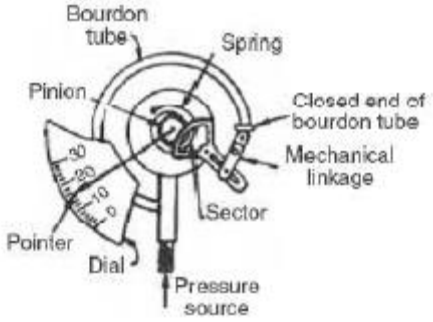
Unit III

Virtual Instrumentation System

Instrumentation System

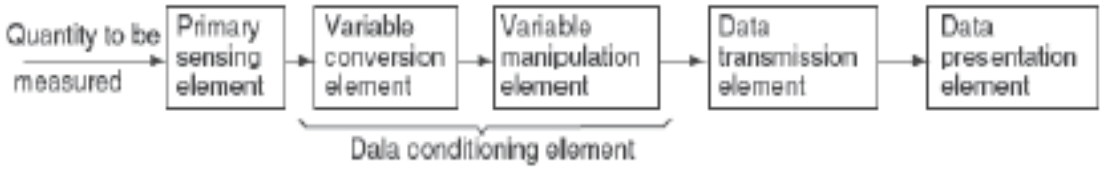
An instrument is a device designed to collect data from an environment, or from a unit under test, and to display information to a user based on the collected data. Such an instrument may employ a transducer to sense changes in a physical parameter, such as temperature or pressure, and to convert the sensed information into electrical signals, such as voltage or frequency variations. The term instrument may also be defined as a physical software device that performs an analysis on data acquired from another instrument and then outputs the processed data to display or recording devices. This second category of recording instruments may include oscilloscopes, spectrum analyzers, and digital millimeters. The types of source data collected and analyzed by instruments may thus vary widely, including both physical parameters such as temperature, pressure, distance, frequency and amplitudes of light and sound, and also electrical parameters including voltage, current, and frequency.

Instrumentation system is a combination of instruments or components which act together to perform certain set of operation. Instrumentation systems have generally consisted of individual instruments, for example, pressure gauge which comprising a sensing transducer i.e bourdon tube wired to signal conditioning circuitry, outputs a processed signal to a display panel, creating a time record of pressure changes.



Functional element of instrumentation system

Functional element of instrumentation system



Primary sensing element

The quantity under measurement makes its first contact with the primary sensing element of a measurement system. In other words the measurand is first detected by primary sensor. This act is then immediately followed by the conversion of measurand into an analogous electrical signal performed by a transducer.

Variable conversion element

The output of the primary sensing element may be electrical signal such as a voltage, a frequency or some other electrical parameter. Sometimes this output is not suited to the system. For the instrument to perform the desired function, it may be necessary to convert this output to some other suitable form while preserving the information content of the original signal.

Variable manipulation element

The function of this element is to manipulate the signal presented to it preserving the original nature of the signal. Manipulation here means only a change in numerical value of the signal. For example, an electronic amplifier accepts a small voltage signal as input and produces an output signal which is also voltage but of greater magnitude. Thus voltage amplifier acts as a variable manipulation element.

Data Transmission Element

When the elements of an instrument are actually physically separated, it becomes necessary to transmit data from one to another. The element that performs this function is called a Data Transmission Element.

Data presentation element

The information about the quantity under measurement has to be conveyed to the personnel handling the instrument or the system for monitoring, control, or analysis purposes. The information conveyed must be in a form intelligible to the personnel or to the intelligent instrumentation system. This function is done by data presentation element.

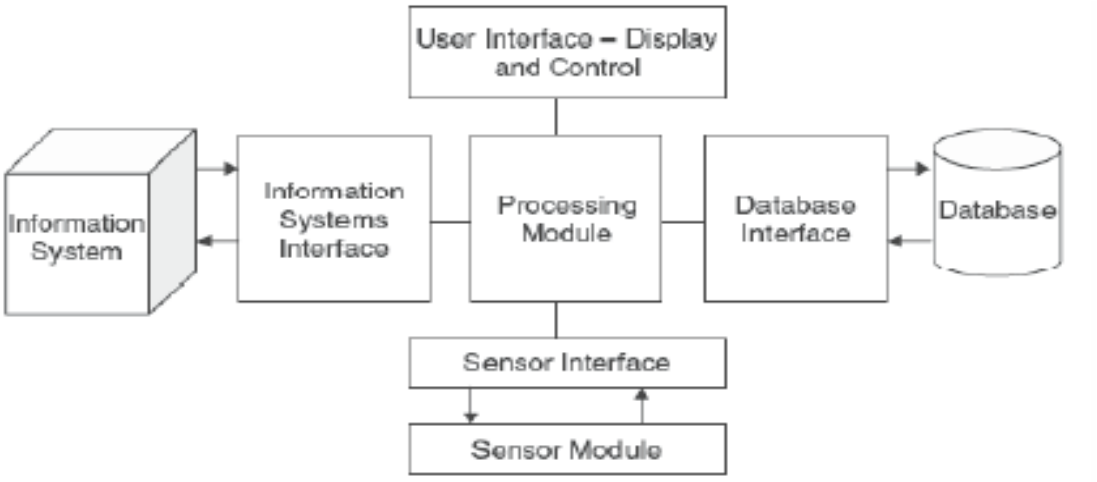
Virtual Instrumentation

A virtual instrumentation system is software that is used by the user to develop a computerized test and measurement system, for controlling an external measurement hardware device from a desktop computer, and for displaying test or measurement data on panels in the computer screen. The test and measurement data are collected by the external device interfaced with the desktop computer. Virtual instrumentation also extends to computerized systems for controlling processes based on the data collected and processed by a PC based instrumentation system.

