

# VIRTUAL INSTRUMENTATION

Janani R

Assistant Professor  
Electronics and Instrumentation Engineering  
Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya University



## 1 Unit-1 Introduction to LabVIEW

- Objective
- Pre-requisite
- Graphical System Design Model
- Hardware, Software, Design in Virtual Instrumentation
- Software Environment, Front Panel, Block Diagram
- Summary
- Examples/Work-Out
- Assignment Questions

# Table of Contents

- 1 Introduction to Graphical System Design (GSD) model
- 2 Virtual Instrument and Traditional Instrument
- 3 Hardware and Software in Virtual Instrumentation
- 4 Design and Virtual Instrumentation Advantages
- 5 Comparison of Graphical Programming with Textual Programming
- 6 Software Environment : Creating and Saving a VI
- 7 Front panel Toolbar, Block diagram Toolbar, Palettes, Controls and Indicators.
- 8 Block diagram- Terminals, nodes, Functions, wires, Data types and Data flow program.

# Introduction to LabVIEW

- **Objective :** The main aim of this chapter to know graphical system design model, how GSD is differ from Textual Programming, how to create, save a LabVIEW program.
- **Pre-requisite :**
  - ▶ Basic Engineering Mathematics
  - ▶ Basic Programming Language
- **Introduction**
  - ▶ The term scientific computing has been used for many years define the use of computers for solving problems related to science and engineering, usually involving experimental or applied research, modeling and simulation.
  - ▶ In simple it refers to the use of computers in solving scientific problems.
  - ▶ Scientific computing applications usually follow a three-step process : data acquisition, data analysis and data visualisation
  - ▶ This three step approach has been used by National Instruments and developed Virtual Instrumentation Model, which has been then expanded into a more comprehensive model known as Graphical System Design shown in Fig.

# Introduction

- Virtual Instrumentation Model



FIGURE – Virtual Instrumentation Model

- Graphical System Design Model



FIGURE – Graphical System Design Model

# Graphical System Design Model

- What is GSD Model

- ▶ In this Graphical System Design model, the focus is to accelerate the research and development cycle, delivering mathematical models to embedded real time computers faster and easier.
- ▶ This design flow acceleration is achieved by using NI LABVIEW software and its G programming language as a common system level design tool for all the different phases in the design -to-deployment flow.
- ▶ In reality the virtual instrumentation model is applied in each of the three phases of the graphical system design model as shown in Fig. because data acquisition, analysis and presentation functions are used in the design, prototyping and deployment phases.

# Graphical System Design Model

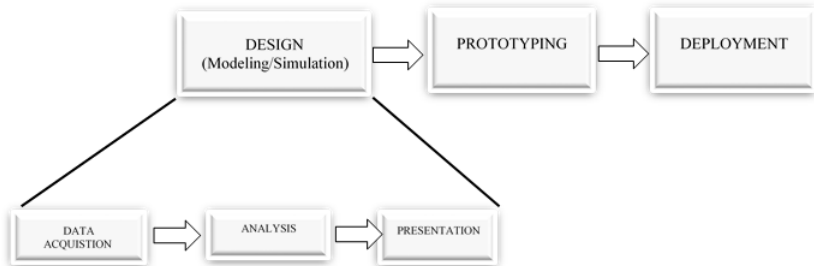


FIGURE – Graphical System Design Model and Virtual Instrumentation

# Graphical System Design Model

- Design (Model)

- ▶ In the design phase as shown in fig. the researcher develops a mathematical model of the system, including sensors, actuators, plants and controllers, and simulates them under a variety of initial conditions and constraints.
- ▶ The researcher uses different numerical methods with the objective of validating the performance of the model and optimizing it.
  - In this phase, researchers can acquire reference data from files or databases and incorporate it into the model.
  - A "Virtual plant/process" is created, which can be used later for hardware-in-the-loop(HIL) tests.
  - Results from the simulation process are saved for post analysis and visualization and can be used to introduce changes into the models.
  - This is usually a software-centric process with a strong focus on numerical methods/analysis and mathematics



# Graphical System Design Model

- Design (Model)

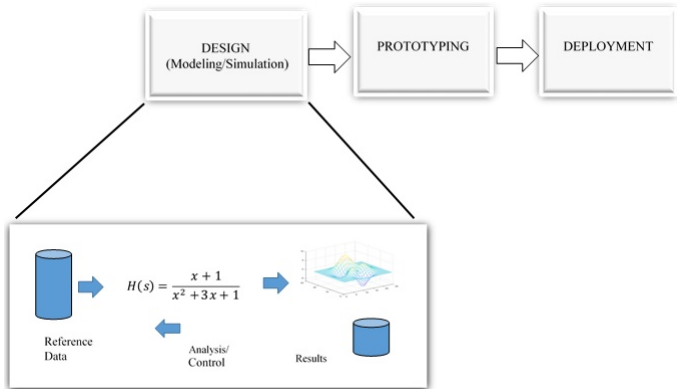


FIGURE – The design phase of the graphical system design model

# Graphical System Design Model

- Prototype Lab :

- ▶ If experimental validation of the model is required, researchers develop and test a prototype in the laboratory. Signal processing and analysis as well as visualization can be implemented online while data is being measured and acquired, or while the process is being controlled.
- ▶ This process as shown in Figure is usually more software/hardware-centric because sensors, actuators, data acquisition devices, controllers, and the controlled/analyzed plant itself are all key elements in the experimental setup.

# Graphical System Design Model

- Prototype (Lab)

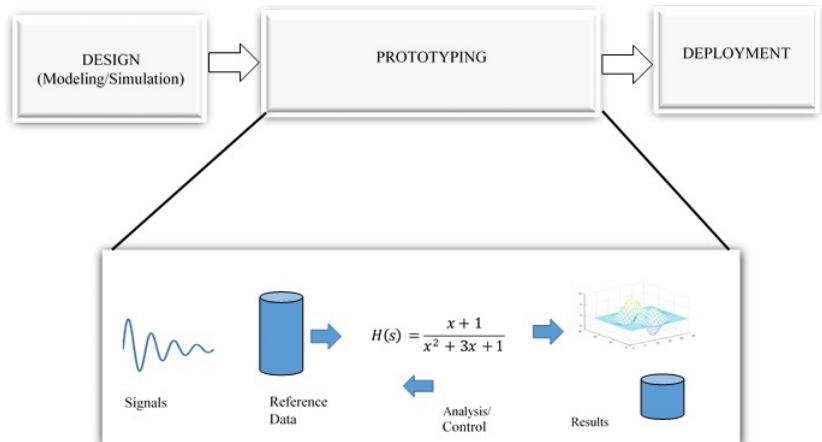


FIGURE – The prototyping phase of the graphical system design model

# Graphical System Design Model

- **Deployment - Field :**

- ▶ Finally, the model (controller, analyzer or both) is deployed in the field or lab using either a PC (desktop, server or industrial) or PXI, or it can be downloaded to a dedicated embedded controller such as CompactRIO, which usually operates in stand-alone mode and in real-time (deterministic) mode as shown in Figure
- ▶ When using the LabVIEW Real-Time Module, symmetric multiprocessing (SMP) techniques can be easily applied.
- ▶ In many cases, the system is deployed as a headless system (no monitors or interfaces to the user), executing the analysis/control algorithms in real time as a dedicated device. If on-the-field graphical user interfaces (GUIs) or operator interfaces (OIs) are needed, the LabVIEW TouchPanel Module and industrial-grade monitors with touch-sensitive screens can be used locally

# Graphical System Design Model

- Deployment(Field)

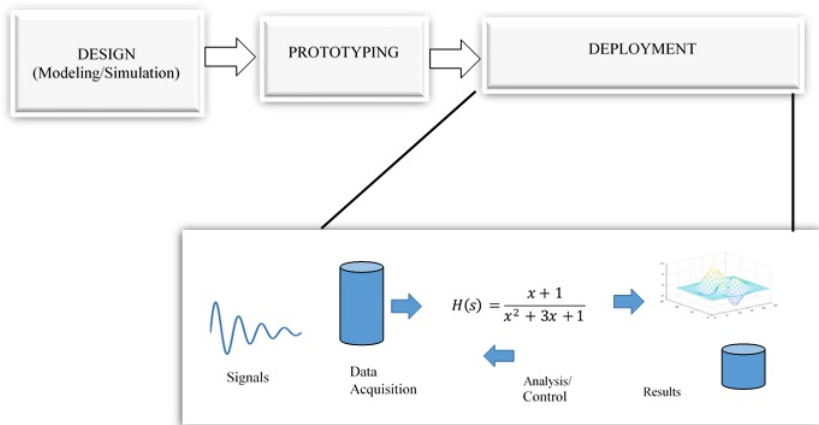


FIGURE – The deployment phase of the graphical system design model

# Graphical System Design Model

- Design Flow with GSD

Typical Embedded System software and hardware design flows is shown below.



FIGURE – Typical Embedded System Software and Hardware Design Flow

# Graphical System Design Model

- Design Flow with GSD

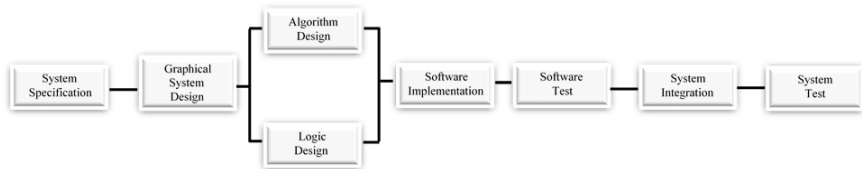


FIGURE – Stream-lines development flow with graphical system design

# Virtual Instrument and Traditional Instrument

- **Virtual Instrument**

- ▶ 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 and to convert the sensed information into electrical signals.
- ▶ The term “instrument” may also cover a physical or software device that performs an analysis on data acquired from another instrument and then outputs the processed data to display or recording means.
- ▶ This second category of instruments includes oscilloscopes, spectrum analyzers and digital millimeters.
- ▶ The types of source data collected and analyzed by instruments vary widely, including both physical parameters like temperature, pressure, distance, and also electrical parameters like voltage



# Virtual Instrument and Traditional Instruments

- **Virtual Instrument**

- ▶ 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.
- ▶ With virtual instrumentation, engineers and scientists reduce development time, design higher quality products, and lower their design costs.
- ▶ Virtual instrumentation is necessary because it is flexible. It delivers instrumentation with the rapid adaptability required for today's concept, product and process design, development and delivery.
- ▶ Only with virtual instrumentation, engineers and scientists can create the user-defined instruments required to keep up with the world's demands.
- ▶ Virtual instruments are defined by the user while traditional instruments have fixed vendor-defined functionality.

# Virtual Instrument and Traditional Instruments

- **Virtual Instrument**

- ▶ Every virtual instrument consists of two parts—software and hardware.
- ▶ A virtual instrument typically has a sticker price comparable to and many times less than a similar traditional instrument for the current measurement task.
- ▶ A traditional instrument provides them with all software and measurement circuitry packaged into a product with a finite list of fixed-functionality using the instrument front panel.
- ▶ A virtual instrument provides all the software and hardware needed to accomplish the measurement or control task.
- ▶ In addition, with a virtual instrument, engineers and scientists can customize the acquisition, analysis, storage, sharing and presentation functionality using productive, powerful software.

# Virtual Instruments and Traditional Instruments

- Virtual Instrument

- ▶ Without the displays, knobs and switches of a 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.
- ▶ In fact, a multiple-purpose virtual instrument can be made by using a single data acquisition board or card.
- ▶ The primary benefits of apply data acquisition technology to configure virtual instrumentation include costs, size, and flexibility and ease of programming

# Virtual Instruments and Traditional Instruments

- Traditional Instruments

- ▶ Traditional instruments and software-based virtual instruments largely share the same architectural components, but radically different philosophies as shown in Figure.
- ▶ Conventional instruments as compared to a virtual instrumentation can be very large and cumbersome.
- ▶ They also require a lot of power, and often have excessive amounts of features that are rarely, if ever used.
- ▶ Most conventional instruments do not have any computational power as compared to a virtual instrument.
- ▶ Virtual instruments are compatible with traditional instruments almost without exception.

# Virtual Instruments and Traditional Instruments

- Traditional Instruments

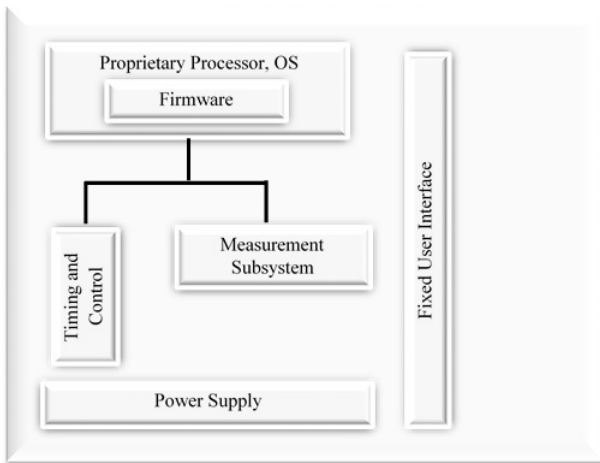


FIGURE – Traditional Instruments Block Diagram

# Virtual Instruments and Traditional Instruments

- Virtual Instruments

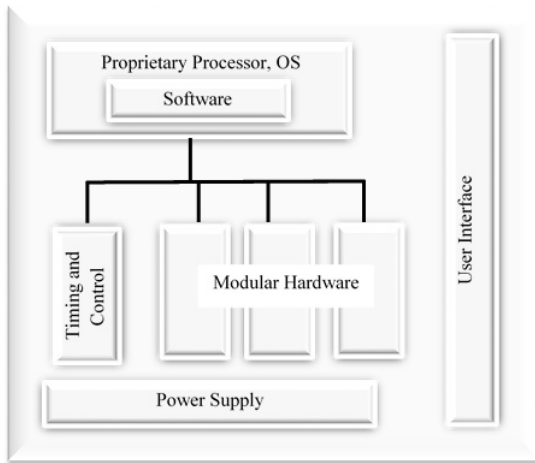


FIGURE – Software based Virtual Instruments

# Traditional Instruments and Virtual Instruments

- The general architecture of stand-alone instruments is very similar to that of a PC-based virtual instrument.
- Both require one or more microprocessors, communication ports (for example, serial and GPIB), and display capabilities, as well as data acquisition modules.
- A traditional instrument might contain an integrated circuit to perform a particular set of data processing functions; in a virtual instrument, these functions would be performed by software running on the PC processor.
- By employing virtual instrumentation solutions, we can lower capital costs, system development costs, and system maintenance costs, while improving time to market and the quality of our own products.

