This book is not a manual-like presentation of LabVIEW, but rather leads its readers to mastery of this powerful laboratory tool through the process of carrying out interesting and relevant projects. Readers--who are assumed to have no prior computer programming or LabVIEW background--will begin writing meaningful programs within the first few pages. Hands-On Introduction to LabVIEW is designed for flexible use so that readers can easily choose the desired depth of coverage.
This book can be useful to anyone who wants to start working with the LabVIEW program, and this program can be used in laboratories such as measurements, control, electron, etc.

Elaf A.Saeed is a systems and control engineer at the University of Al-Nahrain, college of information engineering, Iraq. she works in the fields of control, embedded systems, web design and has a talent of programming. Elaf was ranking as the first student of her stage for all years in B.Sc .

978-620-2-51970-0

Basics LabVIEW Lab



Elaf A.Saeed

# Basics LabVIEW Lab

## Basics Book for Learning LabVIEW with Experiments

LAP LAMBERT Academic Publishing

A.Saeed

**Elaf A.Saeed**

**Basics LabVIEW Lab**

Elaf A.Saeed

# Basics LabVIEW Lab

## Basics Book for Learning LabVIEW with Experiments



LAP LAMBERT Academic Publishing

**Imprint**
Any brand names and product names mentioned in this book are subject to trademark, brand or patent protection and are trademarks or registered trademarks of their respective holders. The use of brand names, product names, common names, trade names, product descriptions etc. even without a particular marking in this work is in no way to be construed to mean that such names may be regarded as unrestricted in respect of trademark and brand protection legislation and could thus be used by anyone.

Cover image: www.ingimage.com

Publisher:
LAP LAMBERT Academic Publishing
is a trademark of
International Book Market Service Ltd., member of OmniScriptum Publishing Group
17 Meldrum Street, Beau Bassin 71504, Mauritius

Printed at: see last page
**ISBN: 978-620-2-51970-0**

*To my parents and to our family who made this accomplished possible*

TABLE OF CONTENTS

# Contents

# TABLE OF CONTENTS

# CHAPTER ONE
# LABVIEW INTRODUCTION

## List of Abbreviation

| Abbreviation | Meaning |
|---|---|
| LabVIEW | Laboratory Virtual Instrument Engineering Workbench |
| VI | Virtual Instrument |
| MAX | Measurement and Automation eXplorer |
| VISA | Virtual Instrument Software Architecture |
| IDNet | Instrument Driver Network |
| ANSI | American National Standards Institute |
| IEEE | Institute of Electrical and Electronics Engineers |
| ISO | International Organization for Standardization. |
| DMM | Digital Multimeter |
| UI | User Interface |

# CHAPTER ONE
# LABVIEW INTRODUCTION

## Introduction to Book

**LabVIEW** programs are called **virtual instruments**, or **VIs**, because their appearance and operation imitate physical instruments, such as **oscilloscopes** and **multimeters**. **LabVIEW** contains a comprehensive set of tools for **acquiring**, **analyzing**, **displaying**, and **storing data**, as well as tools to help you troubleshoot code you write. In **LabVIEW**, you build a user interface, or front panel, with controls and indicators. **Controls** are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays. After you build the **front panel**, you add code using VIs and structures to control the front panel objects. The block diagram contains this code.

You can use **LabVIEW** to communicate with hardware such as **data acquisition**, **vision**, and **motion control devices**, as well as GPIB, PXI, VXI, RS232, and RS485 instruments. This book rather leads its readers to mastery of this powerful laboratory tool through the process of carrying out interesting and relevant projects. Readers--who are assumed to have no prior computer programming or LabVIEW background--will begin writing meaningful programs within the first few pages. Hands-On Introduction to LabVIEW is designed for flexible use so that readers can easily choose the desired depth of coverage.

Use this book as a tutorial to familiarize yourself with the **LabVIEW** graphical programming environment and the basic **LabVIEW** feature you use to build data acquisition and instrument control applications. This book contains exercises that you can use to learn how to develop basic applications in **LabVIEW**. These exercises take a short amount of time to complete and help you get started with LabVIEW.

# CHAPTER ONE
# LABVIEW INTRODUCTION

## Introduction

**Laboratory Virtual Instrument Engineering Workbench (LabVIEW)** is a system-design platform and development environment for a visual programming language from National Instruments.