Architecture of Virtual Instrumentation

A virtual instrument is composed of the following blocks:
– Sensor module
– Sensor interface
– Information systems interface
– Processing module
– Database interface
– User interface

Figure 1.1 shows the general architecture of a virtual instrument.



Sensor module

The sensor module detects physical signal and transforms it into electrical form, conditions the signal, and transforms it into a digital form for further manipulation. The sensor module interfaces a virtual instrument to the external, mostly analog world transforming measured signals into computer readable form. A sensor module principally consists of three main parts:

– The sensor

– The signal conditioning part

– The A/D converter

The sensor detects physical signals from the environment.

The signal-conditioning module performs (usually analog) signal conditioning prior to AD conversion. This module usually does the amplification, transducer excitation, linearization, isolation, or filtering of detected signals.

The A/D converter changes the detected and conditioned voltage into a digital value.

Sensor interface

Through a sensor interface, the sensor module communicates with a computer. There are many interfaces used for communication between sensors modules and the computer. According to the type of connection, sensor interfaces can be classified as wired and wireless.

Processing module

Once the data are in a digital form on a computer, they can be processed, mixed, compared, and otherwise manipulated, or stored in a database. Then, the data may be displayed, or converted back to analog form for further process control.

Database Interface

Computerized instrumentation allows measured data to be stored for off-line processing, or to keep records as a part of the patient record. There are several currently available database technologies that can be used for this purpose. Many virtual instruments use DataBase

Management Systems (DBMSs)

They provide efficient management of data and standardized insertion, update, deletion, and selection.

Display and control

Without the displays, knobs and switches of conventional, external box-based instrumentation products, a virtual instrument uses a personal computer for all user interaction and control. In many common measurement applications, a data acquisition board or card, with a personal computer and software, can be used to create an instrument.

Traditional versus virtual instruments

A traditional instrument is designed to collect data from an environment, or from a unit under test, and to display information to a user based on the collected data. Such an instrument may employ a transducer to sense changes in a physical parameter such as temperature or pressure, and to convert the sensed information into electrical signals such as voltage or frequency variations.

A virtual instrument (VI) is defined as an industry-standard computer equipped with userfriendly application software, cost-effective hardware and driver software that together perform the functions of traditional instruments. Simulated physical instruments are called virtual instruments (VIs). Virtual instrumentation software based on user requirements defines general-purpose measurement and control hardware functionality. With virtual instrumentation, engineers and scientists reduce development time, design higher quality products, and lower their design costs. In test, measurement and control, engineers have used virtual instrumentation to downsize automated test equipment (ATE) while experiencing up to a several times increase in productivity gains at a fraction of the cost of traditional instrument solutions.

Traditional instruments	Virtual instruments
➢ Vendor-defined	User-defined
➢ Function-specific, stand-alone with limited connectivity	Application-oriented system with connectivity to networks, peripherals, and applications
➢ Hardware is the key	Software is the key
➢ Expensive	Low-cost, reusable
➢ Closed, fixed functionality	Open, flexible functionality leveraging off familiar computer technology
➢ Slow turn on technology (5–10 year life cycle)	Fast turn on technology (1–2 year life cycle)
➢ Minimal economics of scale	Maximum economics of scale
➢ High development and maintenance costs	Software minimizes development and maintenance costs

Software

A Virtual Instrument (VI) is a Lab VIEW programming element. Lab VIEW (Laboratory VirtualInstrument Engineering Workbench) is a graphical programming environment which has become prevalent throughout research labs, academia and industry. It is a powerful and versatile analysis and instrumentation software system for measurement and automation.

A VI consists of a front panel, block diagram, and an icon that represents the program. The front panel is used to display controls and indicators for the user, and the block diagram contains the code for the VI. The icon, which is a visual representation of the VI, has connectors for program inputs and outputs. Programming languages such as C and BASIC use functions and subroutines as programming elements. Lab VIEW uses the VI. The front panel of a VI handles the function inputs and outputs, and the code diagram performs the work of the VI. Multiple VIs can be used to create large scale applications; in fact, large scale applications may have several hundred VIs. A VI may be used as the user interface or as a subroutine in an application. User interface elements such as graphs are easily accessed, as drag-and drop units in Lab VIEW.

Lab VIEW programs are called virtual instruments (VIs), because their appearance and operation imitate physical instruments like oscilloscopes. Lab VIEW is designed to facilitate data collection and analysis, as well as offers numerous display options. With data collection, analysis and display combined in a flexible programming environment, the desktop computer functions as a dedicated measurement device. Lab VIEWcontains a comprehensive set of VIs and functions for acquiring, analyzing, displaying, and storing data, as well as tools to troubleshoot code.