# Traditional Instruments and Virtual Instruments

- Virtual instruments take advantage of PC performance increase by analyzing measurements and solving new application challenges with each new-generation PC processor, hard drive, display and I/O bus.
- These rapid advancements combined with the general trend that technical and computer literacy starts early in school, contribute to successful computer-based virtual instrumentation adoption.
- Virtual instrumentation hardware uses widely available semiconductors to deliver high-performance measurement front ends.



# Traditional Instruments and Virtual Instruments

- Various interface standards are used to connect external devices to the computer. PC is the dominant computer system in the world today.
- VI is supported on the PC under Windows, Linux, Macintosh, Sun, and HP operating systems.
- All VI platforms provide powerful Graphical User Interfaces (GUIs) for development and implementation of the solutions.

# Hardware and Software in Virtual Instrumentation

- Role of Hardware

- ▶ Input/Output plays a critical role in virtual instrumentation. To accelerate test, control and design, I/O hardware must be rapidly adaptable to new concepts and products.
- ▶ Virtual instrumentation delivers this capability in the form of modularity within scalable hardware platforms.
- ▶ Virtual instrumentation is software-based ; if we can digitize it, we can measure it.
- ▶ Standard hardware platforms that house the I/O are important to I/O modularity.
- ▶ Laptops and desktop computers provide an excellent platform where virtual instrumentation can make the most of existing standards such as the USB, PCI, Ethernet, and PCMCIA buses.

# Hardware and Software in Virtual Instrumentation

- Role of Software

- ▶ Software is the most important component of a virtual instrument.
- ▶ With the right software tool, engineers and scientists can efficiently create their own applications by designing and integrating the routines that a particular process requires.
- ▶ You can also create an appropriate user interface that best suits the purpose of the application and those who will interact with it.
- ▶ You can define how and when the application acquires data from the device, how it processes, manipulates and stores the data, and how the results are presented to the user.
- ▶ When dealing with a large project, engineers and scientists generally approach the task by breaking it down into functional solvable units.

# Hardware and Software in Virtual Instrumentation

- Role of Software

- ▶ A virtual instrument is not limited or confined to a stand-alone PC.
- ▶ In fact, with recent developments in networking technologies and the Internet, it is more common for instruments to use the power of connectivity for the purpose of task sharing.
- ▶ Every virtual instrument is built upon flexible, powerful software by an innovative engineer or scientist applying domain expertise to customize the measurement and control application.
- ▶ Virtual instrumentation software can be divided into several different layers like the application software, test and data management software, measurement and control services software as shown in Figure

# Hardware and Software in Virtual Instrumentation

- Layers of Virtual Instrumentation Software

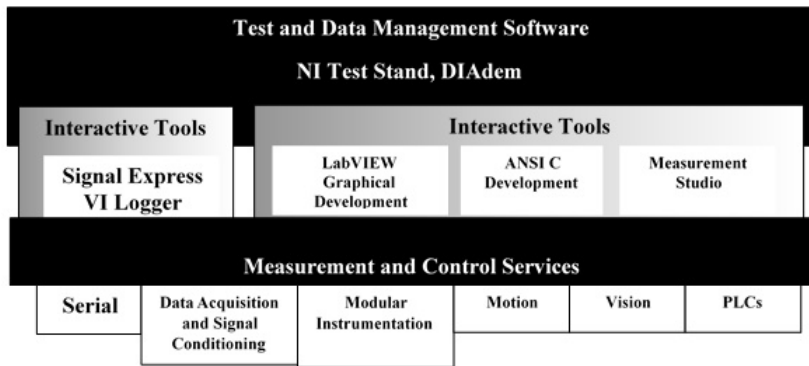


FIGURE – Layers of Virtual Instrumentation Software

# Text-based programming and Graphical programming

- Comparison

	<b>Text-based programming</b>	<b>Graphical Programming</b>
1	<i>Syntax</i> must be known to do programming	<i>Syntax</i> is knowledge but it is not required for programming.
2	The execution of the program is from <i>top to bottom</i>	The execution of program is from <i>left to right</i>
3	To check for the <i>error</i> the programs has to be compiled or executed	<i>Errors</i> are indicated as we wire the blocks
4	<i>Front Panel</i> design needs extra coding or needs extra work	<i>Front Panel</i> design is a part of programming
5	Text based programming is <i>non interactive</i>	Graphical Programming is highly <i>interactive</i>

# Text-based programming and Graphical programming

- Comparison

	<b>Text-based programming</b>	<b>Graphical Programming</b>
6	This is text-based programming where the programming is a <i>conventional method</i>	The programming is <i>Data Flow programming</i>
7	<i>Logical Error</i> finding is easy in large programs	<i>Logical Error</i> finding in large programs is complicated
8	Program flow is <i>not visible</i>	Data flow is <i>visible</i>
9	It is <i>test-based programming</i>	It is <i>icon based</i> programming and wiring
10	Passing parameters to <i>Sub routine</i> is difficult	Passing parameters to <i>Sub VI</i> is easy

# Advantages of LabVIEW

- Advantages
  - ▶ Graphical User Interface
  - ▶ Drag-and-Drop built-in functions
  - ▶ Modular design and hierarchical design
  - ▶ Multiple high level development tools
  - ▶ Professional Development Tools
  - ▶ Multi platforms
  - ▶ Reduces cost and preserves investment
  - ▶ Flexibility and scalability
  - ▶ Connectivity and instrument control



# Advantages of LabVIEW

- Advantages

- ▶ Open environment
- ▶ Distributed development
- ▶ Visualization capabilities
- ▶ Rapid development with express technology
- ▶ Compiled language for fast execution
- ▶ Simple application distribution
- ▶ Target management
- ▶ Object-oriented design
- ▶ Algorithm design

# Software Environment

- Software Environment

- ▶ Three steps to create our application
  - 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.
- ▶ The two LabVIEW 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.

# Software Environment

- Software Environment

- ▶ In LabVIEW, 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.

# Software Environment

## ● Launch LabVIEW

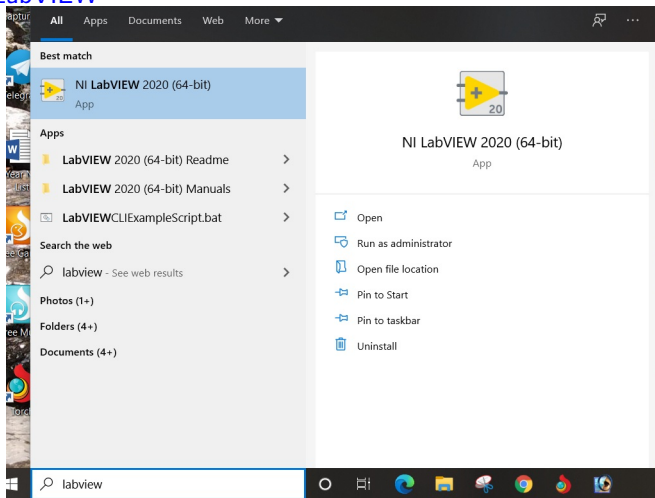


FIGURE – Launch LabVIEW

# Software Environment

- Launch LabVIEW

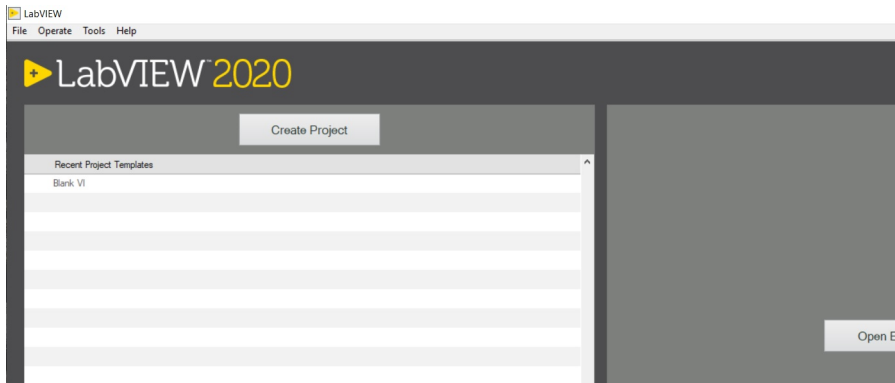


FIGURE – Create Project

# Creating a VI

- Launch Labview

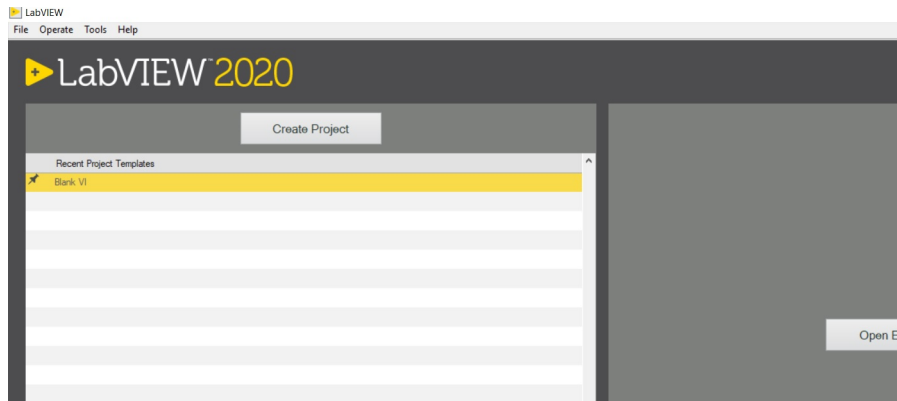


FIGURE – Blank VI

# Creating a VI

- Launch Labview

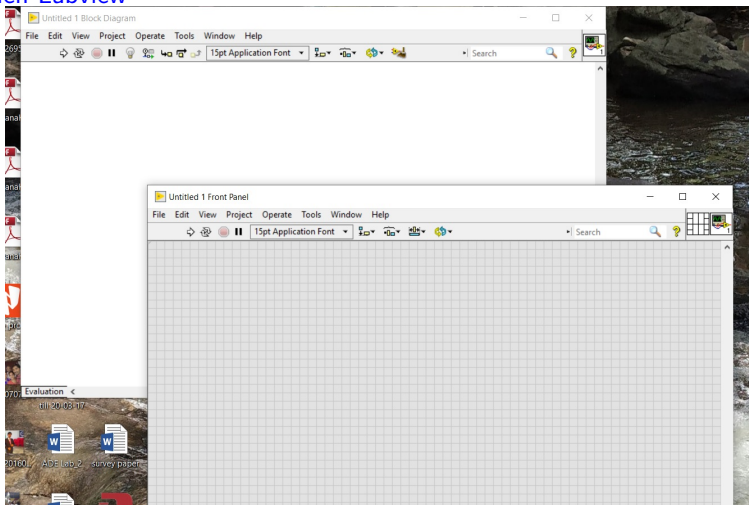


FIGURE – LabVIEW Windows

# Creating a VI

- Launch Labview

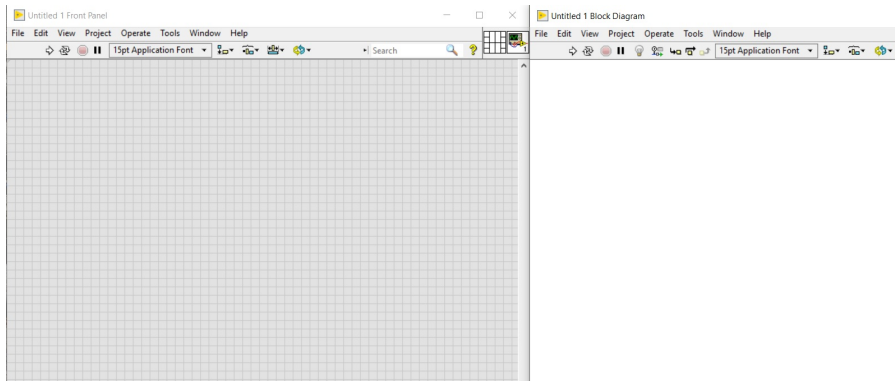






FIGURE – LabVIEW Windows



# Front Panel Toolbar

- List of Front panel toolbars button

	<p><b><i>Run Button</i></b> : To run a VI and it appears as a solid white arrow.</p>
	<p><b><i>Run Continuously Button</i></b> : To run the VI until you abort or pause execution. Click the button again to disable continuous running.</p>
	<p><b><i>Broken Run Arrow</i></b> : means a non-executable VI and the VI you are creating or editing contains errors. Click this button to display the Error list window, which lists all errors and warnings.</p>
	<p><b><i>Abort Execution Button</i></b> : To stop the VI immediately if there is no other way to stop the VI</p>




# Front Panel Toolbar

- List of Front Panel toolbars button

	<p><i>Text Settings</i> : To change the font settings including size, style, colour.</p>
	<p><i>Align Objects</i> : To align objects along axes, including vertical, top edge, left.</p>
	<p><i>Distribute Objects</i> : To space objects evenly, including gaps, compression</p>
	<p><i>Resize Objects</i> : To resize multiple front panel objects to the same size.</p>

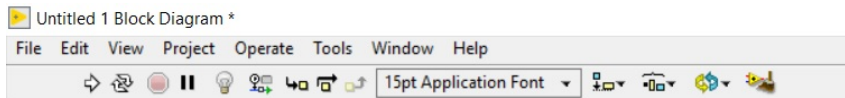
# Front Panel Toolbar

- List of Front Panel Toolbars button

	<p><i>Reorder</i> : When you have objects that overlap each other and you want to define which one is in front or back of another, select one of the objects with the positioning tool</p>
	<p><i>Pause</i> : To pause a running VI.</p>
	<p><i>Show Context Help Window button</i> : To toggle the display of the Context Help window.</p>




