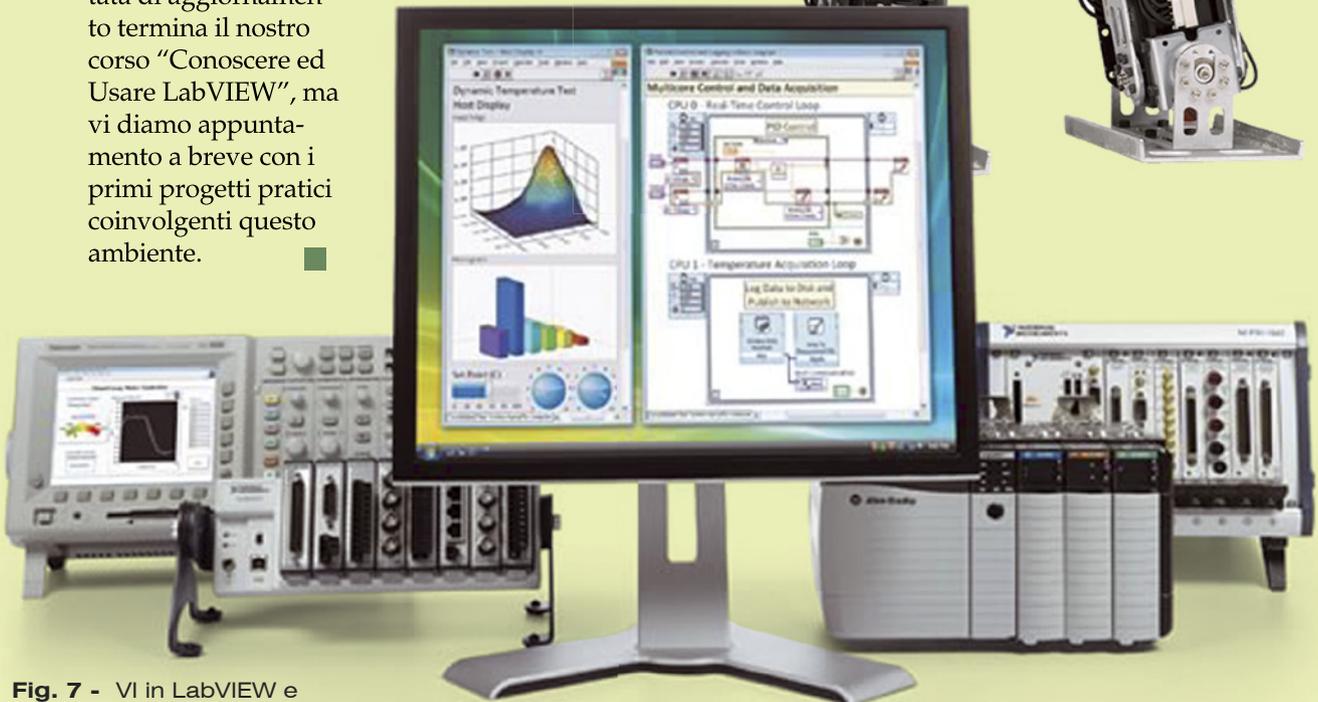
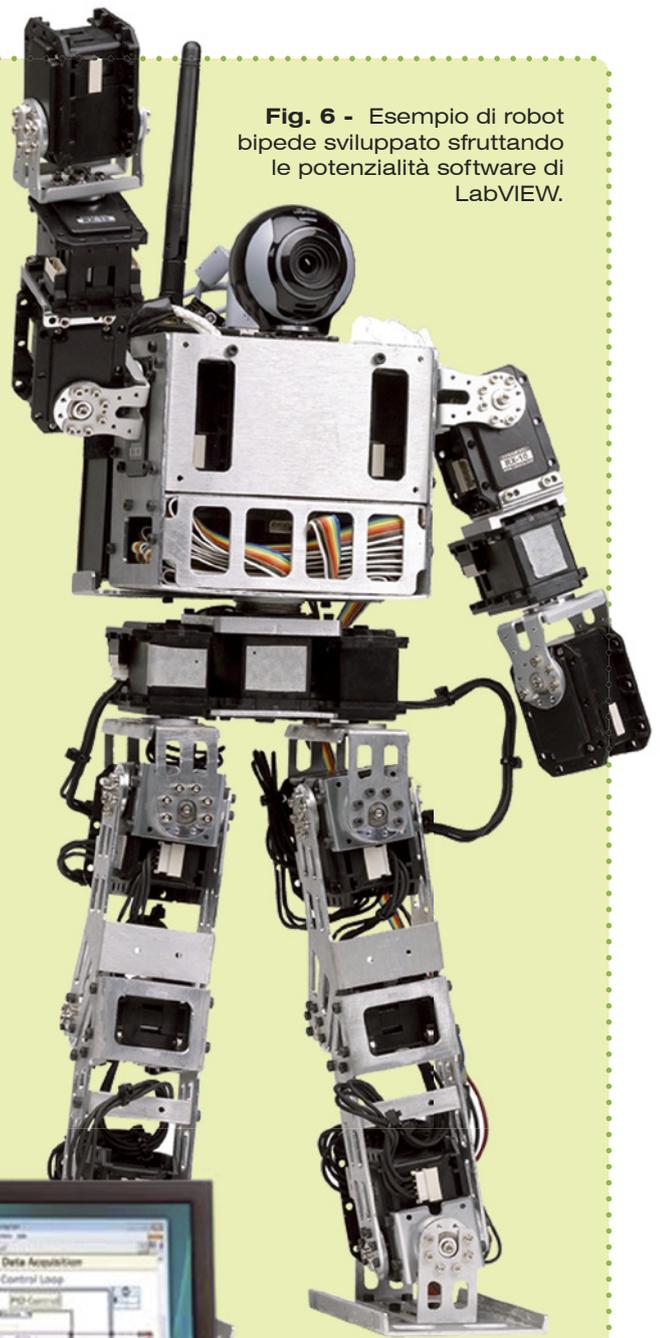
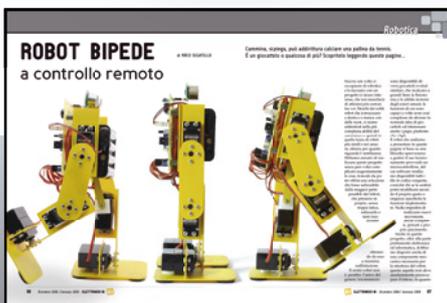


Fig. 7 - VI in LabVIEW e scheda di acquisizione dati.

La più prestigiosa rivista di progettazione elettronica



Elettronica applicata



Attualità scientifiche



Novità tecnologiche

Abbonamenti, numeri arretrati,
download su www.elettronica.in

TUTTI I MESI IN EDICOLA