Software Environment

There are three steps to create our application in the software environment:
- Design a user interface
- Draw our graphical code
- Run our program

A virtual instrument (VI) has three main components—the front panel, the block diagram and the icon/connector pane. In order to use a VI as a subVI in the block diagram of another VI, it is essential that it contains an icon and a connector. The two Lab VIEW windows are the front panel (containing controls and indicators) and block diagram (containing terminals, connections and graphical code). The front panel is the user interface of the virtual instrument. The code is built using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code. In Lab VIEW, you build a user interface or front panel with controls and indicators. Controls are knobs, push buttons, dials and other input devices. Indicators are graphs, LEDs and other displays. After you build the user interface, you can add code using VIs and structures to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

THE FRONT PANEL

When a blank VI is opened, an untitled front panel window appears. This window displays the front panel and is one of the two Lab VIEW windows used to build a VI. The front panel is the window through which the user interacts with the program. The input data to the executingprogram is fed through the front panel and the output can also be viewed on the front panel, thus making it indispensable. One of the most powerful features that Lab VIEW offers engineers and scientists is its graphical programming environment to design custom virtual instruments by creating a graphical user interface on the computer screen to

- Operate the instrumentation program
- Control selected hardware
- Analyze acquired data
- Display results

The front panel can include knobs, push buttons, graphs and various other controls (which are user inputs) and indicators (which are program outputs). Controls are inputs used to simulate instrument input devices and supply data to the block diagram of the VI, and indicators are outputs displays used to simulate instrument output devices and display data the block diagram acquires or generates. The front panel is customized toemulate control panels of traditional instruments, create custom test panels, or visually represent the control and operation of processes.

BLOCK DIAGRAM

The block diagram window holds the graphical source code of a Lab View's block diagram corresponds to the lines of text found in a more conventional language like C or BASIC – it is the actual executable code. The block diagram can be constructed with the basic blocks such as: terminals, nodes, and wires.The block diagram is the actual executable program as shown in Figure.

The components of a block diagram are lower-level VIs, built-in functions, constants and program execution control structures. Wires have to be drawn to connect the corresponding objects together to indicate the flow of data between each of them. Front panel objects have analogous terminals on the block diagram so that data can pass easily from the user to the program and back to the user. Use Express VIs, standard VIs and functions on the block diagram to create our measurement code. Block diagram objects include the terminals, subVIs, functions, constants, structures and wires.

Icon/Connector Pane

To use a VI as a subVI, it must have an icon and a connector pane. Every VI displays an icon in the upper-right corner of the front panel and block diagram windows. An icon is a graphical representation of a VI. The icon can contain both text and images. To use a VI as a subVI, you need to build a connector pane. The connector pane is a set of terminals that correspond to the controls and indicators of that VI.

The various windows associated with Lab VIEW is summarized below:

- FRONT PANEL TOOLBAR
- BLOCK DIAGRAM TOOLBAR
- PALETTES
- Tools Palette
- Front Panel—Controls Palette
- Block Diagram—Functions Palette
- PROPERTY DIALOG BOXES
- FRONT PANEL CONTROLS AND INDICATORS

EXECUTING VIs

A Lab VIEW program is executed by pressing the arrow or the Run button located in the palette along the top of the window. While the VI is executing, the Run button changes to a black color as depicted in Figure. Note that not all of the items in the palette are displayed during execution of a VI. As you proceed to the right along the palette, you will find the Continuous Run, Stop, and Pause buttons.

If the Run button appears as a broken arrow, this indicates that the Lab VIEW program or VI cannot compile because of programming errors. When all of the errors are fixed, the broken Run button will be substituted by the regular Run button. Lab VIEW has successfully compiled the diagram. While editing or creating a VI, you may notice that the palette displays the broken Run button. If you continue to see this after editing is completed, press the button to determine the cause of the errors. An Error List window will appear displaying all of the errors that must be fixed before the VI can compile.

LAB VIEW FILE EXTENSIONS

Lab VIEW programs utilize the .vi extension. However, multiple VIs can be saved into library format with the .llb extension. Libraries are useful for grouping related VIs for file management. When loading a particular VI that makes calls to other VIs, the system is able to find them quickly. Using a library has benefits over simply usinga directory to group VIs. It saves disk space by compressing VIs, and facilitates themovement of VIs between directories or computers. When saving single VIs, remember to add the .vi extension. If you need to create a

library for a VI and its subVIs, you will need to create a source distribution using the LabVIEW Project.

Advantages of Lab VIEW

The following are the advantages of Lab VIEW:

● **Graphical user interface:** Design professionals use the drag-and-drop user interface library By interactively customizing the hundreds of built-in user objects on the controls palette.

● **Drag-and-drop built-in functions:** Thousands of built-in functions and IP including analysis and I/O, from the functions palette to create applications easily.

● **Modular design and hierarchical design:** Run modular Lab VIEW VIs by themselves or as SubVIs and easily scale and modularize programs depending on the application.

● **Multiple high level development tools:** Develop faster with application specific development tools, including the Lab VIEWState chart Module, Lab VIEW Control Design and Simulation
Module and Lab VIEW FPGA Module.

● Professional **Development Tools:**Manage large, professional applications and tightly integrated project management tools; integrated graphical debugging tools; and standardized source code control integration.

DATA Types:

Data types indicate what objects, inputs and outputs you can wire together. For example, if a switch has a green border, you can wire a switch to any input with a green label on an Express VI. If a knob has an orange border, you can wire a knob to any input with an orange label. However, you cannot wire an orange knob to an input with a green label. Notice the wires are the same color as the terminal. The dynamic data type stores the information generated or acquired by an Express VI. The dynamic data type appears as a dark blue terminal. Most Express VIs accept and/or return the dynamic data type. You can wire the dynamic data type to any indicator or input that accepts numeric, waveform or Boolean data. Wire the dynamic data type to an indicator that can best present the data. Indicators include a graph, chart or numeric indicator.