# Block Diagram Toolbar

- List of Block Diagram Toolbar buttons





# Block Diagram Toolbar

- List of Block Diagram Toolbar buttons

	<p><b>Highlight Execution button</b> : To display an animation of the block diagram execution when you click the Run button. See the flow of data through the block diagram. Click the button again to disable execution highlighting.</p>
	<p><b>Retain Wire Values</b> : To save the wire values at each point in the flow of execution so that when you place a probe on the wire, you can immediately retain the most recent value of the data that passed through the wire. You must successfully run the VI at least once before you are able to retain the wire values.</p>
	<p><b>Clean-Up-Diagram</b> : Specifies whether to clean up the wires on the subsystem VI.</p>

# Block Diagram Toolbar

- List of Block Diagram Toolbar buttons

	<p><b>Step Into</b> : To open a node and pause. When you click the Step Into button again, it executes the first action and pauses at the next action of the subVI or structure. You also can press Ctrl and down arrow keys. Single-stepping through a VI steps through the VI node by node. Each node blinks to denote when it is ready to execute. By stepping into the node, you are ready to single-step inside the node.</p>
	<p><b>Step Over</b> : To execute a node and pause at the next node. You also can press Ctrl and right arrow keys. By stepping over the node, you execute the node without single-stepping through the node</p>

# Front Panel Controls and Indicators

- **Controls and Indicators**

- ▶ Controls are knobs, push buttons, dials and other input devices.
- ▶ Indicators are graphs, LEDs and other displays.
- ▶ Controls simulate instrument input devices and supply data to the block diagram of the VI.
- ▶ Indicators simulate instrument output devices and display data the block diagram acquires or generates.
- ▶ Every control or indicator has a data type associated with it. They are **numeric data type, boolean data type, string data type**

# Front Panel Controls and Indicators

- Data Types

- ▶ Numeric data type are of integer or real. The two most commonly used numeric objects are the **numeric control**, **numeric indicator**
- ▶ Boolean data type represents data that only has two parts, such as **TRUE** and **FALSE** or **ON** and **OFF**. Boolean objects simulate switchesm push buttons and LEDs.
- ▶ String data type which is a sequence of **ASCII** characters. Use string controls to receive text from the user such as a **password** or user name and indicators to display text to the user.



# Block Diagram

- Block diagram objects include
  - ▶ Terminals
  - ▶ Functions
  - ▶ Wires
  - ▶ Constants.

# Block Diagram

- **Terminals**

- ▶ Front panel object appear as **terminals** on the block diagram.
- ▶ Terminals are entry and exit ports that exchange information between the front panel and block diagram.
- ▶ Terminals are analogous to parameters and constants in text-based programming languages. Types of terminals include control or indicator terminals and node terminals.
- ▶ Control and indicator terminals belong to front panel controls and indicators.
- ▶ Data you enter into the front panel controls enter the block diagram through the control terminals.
- ▶ The terminals represent the data type of the control or indicator.
- ▶ You can configure front panel controls or indicators to appear as icon or data type terminals on the block diagram.
- ▶ By default, front panel objects appear as icon terminals.
- ▶ To display a terminal as a data type on the block diagram, right-click the terminal and select *View As Icon* from the shortcut menu

# Block Diagram

- **Functions**

- ▶ Functions are the fundamental operating elements of LabVIEW.
- ▶ Functions do not have front panels or block diagrams but do have connector panes.
- ▶ Double-clicking a function only selects the function.
- ▶ A function has a pale yellow background on its icon.

# Block Diagram

- Wires

- ▶ You can transfer data among block diagram objects through wires.
- ▶ Each wire has a single data source, but you can wire it to many VIs and functions that read the data.
- ▶ Wires are different colors, styles and thicknesses, depending on their data types.
- ▶ A broken wire appears as a dashed black line with a red X in the middle.
- ▶ Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types.
- ▶ You must connect the wires to inputs and outputs that are compatible with the data that is transferred with the wire.
- ▶ You cannot wire an array output to a numeric input.
- ▶ In addition, the direction of the wires must be correct. You must connect the wires to only one input and at least one output.
- ▶ **You cannot wire two indicators together**
- ▶ **<Ctrl+B>** to delete all broken wires or right click and select **Clean Up Wire** to reroute the wire.

# Data Flow Program

- Data Flow Program

- ▶ LabVIEW follows a dataflow model for running VIs.
- ▶ A block diagram node executes when all its inputs are available.
- ▶ When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path.
- ▶ Visual Basic, C++, JAVA, and most other text-based programming languages follow a **control flow model of program execution**.

# Keyboard Shortcuts

- Frequently uses menu options and keyboard shortcuts

- < Ctrl + S > Save a VI

- < Ctrl + R > Run a VI

- < Ctrl + E > Toggle between the front panel and block diagram

- < Ctrl + H > Activate *Context Help* window.

- < Ctrl + B > Remove all broken wires

- < Ctrl + F > Find object

Press the < Ctrl > key while using the positioning tool to click and drag a selection to **duplicate an object**

# Summary

- Summary

- ▶ Virtual instruments (VIs) have three main parts — the front panel, the block diagram, and the icon and connector pane.
- ▶ The front panel is the user interface of a LabVIEW program and specifies the inputs and displays the outputs of the VI.
- ▶ Place controls (inputs) and indicators (outputs) in the front panel window.
- ▶ Control terminals have thicker borders than indicator terminals.
- ▶ All front panel objects have property pages and shortcut menus.
- ▶ The block diagram contains the executable graphical source code composed of nodes, terminals, wires, and functions on the block diagram to create measurement code.
- ▶ Use the *Wiring* tool to connect diagram objects.

# Summary

- Summary

- ▶ To change a control to an indicator or to change an indicator to a control, right-click the object and select *Change to Indicator* or *Change to Control* from the shortcut menu. The broken *Run* button appears on the toolbar to indicate the VI is non executable. Click the broken *Run* button to display the *Error list* window, which lists all the errors.
- ▶ Various debugging tools and options such as setting probes and breakpoints, execution highlighting, and single stepping are available.
- ▶ Use the *Search* button on the *Controls* and *Functions* palettes to search for controls, VIs and functions.
- ▶ All LabVIEW objects and empty space on the front panel and block diagram have associated shortcut menus, which you access by right-clicking an object, the front panel or the block diagram.
- ▶ Use execution highlighting, single-stepping, probes and breakpoints to debug VIs by animating the flow of data through the block diagram.
- ▶ Use the *Help* menu to display the Context Help window and the LabVIEW Help, which describes most palettes, menus, tools, VIs, functions and features.



# Problems

- 1 Check controls and indicators.
  - ▶ Numeric Control and Indicator
  - ▶ Boolean Control and Indicator
  - ▶ String Control and Indicator
  - ▶ Dial connected to Gauge and Meter
  - ▶ Thermometer to Numeric Indicator
  - ▶ Tank to Numeric Indicator
  - ▶ Vertical Slide Control to Horizontal Slide Control

# Problems

- 2 Add, multiply, subtract and divide two numeric inputs.

**Solution :** The front panel has controls A and B and four indicators. the block diagram has the respective terminals and functions.

- 3 Find whether the given number is odd or even

# Problems

- 4 Add and multiply more than two numeric inputs.

**Solution :** The front panel has three numeric inputs while the block diagram contains compound arithmetic functions.

- 5 Divide two numbers and find the remainder and quotient.

**Solution :** Using the function Quotient and Remainder.

# Problems

- 6 Compute the expressions

$$Y = (A * B * C) + (D * E)$$

$$Y = mx + C$$

- 7 Convert Celsius to Fahrenheit

$$F = (1.8 * C + 32)$$

- 8 Perform various Boolean Operations (AND, OR, NAND, NOR, XOR)

# Assignment

- 1 Divide two numbers and glow an **LED** if the result of the division is infinity (i.e the divisor is zero)
- 2 Create **NOT**, **AND**, **OR** gates using **NAND** gate and verify their truth table.
- 3 The population of a town is 80,000 and the percentage of men is 52. The percentage of total literacy is 48. If the total percentage of literate men is 35 of the total population, build a VI to find the total number of illiterate men and women.
- 4 Find the equivalent gray code for a given BCD.
- 5 Find the equivalent BCD of an input binary value.
- 6 Design a  $4 \times 1$  multiplexer with enable and select options.
- 7 From the given two numeric inputs, find the maximum value and minimum value.

## 1 Unit-2 Modular Programming and Loops

- Objective
- Pre-requisite
- Summary
- Examples/Work-Out
- Assignment Questions

# Table of Contents

- 1 Introduction
- 2 Modular Programming in Labview
- 3 Creating an Icon, Building a connector pane
- 4 Displaying SUBVIs, Creating SUBVIs, Editing SUBVIs.
- 5 Repetition and Loops
- 6 Shift Registers
- 7 Feedback nodes
- 8 Local and Global Variables

# Introduction-Modular Programming

- Objective :
- Pre-requisite :
  - ▶ Create Numeric Controls
  - ▶ Create Numeric Indicators
  - ▶ Basic Data Types
  - ▶ Basic Functions and Logic
- Introduction
  - ▶ Modular programming refers to the idea that programs are easier to read, to write, to debug, and to maintain if they are divided into smaller subprograms.
  - ▶ Benefits of Modular Programming
    - ① It makes our programs easier to write because individual components can be independently written and tested.
    - ② It makes the **main** part of the code easier to read since long code sections are replaced with simple functions (whose internal code is hidden in another file)
    - ③ Individual components can be reused in other programs
  - ▶ The main program simply acts as an outline or driver, triggering execution of the program units that accomplish the tasks.



# Modular Programming in LabVIEW

- Modular Programming

- ▶ Modular programming helps manage changes and debug the block diagram quickly.
- ▶ Modularity defines the degree to which your VI is composed of discrete components such that a change to one component has minimal impact on other components.
- ▶ These components are called modules or subVIs.
- ▶ Modularity increases the readability and reusability of your VIs.
- ▶ A VI within another VI is called a **subVI**.
- ▶ A subVI corresponds to a subroutine in text-based programming languages.
- ▶ You can reuse a subVI in other VIs. If you use a VI as a subVI, the icon identifies the subVI when it is called from the block diagram of another VI.

# Creating an Icon and Building a connector pane

- **Creating an Icon**

- ▶ The default icon of a VI contains a number that indicates how many new VIs you have opened since launching LabVIEW. You can create custom icons to replace the default icon by completing the following steps :
  - ① Right-clicking the icon in the upper-right corner of the front panel or block diagram and select Edit Icon from the shortcut menu to display the *Icon Editor* dialog box.
  - ② Double-click the hatched box which will select the entire icon. Delete the selected portion.
  - ③ Double-click the rectangular box which will create a border for the icon.
  - ④ Use the line/pencil tool to draw.
  - ⑤ Double-click the *Edit Text* tool 'A' to edit the required text.
  - ⑥ Choose the *Text Tool* Font to edit font, font size, color and alignment of the text.
  - ⑦ Use the *Select Color* tool to choose the background color of the icon.
  - ⑧ Use the *Fill With Color* tool to change the background color of the icon.
  - ⑨ Click *OK* to save the icon.

# Creating an Icon and Building a connector pane

- **Creating an Icon**

- ▶ Use the *Edit* menu to cut, copy and paste images from and to the icon.
- ▶ When you select a portion of the icon and paste an image, LabVIEW resizes the image to fit into the selection area.
- ▶ You also can drag a graphic from anywhere in the file system and place it in the upper-right corner of the front panel.
- ▶ LabVIEW converts the graphic to a 32×32 pixel icon. Depending on the type of monitor you use, you can design a separate icon for monochrome, 16-color and 256-color mode.
- ▶ Use the Copy from option on the right side of the Icon Editor dialog box to copy from a color icon to a black-and-white icon and vice versa.
- ▶ After you select a Copy from option, click the OK button to complete the change.

# Creating an Icon and Building a connector pane

- Building a Connector Pane

- ▶ The connector pane is a set of terminals that correspond to the controls and indicators of that VI.
- ▶ The connector pane defines the inputs and outputs you can wire to the VI so that you can use it as a subVI.
- ▶ A connector pane receives data at its input terminals and passes the data to the block diagram code through the front panel controls or receives the results at its output terminals from the front panel indicators.
- ▶ To define a connector pane, right-click the icon in the upper-right corner of the front panel and select Show Connector from the shortcut menu to display the connector pane.
- ▶ The connector pane appears in place of the icon. you view the connector pane for the first time, you see a default connector pattern.
- ▶ You can select an appropriate pattern by right-clicking the connector pane and *selecting Patterns* from the shortcut menu.
- ▶ After you select a connector pane pattern, you can customize it to suit the VI by adding, removing or rotating the terminals.

# Creating an Icon and Building a connector pane

- Building a Connector Pane

- ▶ To add a terminal to the pattern, place the cursor where you want to add the terminal, right-click, and select *Add Terminal* from the shortcut menu.
- ▶ To remove an existing terminal from the pattern, right-click the terminal and select *Remove Terminal* from the shortcut menu.
- ▶ To change the spatial arrangement of the connector pane patterns, right-click the connector pane and select Flip Horizontal, Flip Vertical, or Rotate 90 Degrees from the shortcut menu.

# Building a connector pane

- **Assigning Terminals to Controls and Indicators**

- ▶ After you select a pattern to use for the connector pane, you must assign a front panel control or indicator to each of the connector pane terminals.
- ▶ To link controls and indicators to the connector pane, place inputs on the left and outputs on the right to prevent complicated or confusing wiring patterns.

## Building a connector pane

- Complete the following steps to assign terminals to controls and indicators in a connector pane
  - ▶ Ensure that you have selected a pattern sufficient for the number of controls and indicators you want to assign to the connector pane.
  - ▶ Right-click the icon in the upper-right corner of the front panel and select Show Connector from the shortcut menu to display the connector pane. The connector pane appears in place of the icon.
  - ▶ Click a terminal of the connector pane. The tool automatically changes to the wiring tool and the terminal turns black.
  - ▶ Click the front panel control or indicator you want to assign to the terminal. A marquee highlights the object.
  - ▶ Click an open space of the front panel. The marquee disappears, and the terminal changes to the data type color of the control to indicate that you connected the terminal. **If the connector pane terminal turns white, a connection was not made**
  - ▶ If you need to change the control or indicator assigned to a terminal, you must first delete the connection and assign another control or indicator to the terminal.
  - ▶ If necessary, confirm each terminal connection. You can specify which terminals are required, recommended and optional.