**LabVIEW** is a very large **virtual laboratory** program and an integrated development environment based on **graphical programming**, through the use of forms familiar to all engineers and technicians. Where LabVIEW program relies on **graphical symbols** instead of using a specific programming language, where in scripting the idea is based on the scribes of a text program either in **graphical programming**, it is replaced by a set of tools and components, which is known **dataflow programming**. Of course, it supports programming languages such as **C** and **HTML** in advanced tasks. What distinguishes it is the ease of use through simple procedures for configuring the graphical user interface and linking these tools by specifying properties and functions to perform the required tasks. **LabVIEW** also includes a wizard that creates various applications by following a series of instructions and options that lead to the creation of the application. It also has a very important feature, which is creating executables for the applications you have created. It is mainly integrated to communicate with devices such as **GPIP**, **VXI, RS-485**, **RS-232** and others. Ports can be used in all its forms to connect to the surrounding, be it parallel, USB or other ports. It contains very large internal public libraries, for use in software standards such as transmission control systems, the connection of **TCP / IP** networks, **ACTIVE X** and others. Supports **data acquisition** by providing very powerful tools, **image processing** and also **IMAQ.** The program comes within a **package of programs** that are used for a variety of purposes in the same work environment of the LabVIEW program, which creates amazing integration, and the tasks of these programs vary from processors that link devices that we want to communicate with applications to

monitor these devices and record and record their errors. In figure 1.1 that shown the LabVIEW symbols.

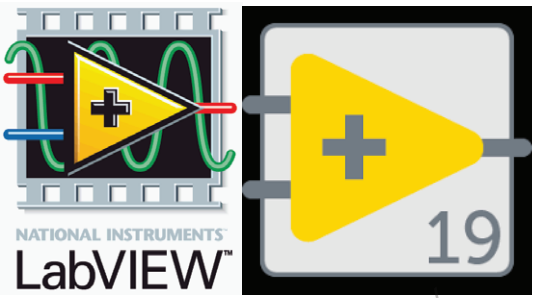



**Figure 1.1: LabVIEW Symbols**



## Dataflow programming

The programming paradigm used in **LabVIEW**, sometimes called **G**, is based on **data availability**. If there is enough data available to a subVI or function, that subVI or function will execute. Execution flow is determined by the structure of a graphical block diagram (the LabVIEW-source code) on which the programmer connects different function-nodes by drawing wires. These wires propagate variables and any node can execute as soon as all its input data become available. Since this might be the case for multiple nodes simultaneously, LabVIEW can execute inherently in parallel. **Multi-processing** and **multi-threading** hardware is exploited automatically by the built-in scheduler, which **multiplexes** multiple OS threads over the nodes ready for execution.

## Graphical programming

**LabVIEW** integrates the creation of **user interfaces** (termed **front panels**) into the development cycle. **LabVIEW** programs-subroutines are termed **virtual instruments (VIs)**. Each VI has three components: a block diagram, a front panel, and a connector

pane. The last is used to represent the VI in the block diagrams of other, calling VIs. The front panel is built using controls and indicators. Controls are inputs: they allow a user to supply information to the VI. Indicators are outputs: they indicate, or display, the results based on the inputs given to the VI. The back panel, which is a block diagram, contains the graphical source code. All of the objects placed on the front panel will appear on the back panel as terminals. The back panel also contains structures and functions which perform operations on controls and supply data to indicators. The structures and functions are found on the Functions palette and can be placed on the back panel. Collectively **controls**, **indicators**, **structures,** and **functions** are referred to as **nodes**. **Nodes** are connected to one another using **wires**, e.g., two controls and an indicator can be wired to the addition function so that the indicator displays the sum of the two controls. Thus, a virtual instrument can be run as either a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the node through the connector pane. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.



The **graphical approach** also allows **nonprogrammers** to build programs by dragging and dropping virtual representations of lab equipment with which they are already familiar. The LabVIEW programming environment, with the included examples and documentation, makes it simple to create small applications. This is a benefit on one side, but there is also a certain danger of underestimating the expertise needed for high-quality G programming. For complex algorithms or large-scale code, it is important that a programmer possess an extensive knowledge of the special LabVIEW syntax and the topology of its memory management. The most advanced LabVIEW development systems offer the ability to build stand-alone applications. Furthermore, it is possible to create distributed applications, which communicate by a **client–server model**, and are

thus easier to implement due to the inherently parallel nature of G. In figure 1.2 that shown the LabVIEW Front Panel and Block Diagram.