Advantages of Virtual Instrumentation techniques:

1) **Simplification**:
   Suppose you want to check out an electronic circuit,by noting its response to a test signal.You will need a Signal Generator, an oscilloscope and the necessary accessories.And take the readings.With a computer based testing system,one can have the complete system integrated to a single application and the user can do the required testing

operating from a easy to use graphical user interface(GUI).That makes the things easy to work,simplifying the process.

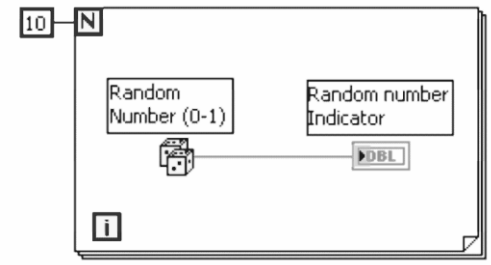
2) **Customization:**With the conventional test devices like function generators or oscilloscopes the scope for customization is very limited.Since the software forms the element of the instrumentation in the virtual instrumentation,it offers the scope of customizing the testing applications to the user's needs.This is a very important advantage since,many of the times there exists the need for custom made testing in the science and engineering.
3) **Cost Advantage:**There is definitely a cost advantage compared to the traditional instrumentation,because the applications can be easily customized and the programming can be carried out so that the single application can replace multiple devices (which may be required otherwise).
4) **Computer Based**:Virtual instrumentation takes advantage of the ever evolving computer technology - the increasing processing speeds,the easy device interface options,larger memory and so on.Hence enhancing the application performance and development process.
5) **Portability:**Portability is an essential feature some times.The virtual instrumentation based applications are powerful and compact so that they can be easily set up and easy to carry.For example,one needs to carry out an engine testing by simulating certain conditions,the complete signal set to and fro from the engine under test can be interface to something like laptop through an USB interface and the testing can be controlled through an application program executing on the computer.so carrying a simple laptop and the USB interface is sufficient.This is a great feature,when you need a good number of systems interfaced during,say,some system testing applications.

Concept of For loops:

A For Loop executes a subdiagram, a set number of times. Figure 4.1(a) shows a For Loop in LabVIEW and Figure 4.1(b) shows the flow chart equivalent of the For Loop functionality. The For Loop is located on the Functions>>Programming>>Structures Palette. Select the For Loop from the palette and use the cursor to drag a selection rectangle to create a new For Loop or around the section of the block diagram you want to repeat. You also can place a While Loop on the block diagram, right-click the border of the While Loop, and select Replace with For Loop from the shortcut menu to change a While Loop to a For Loops.

The value in the count terminal 'N' (an input terminal) indicates how many times to repeat the subdiagram. Set the count explicitly by wiring a value from outside the loop to the left or top side of the count terminal, or set the count implicitly with auto-indexing. Auto-indexing is explained in chapter 5. A VI will not run if it contains a For Loop that

does not have a numeric value wired to the count terminal. The iteration terminal 'i' (an output terminal) contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0. Figure 4.2 shows a simple For Loop which generates 10 random numbers and displays in the Random Number Indicator.



Both the count and iteration terminals are 32-bit signed integers. If you wire a floating point number to the count terminal, LabVIEW rounds it and coerces it to within range. If you wire 0 or a negative number to the count terminal, the loop does not execute and the outputs contain the default data for that data type. A For Loop can only execute an integer a number of times.

While loop:

A While Loop executes a subdiagram until a condition is met. The While Loop is similar to a Do Loop or a Repeat-Until Loop in text-based programming languages. Figure 4.4(a) shows a While Loop in LabVIEW and 4.4(b) is the flow chart equivalent of the While Loop. The While Loop always executes at least once. The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing the subdiagram, only if the expected value at the conditional terminal exists. In LabVIEW, the WHILE Loop is located on the Functions>>Programming>>Structures palette. You also can place a For Loop on the block diagram, right-click the border of the For Loop, and select Replace with While Loop from the shortcut menu to change a For Loop to a While Loop. The While Loop contains two terminals, namely Conditional Terminal and Iteration Terminal.
The Conditional Terminal is used to control the execution of the loop, whereas the Iteration Terminal is used to know the number of completed iterations.

Conventional programming languages support two types of WHILE constructs as illustrated in Figure 4.5. These are called pre- and post-test modes. In the pre-test mode the condition is tested prior to the execution of every iteration and if the result is false, then the execution of the loop is aborted. In the post-test mode the test is carried out only at the end of the loop. Functionally, the major difference is that under the post-test mode even if the condition is false at the first execution or first iteration, the loop will be

executed at least once, since the test is only performed at the end of the loop. LabVIEW supports only the post-test form of the While construct.