# Building a connector pane

- **Confirming Terminal Connections**

- ▶ To confirm which control or indicator is assigned to a connector pane terminal, click the terminal in the connector pane.
- ▶ A marquee highlights the assigned object. You also can use the wiring tool to click the control or indicator.
- ▶ The color of the assigned terminal in the connector pane darkens.

- **Deleting Terminal Connections**

You can delete connections between terminals and the corresponding controls or indicators individually or all at once. Complete the following steps to delete a terminal connection.

- ▶ Right-click the terminal you want to disconnect on the connector pane and select *Disconnect This Terminal* from the shortcut menu.
- ▶ The terminal turns white to indicate that the connection no longer exists.
- ▶ *Disconnect This Terminal* is different from *Remove Terminal*, in that *Disconnect this Terminal* does not remove the terminal from the pattern.
- ▶ To delete all connections on the connector pane, right-click anywhere on the connector pane and select *Disconnect All Terminals* from the shortcut menu.



# Displaying, Creating, Editing SubVIs

- **Displaying SubVIs**

- ▶ You can convert a section of a VI into a subVI by using the positioning tool to select the section of the block diagram.
- ▶ Then select *Edit»Create SubVI* as from the menu to convert the selected portion into a subVI. SubVI created with default icon.
- ▶ An icon for the new subVI replaces the selected section of the block diagram.
- ▶ LabVIEW creates controls and indicators for the new subVI, automatically configures the connector pane based on the number of control and indicator terminals you selected, and wires the subVI to the existing wires.

# Displaying, Creating, Editing SubVIs

- Opening and Editing SubVIs

- ▶ When you double-click a subVI, a front panel and a block diagram appear, rather than a dialog box in which you can configure options.
- ▶ The subVI controls and indicators receive data from and return data to the block diagram of the calling VI. Click the Select a VI icon or text on the Functions palette, navigate to and double-click a VI, and place the VI on a block diagram to call a created subVI. [Complete the following steps to open a subVI and edit it.](#)
  - 1 Use the operating or positioning tool to double-click the subVI on the block diagram.
  - 2 LabVIEW displays the front panel of the subVI.
  - 3 You also can press the <Ctrl> key and use the operating or positioning tool to double-click the subVI on the block diagram to display the block diagram and front panel of the subVI.
  - 4 Edit the subVI.

# Repetition and Loops

- Repetition and Loop

- ▶ Loops and case statements of text-based programming languages are represented as structures in graphical programming.
- ▶ Repetition and loop are used to perform an action frequently with variations in the details each time.
- ▶ LabVIEW consists of FOR Loop and WHILE Loop. These loops are used to control repetitive operations.
- ▶ Structures on the block diagram are used to repeat blocks of code and to execute code conditionally or in a specific order.
- ▶ LabVIEW includes structures like the While Loop, For Loop, Case structure, Stacked Sequence structure, Flat Sequence structure, Event structure, and Formula Node.

# Repetition and Loops

- FOR Loops

- ▶ 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 Loop.
- ▶ **N** The value is the count terminal 'N' indicates how many times to repeat the subdiagram. **A VI will not run if it contains a For Loop that does not have a numeric value wired to the count terminal**
- ▶ **i** The iteration terminal 'i' contains the number of completed iterations. The iteration count always starts at **zero**.
- ▶ **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**

# Repetition and Loops

- **WHILE Loops**

- ▶ 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.
- ▶ 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.

# Repetition and Loops

- WHILE Loops

- ▶ The While Loop executes the subdiagram until the conditional terminal, and receives a specific Boolean value. **The default behavior and appearance of the conditional terminal is Stop if True.**
- ▶ When a conditional terminal is Stop if True, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value.
- ▶ When a conditional terminal is Continue if True, the While Loop executes its subdiagram until the conditional terminal receives a FALSE value.
- ▶ 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.

# Repetition and Loops

- Structure Tunnels

- ▶ Tunnels feed data into and out of structures.
- ▶ The tunnel appears as a solid block on the border of the loop. The block is the color of the data type wired to the tunnel.
- ▶ Data passes out of a loop after the loop terminates.
- ▶ When a tunnel passes data into a loop, the loop executes only after data arrives at the tunnel.
- ▶ The iteration terminal is connected to a tunnel.
- ▶ The value in the tunnel does not pass to the *Iteration Number indicator* until the While Loop has finished execution.
- ▶ Only the last value of the iteration terminal displays in the *Iteration Number* indicator.

# Repetition and Loops

- **Shift Registers**

- ▶ When programming with loops, you often need to access data from previous iterations of the loop.
- ▶ Two ways of accessing this data include the shift register and the feedback node.
- ▶ Shift registers are used with For Loops and While Loops to transfer values from one loop iteration to the next.
- ▶ **Shift registers are similar to static variables in text-based programming languages.**
- ▶ A shift register **appears as a pair of terminals**, directly opposite each other on the vertical sides of the loop border.
- ▶ The terminal on the right side of the loop contains an up arrow and stores data on the completion of an iteration.
- ▶ LabVIEW transfers the data connected to the right side of the register to the next iteration.
- ▶ After the loop executes, the terminal on the right side of the loop returns the last value stored in the shift register.
- ▶ **Create a shift register by right-clicking the left or right border of a loop and selecting the Add shift register from the shortcut menu**



# Repetition and Loops

- **Shift Registers**

- ▶ A shift register transfers any data type and automatically changes to the data type of the first object wired to the shift register.
- ▶ The data you wire to the terminals of each shift register must be the same type.
- ▶ You can add more than one shift register to a loop.
- ▶ If you have multiple operations that use previous iteration values within our loop, you can use multiple shift registers to store the data values from those different processes in the structure.
- ▶ **Initializing Shift Registers :**
  - Initialize a shift register by wiring a control or constant to the shift register terminal on the left side of the loop.
  - If you do not initialize the shift register, the loop uses the value written to the shift register when the loop last executed or the default value for the data type if the loop has never executed.

# Repetition and Loops

- Shift Registers

- ▶ Stacked Shift Registers :

- Stacked shift registers remember values from multiple previous iterations and carry those values to the next iterations.
- To create a stacked shift register, right-click the left terminal and select Add Element from the shortcut menu.

- ▶ Replacing Tunnels with Shift Registers :

- Tunnels can be replaced with shift registers wherever necessary. To replace a tunnel into a shift register, right-click the tunnel and select Replace with Shift Register.
- If no tunnel exists on the loop border opposite of the tunnel you right-clicked, LabVIEW automatically creates a pair of shift register terminals.
- If one or more than one tunnel exists on the loop border opposite of the tunnel you right-clicked, the mouse pointer will turn to the symbol of a shift register.

# Repetition and Loops

- Shift Registers

- ▶ Replacing Shift Registers with Tunnels :

- Replace shift registers with tunnels when you no longer need to transfer values from one loop iteration to the next.
- To replace a shift register with a tunnel, right-click the shift register and select Replace with Tunnels.
- If you replace an output shift register terminal with a tunnel on a For Loop, the wire to any node outside the loop breaks because the For Loop enables auto-indexing by default.
- Right click the auto-indexed tunnel and select Disable Indexing on the tunnel to correct the broken wire.
- This problem does not occur in While Loops because auto-indexing is disabled by default in While Loops.

# Repetition and Loops

- Feedback Nodes

- ▶ When the output of a node is connected directly to the input, the feedback node is generated automatically. The feedback node appears automatically in a For Loop or While Loop if we wire the output of a node or group of nodes to the input of that node or group of nodes.
- ▶ Like a shift register, the feedback node stores data when the loop completes an iteration, sends that value to the next iteration of the loop, and transfers any data type.
- ▶ Use the feedback node to avoid unnecessarily long wires in loops.
- ▶ The feedback node arrow indicates the direction in which the data flows along the wire.
- ▶ The arrow automatically changes direction if the direction of data flow changes.

# Control Timing and Communicating among Multiple Loops

- Control Timing

- ▶ When a loop finishes executing an iteration, it immediately begins executing the next iteration unless it reaches a stop condition.
- ▶ Most applications need precise control of the frequency or timing of the iteration to be maintained between successive operations of the loop.
- ▶ You can use a wait function in the loop to wait an amount of time in milliseconds before the loop re-executes.
- ▶ LabVIEW consists of **two wait functions**.
- ▶ A *wait function* is placed inside a loop to allow a VI to sleep for a set amount of time. This allows your processor to address other tasks during the wait time. Wait functions use the operating system millisecond clock. They are ***Wait Until Next ms Multiple*** and ***Wait (ms)*** functions

# Control Timing and Communicating among Multiple Loops

- Control Timing

- ▶ Wait Until Next ms Multiple

- The *Wait Until Next ms Multiple* function monitors a millisecond counter and waits until the millisecond counter reaches a multiple of the time you specify.
    - You can place this function within a loop to control the loop execution rate. For this function to be effective, your code execution time must be less than the time specified for this function.
    - The execution rate for the first iteration of the loop is indeterminate.

- ▶ Wait (ms)

- The *Wait (ms)* function adds the wait time to the code execution time.
    - The *Wait (ms)* function waits until the millisecond counter counts to an amount equal to the input you specify. This function guarantees that the loop execution rate is at least the amount of the input you specify.

# Control Timing and Communicating among Multiple Loops

- Communicating among Multiple Loops

- ▶ The flow of data determines the execution order of block diagram elements.
- ▶ Variables are block diagram elements that allow you to access or store data in another location.
- ▶ The actual location of the data varies depending upon the type of the variable.
- ▶ Local variables store data in front panel controls and indicators.
- ▶ Global variables and single process-shared variables store data in special repositories that you can access from multiple VIs.
- ▶ Functional global variables store data in While Loop shift registers.

# Control Timing and Communicating among Multiple Loops

- Local Variables

- ▶ Local variables transfer data within a single VI and allow data to be passed between parallel loops.
- ▶ Two ways to create a local variable are right-click on an object's terminal and select Create » Local Variable.
- ▶ A local variable icon for the object appears on the block diagram
- ▶ Another way is to select the Local Variable from the Structures palette. Create the front panel and select a local variable from the Functions palette and place it on the block diagram.
- ▶ To associate a local variable with a control or indicator, right-click the local variable node and select Select Item from the shortcut menu.
- ▶ You also can configure a variable to behave as a data source, or a read local or global. Right-click the variable and select Change To Read from the shortcut menu to configure the variable to behave as a control.
- ▶ When this node executes, the VI reads the data in the associated front panel control or indicator.
- ▶ When you write to a local variable, you update its corresponding front panel object and when you read from a local variable, you read the current value of its corresponding front panel object. Initialize local and global variables before reading them



# Control Timing and Communicating among Multiple Loops

- **Global Variables**

- ▶ Global variables are built-in LabVIEW objects. You can use variables to access and pass data among several VIs that run simultaneously.
- ▶ A local variable shares data within a VI; a global variable also shares data, but it shares data with multiple VIs.
- ▶ When you create a global variable, LabVIEW automatically creates a special global VI, which has a front panel but no block diagram.
- ▶ Add controls and indicators to the front panel of the global VI to define the data types of the global variables. Select a global variable from the Functions palette and place it on the block diagram.
- ▶ Double-click the global variable node to display the front panel of the global VI. Place controls and indicators on this front panel the same way you do on a standard front panel.
- ▶ A global variable front panel with a numeric, a string, and a cluster containing a digital and a Boolean control. The toolbar does not show the Run, Stop or related buttons as a normal front panel.

# Control Timing and Communicating among Multiple Loops

- Global Variables

- ▶ After you finish placing objects on the global VI front panel, save it and return to the block diagram of the original VI. You then must select which object in the global VI that you want to access.
- ▶ Right-click the global variable node and select a front panel object from the Select Item shortcut menu. The shortcut menu lists all the front panel objects that have owned labels.
- ▶ You also can use the operating tool or labeling tool to click the local variable node and select the front panel object from the shortcut menu. If you want to use this global variable in other VIs, select *Functions » All Functions » Select a VI*.
- ▶ The global variable is associated with the first front panel object with an owned label that you placed in the global VI.
- ▶ Right-click the global variable node you placed on the block diagram and select a front panel object from the Select Item shortcut menu to associate the global variable with the data from another front panel object

# Summary

- **Modular Programming**

- ▶ Modularity increases the readability and reusability of your VIs.
- ▶ A VI within another VI is called modules or subVI.
- ▶ SubVIs correspond to a subroutine in text-based programming languages.
- ▶ The upper-right corner of the front panel and block diagram displays the icon for the VI.
- ▶ After you build a VI front panel and block diagram, build the icon and the connector pane to use the VI as a subVI.
- ▶ Right-click the icon in the upper-right corner of the front panel or block diagram and selecting Edit Icon, you can create custom icons to replace the default icon.
- ▶ Right-click the icon in the upper-right corner of the front panel and select Show Connector.
- ▶ The connector pane is a set of terminals that correspond to the controls and indicators of that VI. Define connections by assigning a front panel control or indicator to each of the connector pane terminals using the wiring tool.
- ▶ Load subVIs using the Select a VI option in the All Functions palette or dragging the icon onto a new diagram.

# Summary

- Repetition and Loops

- ▶ The While Loop executes the subdiagram until the conditional terminal receives a specific Boolean value.
- ▶ By default, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value.
- ▶ The For Loop executes a subdiagram a set number of times.
- ▶ The Wait Until Next ms Multiple function makes sure that each iteration occurs at certain intervals. Use this function to add timing to loops.
- ▶ The Wait (ms) function waits a set amount of time.
- ▶ Use shift registers on For Loops and While Loops to transfer values from one loop iteration to the next.
- ▶ Create a shift register by right-clicking the left or right border of a loop and selecting Add Shift Register from the shortcut menu.
- ▶ To configure stacked shift register to remember values from multiple previous iterations and carry over values to the next iteration, right-click the left terminal and select Add Element from the shortcut menu.
- ▶ The feedback node stores data when the loop completes iteration, sends that value to the next iteration of the loop and transfers any data type.