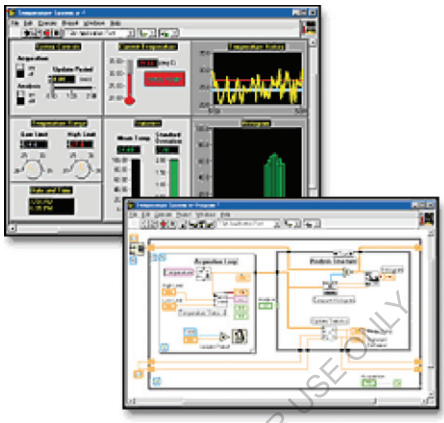


**Figure 1.2: LabVIEW Front Panel and Block Diagram**



## What basic functions can LabVIEW perform?

LabVIEW can be used to perform a huge number of mathematical and logic functions, including, but certainly not limited to: basic **arithmetic**, **if/then/elseif conditional statements**, **case statements**, **FFTs**, **filtering, PID control loops**, etc. There are huge libraries of functions to pull from. You can also interface to code developed in other languages, through DLLs. .NET assemblies, and run-time interpreters (e.g., MATLAB), for example.

Another somewhat unique capability that LabVIEW offers is **real-time compilation** and the ability to execute function blocks without requiring development of a test case. Each LabVIEW function is designed with a user interface so you can interact with your code immediately after you write it.

# CHAPTER ONE
# LABVIEW INTRODUCTION

## LabVIEW Benefits

### Interfacing to devices

**LabVIEW** includes extensive support for interfacing to devices, instruments, camera, and other devices. Users interface to hardware by either writing direct bus commands (**USB**, **GPIB**, **Serial**) or using high-level, device-specific, drivers that provide native LabVIEW function nodes for controlling the device.

**LabVIEW** includes built-in support for **NI** hardware platforms such as **CompactDAQ** and **CompactRIO**, with a large number of device-specific blocks for such hardware, the **Measurement and Automation eXplorer (MAX)** and **Virtual Instrument Software Architecture (VISA)** toolsets.

**National Instruments** makes **thousands** of device drivers available for download on the NI **Instrument Driver Network (IDNet)**.



### Code compiling

LabVIEW includes a compiler that produces native code for the CPU platform. The graphical code is converted into Dataflow Intermediate Representation, and then translated into chunks of executable machine code by a compiler based on **LLVM**. Run-time engine calls these chunks, allowing better performance. The **LabVIEW** syntax is strictly enforced during the editing process and compiled into the executable machine code when requested to run or upon saving. In the latter case, the executable and the source code are merged into a single binary file. The execution is controlled by **LabVIEW run-time** engine, which contains some pre-compiled code to perform common tasks that are defined by the **G language**. The run-time engine governs execution flow, and provides a consistent interface to various operating systems, graphic systems and hardware components. The use of run-time environment makes the source code files portable across supported platforms. LabVIEW programs are slower than equivalent compiled C code,

though like in other languages, program optimization often allows to mitigate issues with execution speed.

### Large libraries

Many libraries with a large number of functions for **data acquisition**, **signal generation**, **mathematics**, **statistics**, **signal conditioning**, **analysis**, etc., along with numerous for functions such as integration, filters, and other specialized abilities usually associated with **data capture** from hardware sensors is enormous. In addition, **LabVIEW** includes a text-based programming component named **MathScript** with added functions for signal processing, analysis, and mathematics. **MathScript** can be integrated with graphical programming using **script nodes** and uses a syntax that is compatible generally with **MATLAB**.



### Parallel programming

**LabVIEW** is an inherently **concurrent language**, so it is very easy to program multiple tasks that are performed in **parallel via multithreading**. For example, this is done easily by drawing two or more parallel while loops and connecting them to two separate nodes. This is a great benefit for **test system automation**, where it is common practice to run processes like **test sequencing**, **data recording**, and **hardware interfacing** in parallel.

### Ecosystem

Due to the longevity and popularity of the **LabVIEW** language, and the ability for users to extend its functions, a large **ecosystem** of third-party add-ons has developed via contributions from the community. This **ecosystem** is available on the **LabVIEW** Tools Network, which is a marketplace for both free and paid LabVIEW add-ons.

### User community

There is a low-cost **LabVIEW** Student Edition aimed at educational institutions for learning purposes. There is also an active community of **LabVIEW** users who

communicate through several **electronic mailing lists** (email groups) and **Internet forums**.