ARRAYS IN LabVIEW:

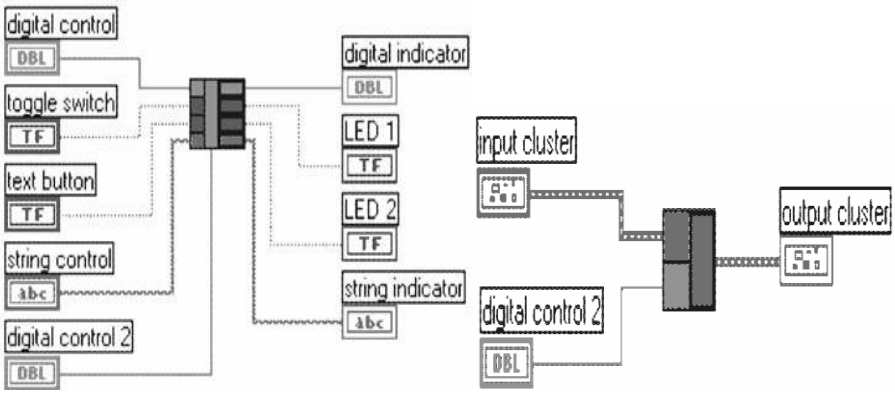
In LabVIEW arrays group data elements of the same type. They are analogous to arrays in traditional languages. An array consists of elements and dimensions. Elements are the data that make up an array. A dimension is the length, height or depth of an array. An array can have one or more dimensions and as many as (231) – 1 elements per dimension, memory permitting.

You can build arrays of numeric, Boolean, path, string and cluster data types. You cannot create arrays of arrays. However, you can use a multidimensional array or an array of clusters where each cluster contains one or more arrays. Also, you cannot create an array of subpanel controls, tab controls, .NET controls, ActiveX controls, charts or multiplot XY graphs. Array elements are ordered. To locate a particular element in an array requires one index per dimension. For example, if you want to locate a particular element in a two-dimensional array, you need both row index and column index. In LabVIEW, indexes let us navigate through an array and retrieve elements, rows and columns from an array on the block diagram. The index is zero-based, which means it is in the range 0 to n – 1, where n is the number of elements in the array with the first element at index 0 and the last one at index (n – 1). For example, if you create an array of 10 elements, the index ranges from 0 to 9. The fourth element has an index of 3.

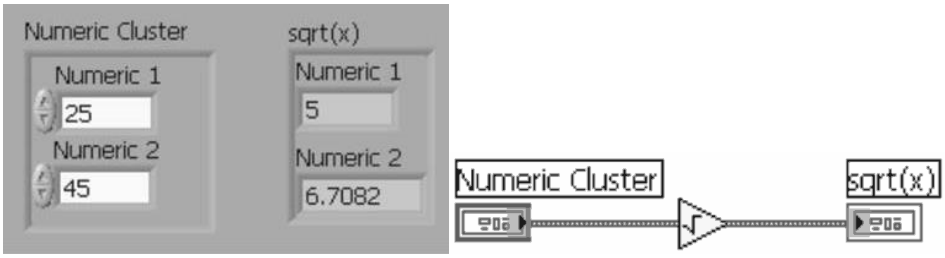
CLUSTERS:

Clusters group data elements of mixed types. An example of a cluster is the LabVIEW error cluster,which combines a Boolean value, a numeric value and a string. A cluster is similar to a record or astruct in text-based programming languages. Figure show the error cluster control and thecorresponding terminal created in the block diagram. This cluster consists of a Boolean control(status), a numeric control (code) and a string control (source).

Bundling several data elements into clusters eliminates wire clutter on the block diagram. Italso reduces the number of connector pane terminals that subVIs need by passing several values toone terminal. The connector pane has, at most, 28 terminals. If your front panel contains more than28 controls and indicators that you want to pass to another VI, group some of them into a clusterand assign the cluster to a terminal on the connector pane. In Figure individual values arepassed to the connector pane terminals. In Figure shows controls and indicators are grouped intoclusters and connected to one control panel terminal each.



Most clusters on the block diagram have a pink wire pattern and data type terminal. Clustersof numeric values, sometimes referred to as points, have a brown wire pattern and data type terminalas shown in Figure. You can wire brown numeric clusters to numeric functions, such as *Add* or*Square Root*, to perform the same operation simultaneously on all elements of the cluster.



GRAPHS:

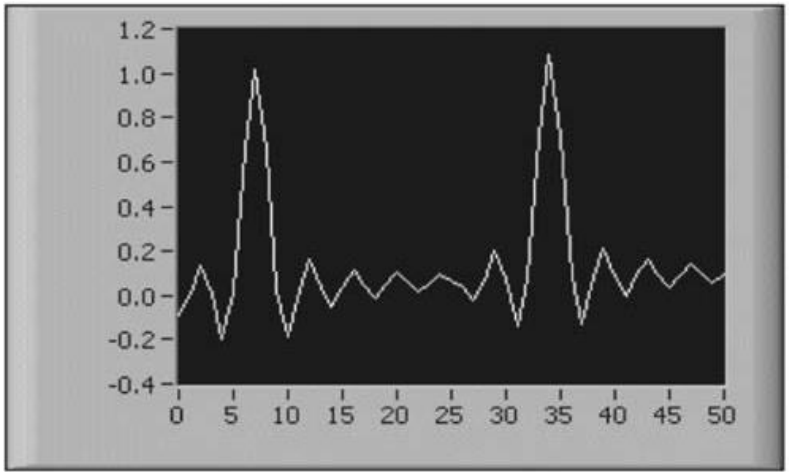
LabVIEW includes the following types of graphs and charts.

- **Waveform graphs and charts:** Display data typically acquired at a constant rate.
- **XY Graphs:** Display data acquired at a non-constant rate and data for multivalued functions.
- **Intensity graphs and charts:**Display 3D data on a 2D plot by using colour to displaythe valuesof the third dimension.
- **Digital waveform graphs:** Display data as pulses or groups of digital lines.