# Assignment

- 1 Create a VI to compute full adder logic using half ladder logic as sub VI.
- 2 Create a VI to find the decimal equivalent of a binary number using sub VI.
- 3 Create a VI to find the Grey code equivalent of a BCD number using sub VIs.
- 4 Create a VI to find the average of two numbers and convert a section of a VI into a subVI.
- 5 Create a subVI to compute the average of five students marks

# Problems

- 6 Factorial of the given number using FOR Loop and Shift Register.

**Solution :** The front panel has the number and its factorial, while the block diagram contains the codes to solve the factorial.

- 7 Sum of first  $n$  natural numbers using a WHILE Loop with a feedback node..

**Solution :** Given a number  $n$ , the sum of first  $n$  natural numbers is obtained when the program is run.

- 8 To change the state of the Boolean indicator  $n$  times between TRUE and FALSE.

**Solution :** Build the front panel with Boolean indicator and numeric indicator.

# Problems

- 9 Sum of first 10 natural numbers using FOR Loop.

**Solution :** The front panel has the number 10 and its sum, while the block diagram contains the codes to find the sum.

- 10 Convert decimal number to binary number using FOR Loop.

**Solution :** Given a number  $n$ , and its binary equivalent is in the Front panel corresponding programming is in the front panel.

- 11 To change the state of the Boolean indicator  $n$  times between TRUE and FALSE.

**Solution :** Build the front panel with Boolean indicator and numeric indicator.

# Assignment

- 1 Program to display a name 27 times using a FOR Loop.
- 2 Program to find the sum of first 100 natural numbers.
- 3 Program to display the numbers from 1 to 100 in steps of 3.
- 4 Determine the square of the numbers from 1 to 100 using **FOR Loop** and **WHILE Loop**
- 5 Program to generate a Fibonacci Series of  $n$  numbers.



## 1 Unit-3 Arrays and Clusters

- Objective
- Pre-requisite
- Arrays and its Functions
- Clusters and its Functions
- Summary
- Examples/Work-Out
- Assignment Questions

# Table of Contents

- 1 Creating One-Dimensional Array
- 2 Deleting, Inserting and Replacing into Arrays
- 3 Array functions
- 4 Auto Indexing
- 5 Creating Clusters control and constant
- 6 Cluster Operations
- 7 Assembling and Disassembling Clusters
- 8 Conversion between Arrays and Clusters

# Introduction-Arrays

- **Objective :** The main aim of this chapter to know about various functions can be performed in arrays using LabVIEW software environment
- **Pre-requisite :**
  - ▶ Create Numeric Controls
  - ▶ Create Numeric Indicators
- **Introduction to Arrays**
  - ▶ A group of homogeneous elements of a specific data type is known as an *array*.
  - ▶ Arrays hold a sequence of data elements, usually of the same size and same data type placed in contiguous memory locations.
  - ▶ Individual elements are accessed by their position in the array.
  - ▶ The position is given by an index, which is also called as *subscript*
  - ▶ Some arrays are multi-dimensional, generally one -and two- dimensional arrays are the most common.
  - ▶ You can build arrays of numeric, boolean, path, string and cluster data types.
  - ▶ **You cannot create arrays of arrays**

# Arrays

- 1D Array Controls, Indicators and Constants

- ▶ Create an array control or indicator on the front panel by placing an array on the front panel and dragging a data object or element, which can be numeric, boolean, string.
- ▶ Array shell can be selected from *Controls > Modern > Arrays, Matrix and Clusters palette*
- ▶ The array elements must be controls or indicators.
- ▶ Insert an object in the array shell before use the array on the block diagram.
- ▶ After placing an element in the array shell, one can expand the array either horizontally or vertically to see more number of elements.
- ▶ Once a data type is assigned to the array shell, the block diagram takes the color and lettering (in [ ]brackets) of the data type.

# Arrays

- 1D Array Controls, Indicators and Constants

- ▶ The index ranges from 0 to 3. The first element in the array is at index 0, the second element is at index 1, etc..
- ▶ In an array the element selected in the index display always refer to the element shown in the upper left corner of the element display.
- ▶ The element (9) at index 0 is not shown in the array, because index 1 is selected in the index display.

- Steps for creating an array constant

- ▶ Select an array constant from *Functions » Programming » Arrays*. Array shell appears with an index display on the left, an empty element display on the right.
- ▶ Place a constant in the array shell.
- ▶ The array shell automatically resizes to accommodate the object place in the array shell.
- ▶ **Alternative method is to copy an existing array on the front panel to the block diagram to create a constant of same data type.**

# Arrays

- 2D Arrays
  - ▶ A 2D arrays stores elements in a grid.
  - ▶ It requires a column index and a row index to locate an element both of which are zero-based.
  - ▶ To create a 2D array on the front panel, right click the index display of the array and select *Add Dimension* from the shortcut menu.

# Arrays

- **Initializing Arrays**

- ▶ When an array is initialized, define the number of elements in each dimension and contents of each element.
- ▶ An uninitialized array has a dimension but no elements.
- ▶ An uninitialized array control with all the elements are dimmed indicating that the array is uninitialized.

# Arrays

- **Deleting Elements within Arrays**

- ▶ One can delete an element within a 1D array and a row or column within a 2D.
- ▶ To delete an element in a 1D array, **right-click the array element on the front panel and select *Data Operations » Delete Element***.
- ▶ To delete a row or column in a 2D array, **right-click the array row or column on the front panel and select *Data Operations » Delete Row or Delete Column***
- ▶ Can delete elements, rows, columns and pages within array using the *Delete From Array* function.



# Arrays

- **Inserting Elements within Arrays**

- ▶ One can insert an element into a 1D array and a row or column into a 2D array.
- ▶ To add an element 1D, right click the array on the front panel and select *Data Operation » Insert Element Before*.
- ▶ To add a row or column to a 2D array, right click the array on the front panel and select *Data Operations » Insert Row Before or Insert Column Before*.
- ▶ One can insert elements, rows, column into arrays using the *Insert Into Array* function.
- ▶ Place an *Insert Into Array* function on the block diagram.
- ▶ the index input specifies the element, row, column where to insert the element or array with 0 being the first.
- ▶ **Elements are added before the value wire to index.**

# Arrays

- Replacing Elements within Arrays

- ▶ Place the *Replace Array Subset* function on the block diagram
- ▶ Wire an array of any dimension to the n-dimension array input of the Replace Array Subset function.
- ▶ The function automatically resizes based on the dimensions of the array.
- ▶ The index input specifies which element, row, column to replace.
- ▶ The new element/subarray input specifies the value you want to replace an element.
- ▶ Resize the *Replace Array Subset* function to replace another element, row, column within an array
- ▶ Run the VI

# Arrays

- **Array Functions**

- ▶ Array functions are used to create and manipulate arrays.
- ▶ Common array operations such as **Extracting individual data elements from an array, inserting, deleting or replacing data elements** using array functions.
- ▶ Array functions including ***Index Array, Replace Array Subset, Insert Into Array, Delete From Array and Array Subset***
- ▶ **Index Array** : The input to the index array function is a 1D array. By providing the index value in the output, get the array element corresponding to the index value.
- ▶ **Index Array** : When connecting a 2D array as input, the function automatically resizes to get two index inputs one for the row index and other for column index.

# Arrays

- Auto Indexing

- ▶ For loops and While loops can index and accumulate arrays at their boundaries. This is known as *auto-indexing*.
- ▶ If you wire an array to a For Loop or While Loop input terminal, can read and process every element in that array by enabling auto-indexing.
- ▶ When you auto-index an array output tunnel, the output array receives a new element from every iteration of the loop.
- ▶ The wire from the output tunnel to the array indicator becomes thicker as it changes to an array at the loop border.
- ▶ Disable auto-indexing by right clicking the tunnel and selecting *Disable Indexing* from the menu.
- ▶ Disable auto-indexing if need only the last value passed to the tunnel.
- ▶ To enable auto-indexing, right click a tunnel and select *Enable Indexing*

# Clusters

- Introduction

- ▶ Clusters group data elements of mixed types.
- ▶ **Example of a Cluster** is the LabVIEW error cluster, which combines a Boolean control(status), a numeric control (Code) and a string control (source).
- ▶ A cluster is similar to a record or a struct in text-based programming languages.
- ▶ If front panel contains more than 28 controls and indicators that want to pass to another VI, some of them grouped into a cluster and assign the cluster to a terminal on the connector pane.

# Clusters

- **Creating Cluster Controls and Indicators**

- ▶ A cluster can be created by placing a cluster shell on the front panel and then placing one of the front panel objects inside the clusters.
- ▶ Select a cluster on the *Controls » All Controls » Arrays and Cluster* palette, place it on the front panel and drag a data object or element which can be numeric, boolean, string, control or indicator, into the cluster shell.
- ▶ Resize the cluster shell by dragging the cursor while you place the cluster shell on the front panel

# Clusters

- **Creating Cluster Constants**

- ▶ To create a cluster constant on the block diagram, first select a cluster constant on the *Functions* palette.
- ▶ Next place the cluster shell on the block diagram, and finally place a string constant, numeric constant, or cluster constant in the cluster shell.
- ▶ Cluster constant can be used to store constant data or as basis for comparison with another.
- ▶ If a cluster control or indicator present in the front panel and want to create a cluster constant containing the same elements on the block diagram, can either drag that cluster from the front panel to the block diagram or right-click the cluster on the front panel and select *Create - Constant* from the shortcut menu.

# Clusters

- Order of Cluster elements

- ▶ Cluster elements have a logical order unrelated to their position in the shell.
- ▶ The first object you place in the cluster is element 0, the second is element 1 and so on.
- ▶ If the element is deleted, the order adjusts automatically.
- ▶ The cluster order determines the order in which the elements appear as terminals on the *Bundle* and *Unbundle* functions on the block diagram.
- ▶ One can view and modify the cluster order by right-clicking the cluster border and selecting *Reorder Controls In Cluster* from the shortcut menu.
- ▶ To wire clusters to each other, both clusters must have the same number of elements.
- ▶ Corresponding elements, determined by the cluster order, must have compatible data types.



# Clusters

- Order of Cluster elements

- ▶ Fig. shows the reordering of a cluster which contains a numeric control (*Digital Control*), a Boolean control (*OK Button*) and a string control.
- ▶ By clicking over the number displayed with a black background near the cluster element, can change the order of the elements.
- ▶ One must unbundle all cluster elements at once or use the *Unbundle By Name* function to access specific cluster elements.
- ▶ Cluster is either a control or an indicator.

# Clusters

- Cluster Operations

- ▶ The main cluster operations are bundle, unbundled, bundle by name and unbundle by name.
- ▶ Use the cluster functions to create and manipulate clusters.
- ▶ Some of the tasks which can be performed
  - Extract individual data elements from a cluster.
  - Add individual data elements to a cluster
  - Break a cluster out into its individual data elements
- ▶ The *Bundle* function assembles individual components into a single new cluster and allows to replace elements in an existing order.
- ▶ The *Unbundle* function splits a cluster into its individual components.
- ▶ When it is required to operate on a few elements and not the entire cluster elements, can use the *Bundle by Name function*.
- ▶ They are referenced by names rather than by position.
- ▶ The *Unbundle by Name* function returns the cluster elements whose names are specified.

# Clusters

- **Assembling Clusters**

- ▶ The *Bundle* function assembles a cluster from individual elements. This function can also be used to change the values of individual elements in an existing cluster without having to specify new values for all elements.
- ▶ When you wire a cluster to this function, the function resizes automatically to display inputs for each element in the cluster.
  - Place the Bundle function on the block diagram.
  - If necessary, resize the Bundle function to include the number of inputs you intend to use as elements in the cluster. **One cannot leave an input unwired**
  - Wire front panel control terminals or outputs from VIs and functions to the *elementinputs* of the *Bundle* function. The order in which you wire the inputs determines the cluster element order
  - Right click the *Output Cluster* terminal and select *Create » Indicator*. LabVIEW returns the bundled cluster in the cluster output.

# Clusters

- **Assembling Clusters**

- ▶ The *Bundle By Name* function is used to replace one or more elements in an existing cluster.
- ▶ This function refers to cluster elements by name instead of by their position in the cluster.
- ▶ After you wire the node to an input cluster, right-click the name terminals to select elements from the shortcut menu.
- ▶ **Use the operating tool to click the name terminals and select from a list of cluster elements.**
- ▶ All inputs are required.
- ▶ The new value for both the elements must be given, otherwise LabVIEW shows an error.

# Clusters

- **Disassembling Clusters**

- ▶ The *Unbundle* function splits a cluster into each of its individual elements.
- ▶ When you wire a cluster to this function, the function resizes automatically to display outputs for each element in the cluster wired.
- ▶ the connector pane displays the default data types for this polymorphic function.
- ▶ Unbundling elements from clusters accesses and arranges all elements in a cluster in their cluster element order.
- ▶ After unbundle elements from cluster, can wire each element to VIs, functions and indicators.
- ▶ **This method of unbundling a cluster is useful if you need to access all the elements in a cluster.**
- ▶ The steps to unbundle elements from a cluster is place the *unbundled* function on the block diagram and then wire a cluster to the *Unbundle* function.
- ▶ The data type representation of every element appears as element outputs.

# Clusters

- Disassembling Clusters

- ▶ The *Unbundle By Name* function returns the cluster elements whose names you specify.
- ▶ Do not have to keep track of the order of the elements within the cluster.
- ▶ This function does not require the number of elements to match the number in the cluster.
- ▶ After you wire a cluster to this function, you can select an individual element from the function.
- ▶ The connector pane displays the default data types for this polymorphic function

# Clusters

- **Disassembling Clusters**

- ▶ Unbundling elements from clusters by name accesses and arranges the elements in a cluster by name in their cluster element order.
- ▶ A cluster element must have a label for you to unbundle the element by name.
- ▶ After you unbundle an element(s) from a cluster by name, you can wire the element(s) to a VI, function and indicator. This method of unbundling a cluster is useful if you need to access one element from a cluster that includes elements of the same data type.
- ▶ You also can unbundle all the elements from a cluster without using the name.
- ▶ The steps to unbundle elements from a cluster by name are first place the Unbundle By Name function on the block diagram.
- ▶ Then wire a cluster to the Unbundle By Name function. The first element in the cluster element order appears as an element output

# Clusters

- Conversion between Array and Clusters

- ▶ A cluster can be converted into an array first and converted back to a cluster after performing the required operation from the available array functions.
- ▶ Convert a cluster with elements of the same data type to an array using the *Cluster to Array* function and use array functions to manipulate the contents.
- ▶ **This function cannot be used on a cluster of arrays**
- ▶ **LabVIEW does not allow any array or arrays type of structure**
  - Place a cluster on the front panel.
  - Place the *Cluster to Array* function on the block diagram.
  - Wire the cluster to the *Cluster to Array* function.
  - Right click the *Cluster To Array* function and select *Create » Indicator* from the shortcut menu to create an array indicator
  - The array indicator displays the values of the cluster.



# Clusters

- Conversion between Array and Clusters

- ▶ One can convert an array to a cluster using the *Array to Cluster* function.
- ▶ This function converts an  $n$  element, 1D array into a cluster of  $n$  element of the same type.
- ▶ This function is useful when you would like to display the elements of the same type in a front panel but still want to manipulate the elements on the block diagram by their index values.
- ▶ Clusters do not size automatically, you need to specify the cluster size by popping up on function.
- ▶ The default cluster size is 9, and the maximum size permitted is 256.
- ▶ The *Build Cluster Array* function is used to create an array of clusters where each cluster contains an array.
- ▶ The *Index and Build Cluster Array* function indexes a set of array and creates a cluster array in which the  $i^{\text{th}}$  element contains the  $j^{\text{th}}$  element of each input array.
- ▶ All array inputs need not be of the same type and the function yields a cluster array containing one element from each input array.

# Summary

- Summary

- ▶ Clusters group data elements of mixed types.
- ▶ Elements of clusters must be all controls or all indicators or constants.
- ▶ The size of components in a cluster is fixed.
- ▶ Cluster elements are accessed through the cluster order.
- ▶ If a front panel contains more than **28 controls and indicators** that you want to use programmatically, group some of them into a cluster and assign the cluster to a terminal on the connector pane to eliminate clutter on the block diagram.
- ▶ To create a cluster control or indicator, select a cluster on the *Functions » All Functions » Array and Cluster* palette, place it on the front panel, and drag controls or indicators into the cluster shell.
- ▶ Use the cluster functions located on the *Functions » All Functions » Cluster* palette to create and manipulate clusters.
- ▶ Arrays and Clusters are inter-convertible but only under certain conditions.

# Arrays - Clusters

- 1 Create a 1D numeric array using the *Build Array* function which gets array elements from numeric controls.

# Arrays - Clusters

- 2 Create a 1D numeric array from loops(For and While) using random numbers and obtain the reverse of the array

# Arrays - Clusters

- 3 Create a VI to find the determinant of a 2x2 matrix which is represented in the form of a 2D array using Index array function

# Arrays - Clusters

- ④ Create a 1D numeric array which consists of ten elements and rotate it ten times. For each rotation display the equivalent binary number of the first array element in the form of a Boolean array. Also display the reversed Boolean array. Provide delay to view the rotation

# Arrays - Clusters

- 1 Create a VI to check whether the cluster elements are in range or not. Specify the upper and lower limits. Display the correct output and a cluster of LEDs to indicate whether a particular cluster element is in the range or not

# Arrays - Clusters

- 2 Create a VI to compare clusters and Switch ON and LED in the output cluster if the  $n^{\text{th}}$  element of cluster 1 is greater than the  $n^{\text{th}}$  element of the cluster 2.



# Arrays - Clusters

- ④ Create a VI to add a value with every element of an available cluster. (Adding a numeric to a cluster results in the addition of the numerica to each element in the cluster)

# Arrays - Clusters

- ④ Create a VI consisting of two clusters of LEDs. Perform the AND operation between the clusters and display the output in another cluster of LEDs. (When comparing clusters, the *AND* function compares each element with its corresponding value in the second cluster)

# Assignment

- 1 Define an array in LabVIEW ?
- 2 What is an array indexing ?
- 3 How are the individual elements accesses and processed in an array ?
- 4 Define auto-indexing
- 5 What is the function of a cluster ?
- 6 Differentiate an array from a cluster.
- 7 What is cluster order ? Explain why it is important
- 8 What is the difference between a *Bundle* and *Bundle By Name* functions ?
- 9 With the help of an example explain assembling and disassembling clusters

# Assignment Questions

- 1 Create a 1D Boolean array and obtain the reverse of the array
- 2 Create a 1D numeric array and check whether the array elements are odd or even. In the output array display 0s and 1s for odd numbers and even numbers respectively.
- 3 Build a VI that generates two 1D arrays and create another array which consists all the elements of the first two arrays.
- 4 Create a VI to read a set of numbers and sort them in ascending order.
- 5 Create a VI to read a set of numbers and find the sum of array elements.
- 6 Create a VI to read a set of numbers upto n, where the programmer defines n and print the contents of the array in reverse order.  
For example : n =4 the input are 26,46,41,123 and the output should be 123,41,46,26

## 1 Unit-4 Plotting Data and Structure

- Objective
- Pre-requisite
- Types of Graphs and Charts
- Types of Structures
- Basic of File I/O Format
- Summary
- Examples/Work-Out
- Assignment Questions

# Table of Contents

- 1 Types of Graphs and Charts
- 2 Customizing Graphs and Charts
- 3 Types of Structures
- 4 Basic of File I/O format.

# Plotting Data and Structure

- **Objective :** The main aim of this chapter to know various types of graphs and charts, structures and file I/O functions.
- **Pre-requisite :**
  - ▶ Good Knowledge of arrays and clusters.
  - ▶ Good Knowledge in For and While Loop
- **Introduction :**
  - ▶ Graphical display of data is an important aspect of programming in LabVIEW.
  - ▶ VIs with graph usually collects the data in an array and then plots the data to the graph to obtain a waveform.
  - ▶ Charts and graphs let you display plots of data in a graphical form.
  - ▶ **Charts interactively plot data, appending new data to old** so that you can see the current value in the context of previous data, as the new data become available.
  - ▶ **Graphs plot pre-generated arrays of values in a more traditional fashion without retaining previously-generated data**

# Plotting Data

- **Waveform Graphs**

- ▶ **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 color to display the values of 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.



# Plotting Data

- **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.
- ▶ 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

# Plotting Data

## ● Waveform Graphs

- 1 **Displaying a Single Plot on Waveform Graphs** : The waveform graph accepts
  - Several data types for single plot waveform graphs
  - A single array of values, interprets the data as points on the graph and increments the x index by one starting at zero.
  - A cluster of an initial x value, a delta x and an array of y data.
  - The waveform data type, which carries the data, start time and delta t of a waveform.
  - The dynamic data type, which is for use with Express VIs.
  - When the dynamic data type includes a single numeric value or single channel, the graph plots the single value and automatically formats the plot legend and x-scale time stamp.
- 2 **Displaying a Multiple Plot on Waveform Graphs** : The waveform graph accepts
  - Several data types for displaying multiple plots
  - A 2D array of values, where each row of the array is a single plot.
  - A cluster of an initial x value, a delta x and a 2D array of y data.
  - A plot array where the array contains clusters. Each cluster contains a 1D array that contains the y data.
  - The inner array describes the points in a plot, and the outer array has one cluster for each plot.

# Plotting Data

## ● 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.
- ▶ The waveform chart maintains a history of data or buffer from previous updates.
- ▶ **Displaying a Single Plot on Waveform Charts :**
  - ❶ If you pass the chart a single value or multiple values at a time, LabVIEW interprets the data as points on the chart and increments the  $x$  index by one starting at  $x = 0$ .
  - ❷ The chart treats these inputs as new data for a single plot. The waveform chart accepts the waveform data type which carries the data, start time and delta  $t$  of a waveform.
- ▶ **Displaying a Multiple Plot on Waveform Charts :**
  - ❶ Waveform charts can display multiple plots together using the *Bundle* function located on the *Cluster* palette.
  - ❷ The *Bundle* function bundles the outputs of the three VIs to plot on the waveform chart.
  - ❸ To pass data for multiple plots to a waveform chart, you can bundle the data together into a cluster of scalar numeric values, where each numeric represents a single point for each of the plots.

# Plotting Data

- XY Graphs

- ▶ The XY graph is a general-purpose, Cartesian graphing object that plots multivalued functions, such as circular shapes or waveforms with a varying time base.
- ▶ It displays any set of points, evenly sampled or not.
- ▶ The XY graph can display plots containing any number of points.
- ▶ It also accepts several data types, which minimizes the extent to which you must manipulate data before you display it.
- ▶ **Displaying a single plot on XY Graphs** :The XY Graphs accepts
  - 1 Three data types of single-plot XY graphs.
  - 2 cluster that contains an x array and a y array
  - 3 an array of points where a point is a cluster that contains an x value and a y value
  - 4 an array of complex data in which the real part is plotted on the x-axis and the imaginary part is plotted on the y-axis

# Plotting Data

- XY Graphs

- ▶ **Displaying a Multiple plot on XY Graphs** :The XY Graphs accepts
  - ① Three data types for displaying multiple plots.
  - ② An array of plots where a plot is a cluster that contains an x array and a y array.
  - ③ An array of clusters of plots where a plot is an array of points.
  - ④ A point is a cluster that contains an x value and a y value.
  - ⑤ An array of clusters of plots where a plot is an array of complex data, in which the real part is plotted on the x-axis and the imaginary part is plotted on the y-axis.

# Plotting Data

- 3D Graphs

- ▶ With the 3D graphs, you can visualize three-dimensional data and alter the way that data appears by modifying the 3D graph properties.
- ▶ LabVIEW includes the following types of 3D graphs :
  - ① 3D surface graph : Draws a surface in 3D space.
  - ② 3D parametric surface graph : Draws a parametric surface in 3D space.
  - ③ 3D curve graph : Draws a line in 3D space
- ▶ Use the 3D graphs in conjunction with the 3D Graph VIs to plot curves and surfaces.
- ▶ A curve contains individual points on the graph, each point having an x, y and z coordinates.
- ▶ The VI then connects these points with a line. A curve is ideal for visualizing the path of a moving object, such as the flight path of an airplane.
- ▶ A surface plot uses x, y and z data to plot points on the graph.
- ▶ The surface plot then connects these points forming a three-dimensional surface view of the data.

# Plotting Data

- Customizing Graphs and Charts

- ▶ Each graph and chart includes many options that you can use to customize appearance, convey more information, or highlight data.
- ▶ Although graphs and charts plot data differently, they have several common options that you access from the shortcut menu.
- ▶ However, some options are available only for a specific type of graph or chart.
  - ① **Using Multiple X and Y Scales** : All graphs support multiple x- and y-scales, and all charts support multiple y-scales. Use multiple scales on a graph or chart to display multiple plots that do not share a common x- or y-scale. Right-click the scale of the graph or chart and select Duplicate Scale from the shortcut menu to add multiple scales to the graph or chart.

# Plotting Data

- Customizing Graphs and Charts

- ② **Autoscaling** : All graphs and charts can automatically adjust their horizontal and vertical scales to fit the data you wire to them. This behavior is called autoscaling. Right-click the graph or chart and select *X Scale»AutoScale X or Y Scale»Auto Scale Y* from the shortcut menu to turn autoscaling ON or OFF. By default, autoscaling is enabled for the graph or chart. However, autoscaling can slow performance.
- ③ **Formatting X- and Y- Scales** :Use the *Format and Precision* page of the Properties dialog box to specify how the scales of the x-axis and y-axis appear on the graph or chart. By default, the x-scale is configured to use floating point notation and have a label of time, and the y-scale is configured to use automatic formatting and have a label of amplitude. To configure the scales for the graph or chart, right-click the graph or chart and select *Properties* from the shortcut menu to display the *Graph Properties* dialog box or *Chart Properties* dialog box



# Plotting Data

- Customizing Graphs and Charts

- ② **Using the Graph Palette** : With the graph palette, you can move cursors, zoom and pan the display. Right-click the graph or chart and select Visible Items»Graph Palette from the shortcut menu to display the graph palette. The graph palette appears with the following buttons, in order from left to right :
  - ① Cursor movement tool (graph only)—Moves the cursor on the display.
  - ② Zoom—Zooms in and out of the display.
  - ③ Panning tool—Picks up the plot and moves it around on the display.

Click a button in the graph palette to enable moving the cursor, zooming the display, or panning the display. Each button displays a green LED when it is enabled.

- ③ **Exporting Images of Graphs, Charts, and Tables** :You can include black and white images of graphs, charts, tables, and digital data and digital waveform controls and indicators into presentations, email, text documents, and so on. When you export a simplified image, LabVIEW exports only the control or indicator, digital display, plot legend, and index display and does not export scrollbars, the scale legend, the graph palette or the cursor palette.

# Plotting Data

- Customizing Graphs and Charts

- **Customizing Graph and Chart Appearance :**

- **Plot legend**—Defines the color and style of plots. Resize the legend to display multiple plots.
- **Scale legend**—Defines labels for scales and configures scale properties.
- **Graph palette**—Allows you to move the cursor and zoom and pan the graph or chart while a VI runs.
- **X scale and Y scale**—Formats the x- and y-scales.
- **Cursor legend (graph only)**—Displays a marker at a defined point coordinate. You can display multiple cursors on a graph.
- **X scrollbar**—Scrolls through the data in the graph or chart. Use the scroll bar to view data that the graph or chart does not currently display.
- **Digital display (waveform chart only)**—Displays the numeric value of the chart.