- **Windows 3D Graphs:** Display 3D data on a 3D plot in an ActiveX object on the front Panel

WAVEFORM GRAPHS:

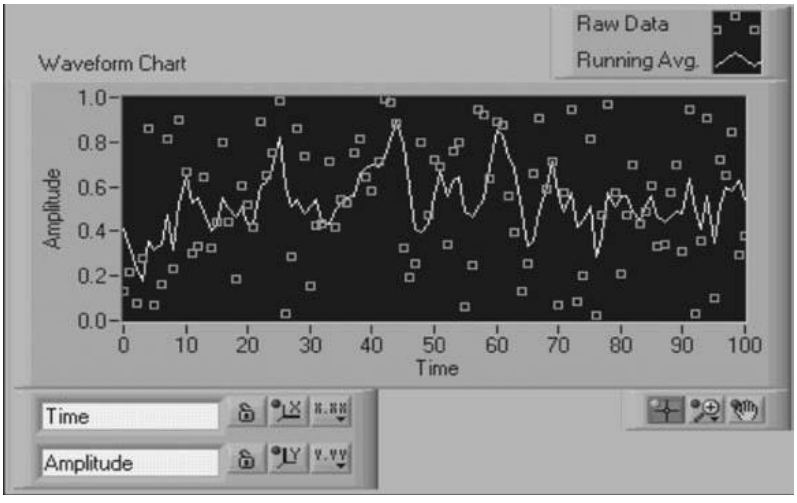
LabVIEW includes the waveform graph and chart to display data typically acquired at a constant rate. The waveform graph displays one or more plots of evenly sampled measurements. The waveform graph plots only single-valued functions, as in y = f (x), with points evenly distributed along the x-axis, such as acquired time-varying waveforms. Figure 7.1 shows an example of a waveform graph. The waveform graph can display plots containing any number of points. The graph also accepts several data types, which minimizes the extent to which you must manipulate data before you display it.



Waveform Graph

WAVEFORM CHARTS:

The waveform chart is a special type of numeric indicator that displays one or more plots of data typically acquired at a constant rate. Waveform charts can display single or multiple plots. Figure 7.3 shows the elements of a multiplot waveform chart. Two plots are displayed: Raw Data and Running Avg. The waveform chart maintains a history of data or buffer from previous updates.



Waveform Chart

STRUCTURE:

Structures are graphical representations of the loops and case statements of text-based programming languages. There are cases when a decision must be made in a program. In text-based programs, this can be accomplished with statements like if-else, case and so on. LabVIEW includes many different ways of making decisions. The simplest of these methods is the select function located in the functions palette. This function selects between two values dependent on a Boolean input. Use structures on the block diagram to repeat blocks of code and to execute code conditionally or in a specific order. Like other nodes, structures have terminals that connect them to other block diagram nodes, execute automatically when input data are available, and supply data to output wires when execution is complete. Each structure has a distinctive, resizable border to enclose the section of the block diagram that executes according to the rules of the structure. The section of the block diagram inside the structure border is called a sub diagram. The terminals that feed data into and out of structures are called tunnels. A tunnel is connection point on a structure border.

CASE STRUCTURES:

A case structure executes one sub diagram depending on the input value passed to the structure.

Complete the following steps to create a Case structure.

Step 1: Place a Case structure on the block diagram.

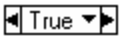
Step 2: Wire an input value to the selector terminal to determine which case executes. You must wire an integer, Boolean value, string, or enumerated type value to the selector terminal. You also can wire an error cluster to the selector terminal to handle errors.

Step 3: Place objects inside the Case structure to create subdiagrams that the Case structure can

execute. If necessary, add or duplicate subdiagrams. If the data type of the selector terminal isBoolean, the structure has a TRUE case and a FALSE case. If the selector terminal is an integer,string, or enumerated type value, the structure can have any number of cases.

Step 4: For each case, use the Labeling tool to enter a single value or lists and ranges of valuesin the case selector label at the top of the Case structure. For lists, use commas to separate values.For numeric ranges, specify a range as 10.20, meaning all numbers from 10 to 20 inclusively. Ifnecessary (optional), specify a default case.

A Case structure, shown, has two or more subdiagrams, or cases.



Only one subdiagram is visible at a time, and the structure executes only onecase at a time. An input value determines which subdiagram executes. The Casestructure is similar to switch statements or if...then...else statements in text-basedprogramming languages. The case selector label at the top of the Case structure,shown, contains the name of the selector value that corresponds to the case in thecenter and decrement and increment arrows on each side.

Click the decrement and increment arrows to scroll through the available cases.

You also can click the down arrow next to the case name and select a case from thepull-down menu.Wire an input value, or selector, to the selector terminal, shown, to determine which caseexecutes.You must wire an integer, Boolean value, string, or enumerated type value to the selectorterminal. You can position the selector terminal anywhere on the left border of the Case structure.

If the data type of the selector terminal is Boolean, the structure has a TRUE case and a FALSE case. If the selector terminal is an integer, string, or enumerated type value, the structure can have any number of cases.

Specify a default case for the Case structure to handle out-of-range values. Otherwise, you must explicitly list every possible input value. For example, if the selector is an integer and you specify cases for 1, 2 and 3, you must specify a default case to execute if the input value is 4 or any other unspecified integer value.

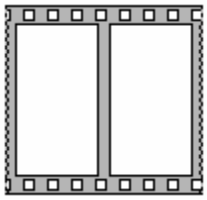
SEQUENCE STRUCTURES:

A sequence structure contains one or more subdiagrams, or frames, that execute in sequential order. Within each frame of a sequence structure, as in the rest of the block diagram, data dependency determines the execution order of nodes. Sequence structures are not used commonly in LabVIEW. Use the sequence structures to control the execution order when natural data dependency does not exist and flow-through parameters are not available. There are two types of sequence structures— the Flat Sequence structure and the Stacked Sequence structure.

i) Flat Sequence Structure

The Flat Sequence structure, shown as follows, displays all the frames at once and executes the frames from left to right and when all data values wired to a frame are available, until the last frame executes. The data values leave each frame as the frame finishes executing.

Use the Flat Sequence structure to avoid using sequence locals and to better document the block diagram. When you add or delete frames in a Flat Sequence structure, the structure resizes automatically.

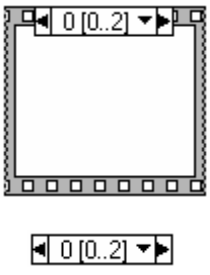


To convert a Flat Sequence structure to a Stacked Sequence structure, right-click the Flat Sequence structure and select Replace with Stacked Sequence from the shortcut menu. If youchange a Flat Sequence to a Stacked Sequence and then back to a Flat Sequence, LabVIEW movesall input terminals to the first frame of the sequence. The final Flat Sequence should operate thesame as the Stacked Sequence. After you change the Stacked Sequence to a Flat Sequence with allinput terminals on the first frame, you can move wires to where they were located in the original Flat Sequence.

ii) Stacked Sequence Structure

The Stacked Sequence structure, shown as follows, stacks each frame so you see only one frame at a time and executes frame 0, then frame 1, and so on until the last frame executes.

The Stacked Sequence structure returns data only after the last frame executes. Use the Stacked Sequence structure if you want to conserve space on the block diagram. To convert a Stacked Sequence structure to a FlatSequence structure, right-click the Stacked Sequence structure and select Replace»Replace with Flat Sequence from the shortcut menu. The sequence selector identifier, shown as follows, at the top of the Stacked Sequence structure contains the current frame number and range of frames.



Use the sequence selector identifier to navigate through the available frames and rearrange frames. The frame label in a Stacked Sequence structure is similar to the case selector label of

theCase structure. The frame label contains the frame number in the center and decrement and increment arrows on each side. Click the decrement and increment arrows to scroll through the available frames. You also can click the down arrow next to the frame number and select a frame from the pull-down menu. Right-click the border of a frame, selectsMake This Frame, and select a frame number from the shortcut menu to rearrange the order of a Stacked Sequence structure. Unlike the case selector label, you cannot enter values in the frame label. When you add, delete, or rearrange frames in a Stacked Sequence structure, LabVIEW automatically adjusts the numbers in the frame labels.

To pass data from one frame to any subsequent frame of a Stacked Sequence structure, use asequence local terminal shown.

An outward-pointing arrow appears in the sequence local terminal of the frame that contains the data source. The terminal in subsequent frames contains an inward-pointing arrow, indicatingthat the terminal is a data source for that frame. You cannot use the sequence local terminal inframes that precede the first frame where you wired the sequence local.

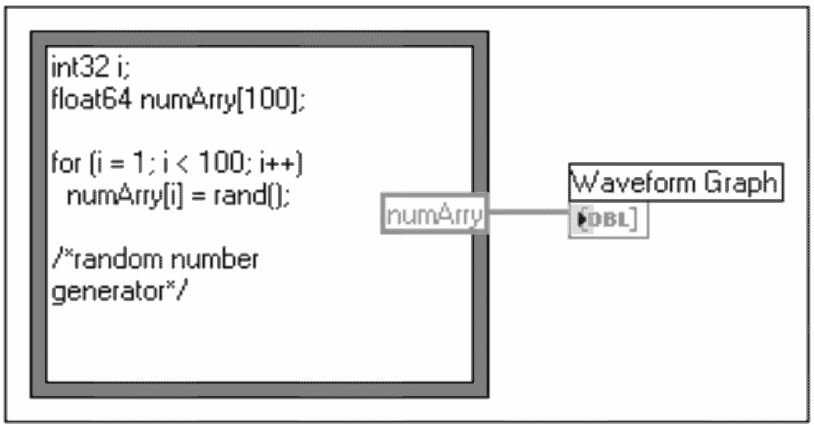
FORMULA NODES:

The Formula Node is a convenient text-based node you can use to perform mathematical operations on the block diagram. You do not have to access any external code or applications, and you do not have to wire low-level arithmetic functions to create equations. In addition to text-based equation expressions, the Formula Node can accept text-based versions of If statements, While loops, For loops, and Do loops which are familiar to C programmers. These programming elements are similar to what you find in C programming but are not identical.

Formula Nodes are useful for equations that have many variables or are otherwise complicated and for using existing text-based code. You can copy and paste the existing text-based code into a Formula Node rather than recreating it graphically. Formula Nodes use type checking to make sure that array indexes are numeric data and that operands to the bit operations are integer data. Formula Nodes also check to make sure array indexes are in range. For arrays, an out-of-range value defaults to zero, and an out-of-range assignment defaults to nop to indicate no operation occurs. Formula Nodes also perform automatic type conversion.

i) Using the Formula Node

The Formula Node, shown in Figure 8.6, is a resizable box similar to the For Loop, While Loop, Case structure, Stacked Sequence structure and Flat Sequence structure.