# Plotting Data

- Customizing Graphs and Charts

- ① **Using Graph Cursors :** Use a cursor on a graph to read the exact value of a point on a plot or a point in the plot area. The cursor value displays in the cursor legend. Right-click the graph and select Visible Items»Cursor Legend from the shortcut menu to view the cursor legend. Add a cursor to the graph by right-clicking anywhere in the cursor legend, selecting Create Cursor, and selecting a cursor mode from the shortcut menu.
- ② **Using Graph Annotations :** Use annotations on a graph to highlight data points in the plot area. The annotation includes a label and an arrow that identifies the annotation and data point. A graph can have any number of annotations. Right-click the graph and select Data Operations»Create Annotation from the shortcut menu to display the Create Annotation dialog box. Use the Create Annotation dialog box to specify the annotation name and how the annotation snaps to plots in the plot area. Use the Lock Style pull-down menu in the Create Annotation dialog box to specify how the annotation snaps to plots in the plot area.

# Structure

- Case structures
- Sequence structure
- Customizing structures
- Timed structures and Formula nodes
- Event structure.
- Creating String control and applications
- Basic of File I/O format

# Structure

- Case 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.

# Structure

- Case Structure

- ▶ 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 **subdiagram**.
- ▶ The terminals that feed data into and out of structures are called **tunnels**.
- ▶ A tunnel is a connection point on a structure border.
- ▶ Use the following structures located on the Structures palette to control how a block diagram executes processes :

# Structure

- **Case Structure**

- ▶ **For Loop**—Executes a subdiagram a set number of times.
- ▶ **While Loop**—Executes a subdiagram until a condition occurs.
- ▶ **Case structure**—Contains multiple subdiagrams, only one of which executes depending on the input value passed to the structure.
- ▶ **Sequence structure**—Contains one or more subdiagrams that execute in sequential order.
- ▶ **Event structure**—Contains one or more subdiagrams that execute depending on how the user interacts with the VI.
- ▶ **Timed Structures**—Execute one or more subdiagrams with time bounds and delays.
- ▶ **Diagram Disable Structure**—Has one or more subdiagrams, or cases, of which only the enabled subdiagram executes.
- ▶ **Conditional Disable Structure**—Has one or more subdiagrams, or cases, exactly one of which LabVIEW uses for the duration of execution, depending on the configuration

# Structure

- **Case Structure** :A case structure executes one subdiagram depending on the input value passed to the structure. Complete the following steps to create a Case structure.
  - ❶ Place a Case structure on the block diagram.
  - ❷ 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.
  - ❸ 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 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.
  - ❹ For each case, use the Labeling tool to enter a single value or lists and ranges of values in 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. If necessary (optional), specify a default case.



# Structure

- A Case structure, shown, has two or more subdiagrams, or cases.
- Only one subdiagram is visible at a time, and the structure executes only one case at a time.
- An input value determines which subdiagram executes.
- The Case structure is similar to switch statements or if...then...else statements in text-based programming languages.
- The case selector label at the top of the Case structure, contains the name of the selector value that corresponds to the case in the center 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 the pull-down menu.
- Wire an input value, or selector, to the selector terminal, shown, to determine which case executes..

# Structure

- You must wire an integer, Boolean value, string, or enumerated type value to the selector terminal
- 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.

# Structure

## 1 Case Selector Values and Data Types

- ▶ You can enter a single value or lists and ranges of values in the case selector label.
- ▶ For lists, use commas to separate values. For numeric ranges, specify a range as 10..20, meaning all numbers from 10 to 20 inclusively.
- ▶ You also can use open-ended ranges. For example, ... 100 represents all numbers less than or equal to 100, and 100.. represents all numbers greater than or equal to 100.
- ▶ When you enter string and enumerated values in a case selector label, the values display in quotation marks, for example "red", "green" and "blue".
- ▶ If you change the data type of the wire connected to the selector terminal of a Case structure, the Case structure automatically converts the case selector values to the new data type when possible.
- ▶ If you wire a floating-point value to the case, LabVIEW rounds the value to the nearest even integer.
- ▶ If you type a floating-point value in the case selector label, the value appears red to indicate that you must delete or edit the value before the structure can execute.

# Structure

## 2 Input and Output Tunnels

- ▶ You can create multiple input and output tunnels for a Case structure. Inputs are available to all cases, but cases do not have to use each input. However, you must define each output tunnel for each case.
- ▶ When you create an output tunnel in one case, tunnels appear at the same position on the border in all the other cases.
- ▶ If even one output tunnel is not wired, all output tunnels on the structure appear as white squares.

## 3 Using Case Structures for Error Handling

- ▶ When you wire an error cluster to the selector terminal of a Case structure, the case selector label displays two cases—Error and No Error—and the border of the Case structure changes color—red for Error and green for No Error.
- ▶ If an error occurs, the Case structure executes the Error subdiagram.

# Structure

- Sequence Structure

- ▶ 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.
- ▶ 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.

# Structure

- Sequence Structure

- ① Flat Sequence Structure

- The Flat Sequence structure, 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.
    - When you add or delete frames in a Flat Sequence structure, the structure resizes automatically.

- ② Stacked Sequence Structure

- The Stacked Sequence structure, 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 Flat Sequence structure, right-click the Stacked Sequence structure and select Replace»Replace with Flat Sequence from the shortcut menu.
    - To pass data from one frame to any subsequent frame of a Stacked Sequence structure, use a sequence local terminal shown

# Structure

- Customizing Structure

- ① Placing Structures on the Block Diagram

- Select a structure on the Structures palette. The cursor becomes a small icon of the structure.
    - Click the block diagram where you want to place the top corner of the structure and move the cursor down and to the right or left.
    - Click the block diagram again when the structure is the size you want.

- ② Placing Objects inside Structures

- Place the Case structure on the block diagram.
    - Place the Tick Count (ms) function inside the structure.
    - Move the Tick Count (ms) function close to the border of the structure. Notice that when you place or move an object in a structure near the structure border, the structure resizes to add space for that object. To disable the automatic resizing behavior for a structure, right-click the structure border and select Auto Grow from the shortcut menu to remove the checkmark.
    - Move the Tick Count (ms) function outside the structure.
    - Place another structure around the Tick Count (ms) function.

# Structure

- Select the second structure and delete it. Notice that you also deleted the function inside the structure.
- Select the Case structure and delete it. Notice that you did not delete the function because it was not inside the structure
- ③ **Removing Structures without Deleting Objects in the Structure**
  - ▶ Right-click the structure you want to remove.
  - ▶ Select Remove For Loop from the shortcut menu or the similar option for any of the other structures
- ④ **Resizing Structures**
  - ▶ Move the positioning tool over the structure border. Resizing handles appear at the corners of the structure and in the middle of each structure border.
  - ▶ Move the cursor over a resizing handle to change the cursor to the resizing cursor.
  - ▶ Use the resizing cursor to drag the resizing handles until the dashed border outlines the size you want.
  - ▶ Release the mouse button. The structure reappears in its new size



# Structure

- Timed Structures

- ① Use timed structures on the block diagram to repeat blocks of code and to execute code in a specific order with time bounds and delays.
- ② Each timed structure has a distinctive, resizable border to enclose a section of the block diagram that executes according to the rules of the structure.
- ③ The section of the block diagram inside the structure border is called a subdiagram.
- ④ A timed structure has Input and Output nodes that feed data into and out of the structure to provide configuration data and return error and timing information.
- ⑤ Timed structures can also have terminals on the structure border that feed data into and out of the structure subdiagrams.
  - **Timed Loop**—Executes a subdiagram until a condition is met or interminably.
  - **Timed Sequence**—Executes multiple subdiagrams in sequence.
  - **Timed Loop with Frames**—Executes multiple subdiagrams in sequence until a condition is met or interminably. Add frames to a Timed Loop to create a Timed Loop with frames

# Structure

- **Formula Nodes**

- ▶ The Formula Node is a convenient text-based node you can use to perform mathematical operations on the block diagram.
- ▶ 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.
- ▶ 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.

# Structure

- **Formula Nodes**

- ▶ 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 input variable 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.
  - 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.

# Structure

- Formula Nodes

- ▶ Creating Formula Nodes

- Place a Formula Node on the block diagram.
- Review the available functions and operators you can use.
- Use the labeling tool or the operating tool to enter the equations you want to calculate inside the Formula Node. Each assignment must have only a single variable on the left side of the assignment (=). Each assignment must end with a semicolon (;). Confirm that you are using the correct Formula Node syntax.
- If a syntax error occurs, click the broken Run button to display the Error list window. LabVIEW marks the syntax error with a symbol.
- 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 that appears. You can edit the variable name at any time using the labeling tool or the Operating tool, except when the VI is running.
- Variable terminals are case sensitive. There is no limit to the number of terminals or equations in a Formula Node. You can change a terminal type or remove a terminal.
- Create an output terminal for each output variable by right-clicking the Formula Node border and selecting Add Output from the shortcut menu. Type the variable name in the terminal that appears. You can edit the variable name at any time using the labeling tool or the operating tool, except when the VI is running. Output variables have thicker borders than input variables.

# File I/O Format

- A typical file I/O operation involves the following process,
  - ❶ Create or open a file. Indicate where an existing file resides or where you want to create a new file by specifying a path or responding to a dialog box to direct LabVIEW to the file location. After the file opens, a refnum represents the file.
  - ❷ Read from or write to the file.
  - ❸ Close the file.
- File I/O VIs and some File I/O functions, such as the Read from Text File and Write to Text File functions, can perform all three steps for common file I/O operations.
- The VIs and functions designed for multiple operations might not be as efficient as the functions configured or designed for individual operations.
- Many File I/O VIs and functions contain flow-through parameters, typically a refnum or path, which return the same value as the corresponding input parameter.

# Summary

- A Case structure has two or more subdiagrams, or cases. Only one subdiagram is visible at a time, and the structure executes only one case at a time.
- If the case selector terminal is a Boolean value, 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 up to  $2^{31} - 1$  cases.
- Inputs are available to all subdiagrams of a Case structure, but subdiagrams do not need to use each input. If output tunnel is not defined, in all cases it appears as white square.
- When creating a subVI from a Case structure, wire the error input to the selector terminal, and place all subVI codes within the No Error case to prevent the subVI from executing if it receives an error.
- Timed Loop—Executes a subdiagram until a condition is met or interminably.
- Timed Sequence—Executes multiple subdiagrams in sequence.
- Timed Loop with Frames—Executes multiple subdiagrams in sequence until a condition is met or interminably. Add frames to a Timed Loop to create a Timed Loop with frames.
- Formula Nodes are useful for equations that have many variables. Each equation statement must terminate with a semicolon (;)

# Summary

- The waveform chart is a special numeric indicator that displays one or more plots.
- The waveform chart has the following three update modes :
  - ❶ A strip chart shows running data continuously scrolling from left to right across the chart.
  - ❷ A scope chart shows one item of data, such as a pulse or wave, scrolling partway across the chart from left to the right.
  - ❸ A sweep display is similar to an EKG display. A sweep works similarly to a scope except it shows the old data on the right and the new data on the left separated by a vertical line.
- Waveform graphs and XY graphs display data from arrays.
- Right-click a waveform chart or graph or its components to set attributes of the chart and its plots.
- You can display more than one plot on a graph using the Build Array function located on the Functions»All Functions»Array palette and the Bundle function located on the Functions»All Functions»Cluster palette for charts and XY graphs. The graph becomes a multiplot graph when you wire the array of outputs to the terminal.

# Summary

- Use the File I/O VIs and functions located on the Functions»File I/O palette to handle all aspects of file I/O.
- When writing to a file, you open, create, or replace a file, write the data and close the file. Similarly, when you read from a file, you open an existing file, read the data and close the file.
- To access a file through a dialog box, do not wire file path in the Open/Create/Replace File VI.
- To write data to a spreadsheet file, you must format the string as a spreadsheet string, which is a string that includes delimiters, such as tabs. Use the Format Into File function to format string, numeric, path and Boolean data as text and write the text to a file.



## 1 Unit-5 Data Acquisition

- Objective
- Pre-requisite
- Introduction
- DAQ hardware
- Analog, Digital Inputs and Outputs
- DAQ software Architecture
- DAQ Assistant
- Selecting and Configuring a Data Acquisition device
- Summary
- Examples/Work-Out
- Assignment Questions

# Table of Contents

- 1 Introduction to Analog and Digital Signals
- 2 DAQ hardware
- 3 Analog and Digital Inputs and Outputs
- 4 DAQ Software Architecture
- 5 DAQ Assistant
- 6 Selecting and Configuring a Data Acquisition device
- 7 Case Study

# Data Acquisition

- **Objective** : The main aim of this chapter to know about methods of acquiring digital and analog signals, configure the data acquisition devices.
- **Pre-requisite** :
  - ▶ Basic LabVIEW Environment
  - ▶ Loops and Arrays
  - ▶ Graphs and Charts
- **Introduction to Data Acquisition**
  - ▶ The fundamental task of a DAQ (Data Acquisition) system is to measure or generate real-world physical signals.
  - ▶ Data acquisition involves gathering signals from measurement sources and digitizing the signal for storage, analysis and presentation on a personal computer (PC).
  - ▶ The five components to be considered when building a basic DAQ system are
    - Transducers
    - Signals
    - Signal Conditioning
    - DAQ Hardware
    - Driver and application software.

# Data Acquisition

- **Transducers**

- ▶ A transducer is a device that converts a physical phenomenon into a measurable electrical signal, such as voltage or current.
- ▶ The ability of a DAQ system to measure different phenomena depends on the transducers to convert the physical phenomena into signals measurable by the DAQ hardware.
- ▶ Transducers are synonymous with sensors in DAQ systems.
- ▶ There are specific transducers for many different applications, such as measuring temperature, pressure or fluid flow.
- ▶ Different transducers have different requirements for converting phenomena into a measurable signal.
- ▶ Some transducers may require excitation in the form of voltage or current.
- ▶ Other transducers may require additional components and even resistive networks to produce a signal.

# Data Acquisition

- Signals

- ▶ The appropriate transducer converts the physical phenomena into measurable signals. Signals can be categorized into two groups. **Analog and Digital** signals.
- ▶ **Analog signals**
  - An analog signal can be at any value with respect to time. A few examples of analog signals include Voltage, Temperature, Pressure, Sound and Load.
  - The three primary characteristics of an analog signal include level, shape and frequency.
  - **Level** gives vital information about the measured analog signal since analog signals can take on any value. The intensity of a light source, the temperature in a room, and the pressure inside a chamber are all examples that demonstrate the importance of the level of a signal.
  - Some signals are named after their specific **shape**—sine, square, sawtooth and triangle. The shape of an analog signal can be as important as the level, because measuring the shape of an analog signal allows further analysis of the signal, including peak values, DC values and slope.
  - Unlike the level or shape of the signal, **frequency** cannot be directly measured. The signal must be analyzed using software to determine the frequency information. This analysis is usually done using an algorithm known as the **Fourier transform**.