However, instead of containing a sub diagram, the Formula Node contains one or more C-like statements delimited by semicolons, as in the following example. As with C, add comments by enclosing them inside a slash/asterisk pair (/*comment*/) or by preceding them with two slashes (//comment).

When you work with variables, remember the following points:

● There is no limit to the number of variables or equations in a Formula Node.

● No two inputs and no two outputs can have the same name, but an output can have the same name as an input.

● Declare an input variable by right-clicking the Formula Node border and selecting Add Input from the shortcut menu. You cannot declare input variables inside the Formula Node.

● Declare an output variable by right-clicking the Formula Node border and selecting Add Output from the shortcut menu. The output variable name must match either an inputvariable name or the name of a variable you declare inside the Formula Node.

● You can change whether a variable is an input or an output by right-clicking it and selecting Change to Input or Change to Output from the shortcut menu.

● You can declare and use a variable inside the Formula Node without relating it to an input or output wire. Structures **173**

● You must wire all input terminals.

● Variables can be floating-point numeric scalars whose precision depends on the configuration of your computer. You also can use integers and arrays of numeric values for variables.

● Variables cannot have units.

Complete the following steps to change a Formula Node terminal from an output to an input or vice versa.

Step 1: Right-click the output or input terminal.

Step 2: Select Change to Input or Change to Output from the shortcut menu.

Complete the following steps to remove a terminal from a Formula Node.

Step 1: Right-click the input or output terminal you want to remove.

Step 2: Select Remove from the shortcut menu.

You also can select the terminal and press the <Delete> or <Backspace> key.

Input arrays and input-output arrays take their type from the array to which they are wired. These arrays do not require you to declare them inside the Formula Node. However, you must declare local arrays and output arrays in the Formula Node. Arrays are zero-based, as they are in C. Unlike C, LabVIEW treats an assignment to an array element that is out of range as a nonoperation, and no assignment occurs. Also unlike C, if you make a reference to an array element that is out of range, LabVIEW returns a value of zero. You must declare array outputs in the Formula Node unless they correspond to an array input, in which case the two terminals mustshare a name.

ii) Creating Formula Nodes

Complete the following steps to create a *Formula Node*.

Step 1: Place a *Formula Node* on the block diagram.

Step 2: Review the available functions and operators you can use.

Step 3: Use the labeling tool or the operating tool to enter the equations you want to calculateinside the *Formula Node*. Each assignment must have only a single variable on the left side of theassignment (=). Each assignment must end with a semicolon (;). Confirm that you are using thecorrect *Formula Node* syntax.

Step 4: If a syntax error occurs, click the broken *Run* button to display the *Error list* window.LabVIEW marks the syntax error with a # symbol.

Step 5: Create an input terminal for each input variable by right-clicking the *Formula Node*border and selecting *Add Input* from the shortcut menu. Type the variable name in the terminal thatappears. You can edit the variable name at any time using the labeling tool or the Operating tool,except when the VI is running.

Step 6: Variable terminals are case sensitive. There is no limit to the number of terminals orequations in a Formula Node. You can change a terminal type or remove a terminal.

Step 7: Create an output terminal for each output variable by right-clicking the Formula Nodeborder and selecting Add Output from the shortcut menu. Type the variable name in the terminalthat appears. You can edit the variable name at any time using the labeling tool or the operatingtool, except when the VI is running. Output variables have thicker borders than input variables.

Step 8: (Optional) The default data type for output terminals is double-precision, floating-point.To change the data type, create an input terminal with exactly the same name as the output terminaland wire a data type to that input terminal. Doing so also provides a default value for the terminal.You also can use the Formula Node syntax to define the variable inside the Formula Node. Forexample, int32 y; changes the data type of the output terminal y to 32-bit integer.

Step 9: Wire the input and output terminals of the Formula Node to their corresponding terminalson the block diagram. All input terminals must be wired. Output terminals do not have to be wired.

iii) Formula Node Syntax

The Formula Node syntax is similar to the syntax used in text-based programming languages.Remember to end assignments with a semicolon (;) as in C. Use scope rules to declare variables inFormula Nodes. The Formula Node syntax is summarized below using Backus-Naur Form (BNFnotation). The summary includes non-terminal symbols: compound-statement, identifier,conditional-expression, number, array-size, floating-point-type, integer-type, left-hand-side,assignment-operator, and function. Symbols in red bold monospace are terminal symbols givenexactly as they should be reproduced. The symbol # denotes any number of the term following it.

Use the break keyword to exit the nearest Do, For, or While Loop or Switch statement in theFormula Node. Use the continue keyword to transfer control to the next iteration of the nearest Do,For, or While Loop in the Formula Node. The conditional statement uses the same syntax as the ifand else statements in C. The do statement, the For statement, the switch statement and the whilestatement all use the same syntax as the respective statements in C.

iv) Scope Rules for Declaring Variables in Formula Nodes

Formula Nodes use the same scope rules for declaring variables as C.

- All variables you declare in bracketed blocks are only accessible within the bracketedblock.
- All input terminals are considered variables at the outermost block (not enclosed inbrackets), and cannot be declared again in the outermost block.
- LabVIEW tries to match variables you declare at the outermost block (not enclosed inbrackets) to output terminals with the same name.
- You can declare output terminals that have no corresponding input terminal and are notdeclared at the outermost block.Use the correct Formula Node syntax when declaring variables.