# Data Acquisition

- Signals

- ▶ Digital Signals

- A digital signal cannot take on any value with respect to time. Instead, a digital signal has two possible levels : **High and Low**.
- Digital signals generally conform to certain specifications that define the characteristics of the signal.
- The state of a digital signal is essentially the level of the signal—on or off, high or low.
- Monitoring the state of a switch—open or closed—is a common application showing the importance of knowing the state of a digital signal.
- The rate of a digital signal defines how the digital signal changes state with respect to time.

# Data Acquisition

## ● Signal Conditioning

- ▶ Signal conditioning is the process of measuring and manipulating signals to improve accuracy, isolation, filtering, and so on.
- ▶ Many stand-alone instruments and DAQ devices have built-in signal conditioning.
- ▶ Signal conditioning also can be applied externally, by designing a circuit to condition the signal or by using devices specifically made for signal conditioning.
- ▶ Signal conditioning accessories amplify low-level signals and then isolate and filter them for more accurate measurements.
- ▶ Common types of signal conditioning are amplification, isolation, multiplexing, filtering, transducer excitation and linearization.
- ▶ Signal conditioning maximizes the accuracy of a system, allows sensors to operate properly and guarantees safety.
- ▶ Signal conditioning accessories can be used in a variety of applications including amplification, attenuation, isolation, bridge completion, simultaneous sampling, sensor excitation, multiplexing, etc.

# Data Acquisition

- Signal Conditioning

- ▶ Amplification

- Amplification is the most common type of signal conditioning.
    - Amplifying electrical signals improves accuracy in the resulting digitized signal and reduces the effects of noise.
    - By amplifying a signal near the device, any noise that attached to the signal is also amplified.
    - Amplifying near the signal source results in the largest signal-to-noise ratio (SNR).

- ▶ Isolation

- Another common signal conditioning application is isolating the transducer signals from the computer for safety purposes.
    - The system being monitored may contain high-voltage transients that could damage the computer without signal conditioning.
    - An additional reason for isolation is ensuring that the readings from the plug-in DAQ device are unaffected by differences in ground potentials or common-mode voltages.
    - When the DAQ device input and the signal being acquired are each referenced to 'ground', problems occur if there is a potential difference in the two grounds.



# Data Acquisition

- Signal Conditioning

- ▶ Multiplexing

- A common technique for measuring several signals with a single measuring device is multiplexing.
- Signal conditioning hardware for analog signals often provides multiplexing for use with slowly changing signals like temperature.
- The ADC samples one channel, switches to the next channel, samples it, switches to the next channel, and so on.

- ▶ Filtering

- The purpose of a filter is to remove unwanted signals from the signal that you are trying to measure.
- A noise filter is used on DC-class signals, such as temperature, to attenuate higher frequency signals that can reduce your measurement accuracy.
- AC-class signals, such as vibration, often require a different type of filter known as an **antialiasing filter**.
- Like the noise filter, the antialiasing filter is also a **lowpass filter**; however, it requires a very steep cutoff rate, so it almost completely removes all signal frequencies that are higher than the input bandwidth of the device.

# Data Acquisition

- Signal Conditioning

- ▶ Transducer Excitation

- Signal conditioning systems can generate excitation, which some transducers require for operation.
- Strain gauges and RTDs require external voltage and currents, respectively, to excite their circuitry into measuring physical phenomena.
- This type of excitation is similar to a radio that needs power to receive and decode audio signals.
- Signal conditioning modules for these transducers usually provide these signals.

- ▶ Linearization

- Another common signal conditioning function is linearization. Many transducers, such as thermocouples, have a nonlinear response to changes in the physical phenomena you measure.
- LabVIEW can linearize the voltage levels from transducers so you can scale the voltages to the measured phenomena.
- LabVIEW provides scaling functions to convert voltages from strain gauges, RTDs, thermocouples, and thermistors.

# Data Acquisition

- **Selecting and Configuring a Data Acquisition Device**

- ▶ Depending on the application needs, you must determine the minimum number of analog input channels, analog output channels, and digital I/O lines that your data acquisition board requires.
- ▶ Other important factors to consider are the sampling rate, the input range, the input mode and the accuracy.
- ▶ When you connect any electrical signal to your data acquisition device, you expect your readings to match the electrical value of the input signal.
  - **Signal Sources** :Analog input acquisitions use grounded and floating signal sources
  - Signal sources can be grounded or ungrounded. The grounded signal sources are signals referenced to a system ground like earth ground and building ground. Because such sources use the system ground, they share a common ground with the measurement device.
  - The grounds of two independently grounded signal sources generally are not at the same potential.
  - The difference in ground potential between two instruments connected to the same building ground system is typically from 10 mV to 200 mV.

# Data Acquisition

- **Selecting and Configuring a Data Acquisition Device**

- ▶ **Measurement Systems** :You configure a measurement system based on the hardware you use and the measurement you take. Three modes of grounding for your Measurement System are Differential, Referenced Single-Ended (RSE) and Non-Referenced Single-Ended (NRSE).
- ▶ Differential measurement systems are similar to floating signal sources in that you make the measurement with respect to a floating ground that is different from the measurement system ground.
- ▶ Referenced and non-referenced single-ended measurement systems are similar to grounded sources in that you make the measurement with respect to a ground. A referenced single-ended measurement system measures voltage with respect to the ground, AIGND (analog input ground), which is directly connected to the measurement system ground

# Data Acquisition

- **Selecting and Configuring a Data Acquisition Device**

- ▶ **Increasing Measurement Quality** :When you design a measurement system, you may find that the measurement quality does not meet your expectations. You might want to record the smallest possible change in a voltage level.
- ▶ The following reasons affect achieving the smallest detectable change in voltage :
  - ① The resolution and range of the ADC.
  - ② The gain applied by the instrumentation amplifier.
  - ③ The combination of the resolution, range and gain to calculate a property called the code width value.
- ▶ **Resolution** is important. The number of bits used to represent an analog signal determines the resolution of the ADC.
- ▶ The resolution on a DAQ device is similar to the marks on a ruler.
- ▶ The more marks a ruler has, the more precise the measurements are.
- ▶ The higher the resolution is on a DAQ device, the higher the number of divisions into which a system can break down the ADC range, and therefore, the smaller the detectable change.

# Data Acquisition

- **Selecting and Configuring a Data Acquisition Device**
  - ▶ **Range** refers to the minimum and maximum analog signal levels that the ADC can digitize.
  - ▶ Many DAQ devices feature selectable ranges (typically 0 to 10 V or -10 to 10 V), so you can match the ADC range to that of the signal to take best advantage of the available resolution to accurately measure the signal.
  - ▶ **Amplification or Attenuation** of a signal can occur before the signal is digitized to improve the representation of the signal. By amplifying or attenuating a signal, you can effectively decrease the input range of an ADC and thus allow the ADC to use as many of the available digital divisions as possible to represent the signal.
  - ▶ **The range, resolution, and amplification** available on a DAQ device determine the smallest detectable change in the input voltage
  - ▶ **Code width** is the smallest change in a signal that a system can detect.

# Data Acquisition

- **DAQ Assistant**

- ▶ The DAQ Assistant is a graphical interface for interactively creating, editing, and running NIDAQmx virtual channels and tasks.
- ▶ A NI-DAQmx task is a collection of virtual channels, timing and triggering information, and other properties regarding the acquisition or generation.
- ▶ DAQ Assistant provides an interactive guide to configuring, testing and acquiring measurement data.
- ▶ DAQ Assistant is completely menu-driven and you will encounter fewer errors. It drastically decreases the time to your first measurement

# Data Acquisition

- DAQ Assistant

- ▶ Launch the DAQ Assistant

- ➊ Open LabVIEW and create a New VI. Switch to the block diagram (Ctrl+E).
- ➋ DAQ Assistant Express VI is located in the Input subpalette of the Functions palette. Place the DAQ Assistant on the block diagram by dragging and dropping it from the Functions palette. The Assistant should automatically launch when you drop the VI on the diagram.
- ➌ It is also available at Express»Output»DAQ Assistant. In the Advanced Functions palette, the DAQ Assistant Express VI is located in the NI Measurements » DAQmx sub-palette. The Create New window opens up for task configuration when the DAQ Assistant is placed on the block diagram. Measurement type can be Analog Input, Analog Output, Counter Input, Counter Output and Digital I/O.
- ➍ Once you have located the DAQ Assistant Express VI in the appropriate location, select it from the palette and drop it on the block diagram of your VI. By default, the properties page should pop up, allowing you to configure your task. The first step is to select your type of measurement.



# Data Acquisition

- DAQ Assistant

- ▶ Create a Task

- ① On the first screen, select Analog Input for your Measurement Type.
- ② Next, select Voltage.
- ③ The next screen lets you select the physical channel (or channels) for which you are creating this task. All supported data acquisition hardware devices should appear in the tree control and you can expand them to view a list of the physical channels that you can select for your task. To select more than one channel, hold down the Ctrl button while clicking on the channel names.
- ④ Click Finish to move on to the configuration stage

# Data Acquisition

- DAQ Assistant

- ▶ Configure a Task

After you create a task, you can configure channel-specific settings such as scaling, input limits and terminal configuration. You also can configure task-specific settings such as timing and triggering.

- 1 Specify the input limits. You can use the default values of 5 for Max and -5 for Min if you do not know the theoretical limits for the signal you are measuring.
- 2 Select the terminal configuration you used for the signal.
- 3 On the Task Timing tab, select Acquire N Samples. Enter 100 for Samples To Read and -enter 1000.00 for Rate (Hz).

- ▶ Test the task

- 1 Launch the test panel for your task by clicking the Test button at the top of the screen.
- 2 The test runs once automatically. Click the Start button to run the test again. Notice that the graph displays the acquired signal.
- 3 Click the Close button when you are done. If necessary, modify the settings for the task and retest the task.
- 4 After the test panel closes, click the OK button. The DAQ Assistant saves the voltage task, containing all the configuration information you entered, to MAX. You have created your voltage task.

# Data Acquisition

- **DAQ Hardware**

- ▶ The DAQ hardware acts as the interface between the computer and the outside world. It primarily functions as a device that digitizes incoming analog signals so that the computer can interpret them.
- ▶ Other data acquisition functionality includes Analog Input/Output, Digital Input/Output, Counter/Timers and Multifunction which is a combination of analog, digital, and counter operations on a single device.
- ▶ A typical desktop DAQ system has three basic types of hardware **a terminal block, a cable and a DAQ device**
- ▶ After you have converted a physical phenomenon into a measurable signal with or without signal conditioning, you need to acquire that signal.
- ▶ To acquire a signal, you need a terminal block, a cable, a DAQ device and a computer.
- ▶ This hardware combination can transform a standard computer into a measurement and automation system.

# Data Acquisition

- DAQ Hardware

- ▶ Terminal Block and Cable

- A terminal block provides a place to connect signals. It consists of screw or spring terminals for connecting signals and a connector for attaching a cable to connect the terminal block to a DAQ device.
- The type of terminal block you should choose depends on two factors—the device and the number of signals you are measuring.
- A terminal block with 68 terminals offers more ground terminals to connect a signal to than a terminal block with 50 terminals.
- Having more ground terminals prevents the need to overlap wires to reach a ground terminal, which can cause interference between the signals.

# Data Acquisition

- DAQ Hardware

- ▶ DAQ Signal Accessory

- The DAQ Signal Accessory is a customized terminal block designed for learning purposes.
    - It has 3 Connectors, Quadrature Encoder, Relay, Digital Trigger, 4 LEDs (reverse logic), Counter I/O, Function Generator, Function Generator Frequency Control, Temperature Sensor, Temperature Sensor Noise Control, Analog Input and Analog Output.
    - The three different cable connectors accommodate many different DAQ devices and spring terminals to connect signals.
    - You can access three analog input channels, one of which is connected to the temperature sensor and two analog output channels.
    - The DAQ Signal Accessory includes a function generator with a switch to select the frequency range of the signal, and a frequency knob.
    - The function generator can produce a sine wave or a square wave. A connection to ground is located between the sine wave and square wave terminal.
    - The DAQ Signal Accessory also has a relay, a thermocouple input and a microphone jack.

# Data Acquisition

- DAQ Hardware

- ▶ DAQ Device

- Before a computer-based measurement system can measure a physical signal such as temperature, a sensor or transducer, must convert the physical or real world signal into an electrical one such as voltage or current.
- You must use signal conditioning accessories to condition the signals before the plug-in DAQ device converts them to digital information.
- The computer receives raw data through the DAQ device. The application you write presents and manipulates the raw data in a form you can understand.
- The software also controls the DAQ system by commanding the DAQ device when and from which channels to acquire data.
- Typically, DAQ software includes drivers and application software.
- Most DAQ devices have four standard elements : analog input, analog output, digital I/O, and counters.
- You can transfer the signal you measure with the DAQ device to the computer through a variety of different bus structures.

# Data Acquisition

- DAQ Hardware

- ▶ DAQ Device

- Analog Input Circuitry has a multiplexer (mux). This switch has multiple input channels but only lets one at a time through to the instrumentation amplifier.
    - The instrumentation amplifier either amplifies or attenuates your signal.
    - Analog-to-Digital Converter (ADC) converts an analog signal to a digital number and is used for analog input.
    - The applications are circuit testing, power supply testing, dynamometer testing, weather station, geophysical studies and filter analysis.
    - Digital-to-Analog Converter (DAC) converts a digital number to an analog signal and is used for analog output. The applications are control systems, function generator, tone generator and servo motor control.
    - Depending on your application, there are several different classes of PC-based data acquisition devices that you can use :
    - Analog Input/Output
    - Digital Input/Output
    - Counter/Timers
    - Multifunction—a combination of analog, digital and counter operations