### Home Bundle Edition

**National Instruments** provides a **low-cost LabVIEW Home Bundle Edition**.

## LabVIEW Criticism

**LabVIEW** is a **proprietary** product of **National Instruments**. Unlike common programming languages such as **C** or Fortran, LabVIEW is not managed or specified by a third party standards committee such as **American National Standards Institute (ANSI)**, **Institute of Electrical and Electronics Engineers (IEEE)**, **International Organization for Standardization (ISO)**, etc. Many users have criticized it for its tendency to freeze or crash during simple tasks, often requiring the software to be shut down and restarted.



### Slow

Very small applications still have to start the run-time environment which is a large and slow task. This tends to restrict **LabVIEW** to monolithic applications. Examples of this might be tiny programs to grab a single value from some hardware that can be used in a scripting language - the overhead of the run-time environment render this approach impractical with **LabVIEW**.

### Non-textual

**G language** being non-textual, software tools such as versioning, side-by-side (or diff) comparison, and version code change tracking cannot be applied in the same manner as for textual programming languages. There are some additional tools to make comparison and merging of code with source code control (versioning) tools such as subversion, CVS and Perforce.

### No zoom functions

There was no ability to zoom in to (or enlarge) a VI which will be hard to see on a large, high-resolution monitor.

## What is LabVIEW used for?

**LabVIEW is used for 4 main purposes:**

**Automated Manufacturing test** of a component/sub-system/system.

***Manufacturing test systems*** are used to verify your product is within spec **before it leaves the plant**. The main drivers for manufacturing test are usually **(1)** test consistency, **(2)** error reduction **(3)** throughput improvements and **(4)** increased reliability/uptime.

Here's some good examples of manufacturing test systems that shown in figure 1.3:




**Figure 1.3: Examples of Manufacturing Test Systems**

**Automated Product design validation** of a component/sub-system/system.

***Product validation systems*** are used **during the design process** to validate that the design works as intended, before production begins. The main driver for automating product validation is that the number of dimensions that need to be swept across (e.g. temperature, power supply voltages, pressure) can be large and take a lot of time (sometimes repeating over many cycles) to both collect and analyze the data.

Here's some good examples of product validation systems that shown in figure 1.4:


**Figure 1.4: Examples of Product Validation Systems**

3. **Control and/or monitoring** of a machine/piece of industrial equipment/process.

The main drivers for using **LabVIEW** (with NI hardware) for industrial embedded applications are: **(1)** rapid prototyping and development using off-the-shelf hardware **(2)** tight tolerance timing or **(3)** acquisition of high-speed signals.

Here's some good examples of industrial embedded systems that shown in figure 1.5:



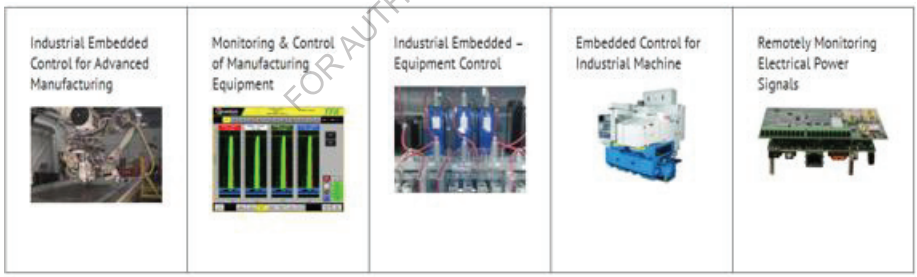



**Figure 1.5: Examples of Industrial Embedded Systems**

4. **Condition monitoring** of a machine/piece of industrial equipment.

The main drivers for condition monitoring are generally either **(1)** improving machine up-time/reliability or **(2)** reducing maintenance costs.

Some examples of condition monitoring applications include that shown in figure 1.6:

# CHAPTER ONE
# LABVIEW INTRODUCTION

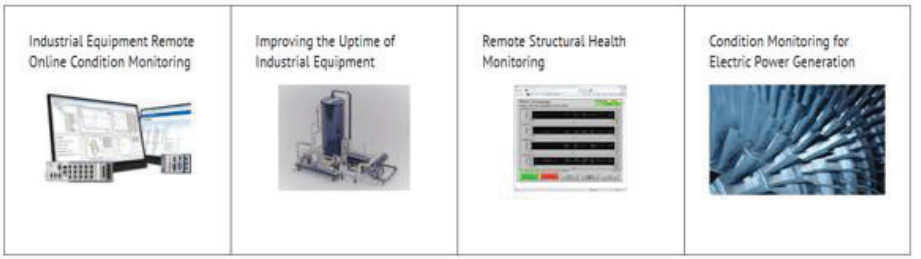



**Figure 1.6: Examples of condition monitoring applications**

## MyDAQ

Designed for hands-on experimentation, that is shown in figure 1.7. **NI MyDAQ** combines portability with a comprehensive set of features. NI MyDAQ allows for real engineering and, when combined with **NI LabView** and **Multisim**, gives students the power to prototype systems and analyze circuits in or outside of the classroom. **NI MyDAQ** hardware integrates with **NI LabVIEW** graphical development software, giving the students hands-on interaction with real analogue circuits, sensor measurements, and signal processing. It bridges the gap between theory and real-world practice by providing students with eight **LabVIEW** software-based instruments including a digital multimeter (**DMM**), oscilloscope, function generator, bode analyzer, dynamic signal analyzer, arbitrary waveform generator, digital reader and digital writer, as shown in figure 1.8. NI MyDAQ is compact enough to fit in a student's pocket, and is powered by a USB connection.

# CHAPTER ONE
# LABVIEW INTRODUCTION

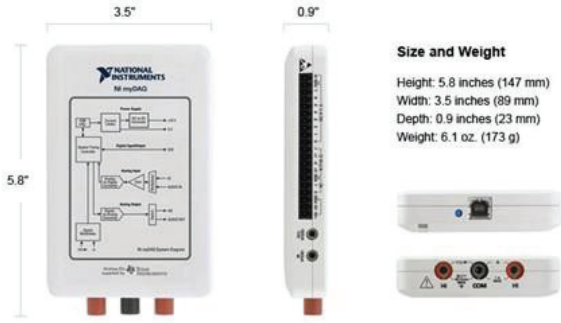



**Figure 1.7: MyDAQ device**

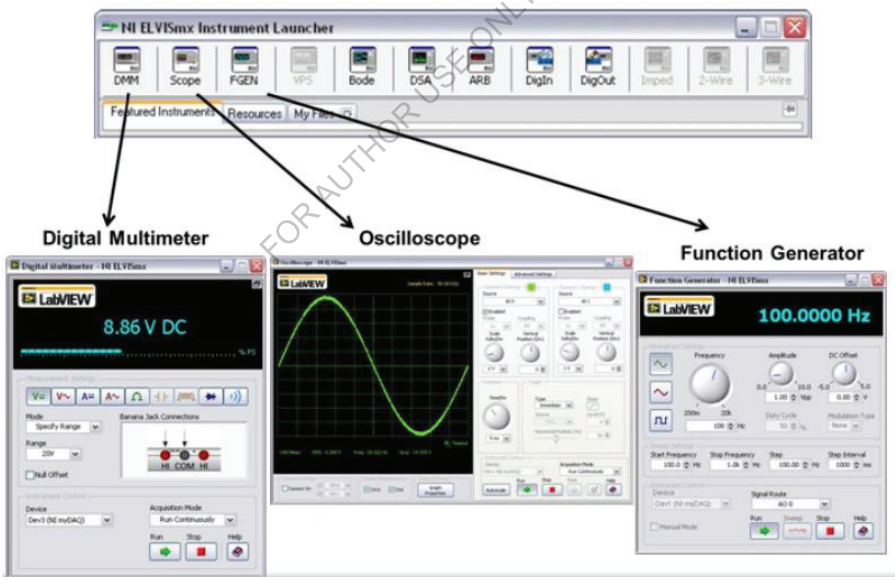





**Figure 1.8: LabVIEW software-based instruments provided by MyDAQ**

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

## Introduction

In this chapter, we will take a set of lessons from which to get to know the **LabVIEW** program. Through these lessons, you will be able to use the **LabVIEW** and create projects. We will learn about the toolbar in **Block Diagram** and **Front panel**. You will learn how to work with the program and make simple examples. All this will be done with the use of program illustrations.

## Lesson One: Learn about the environment of the LabVIEW program



### Aim of the lesson

**1) What are Virtual Instruments and what are their components.**

**2) Learn about the toolbar in the Front Panel.**

**3) Learn about the toolbar in Block Diagram.**

**4) Learn about Palette Tools.**

**5) Learn about Controls Palette.**

**6) Learn about Function Palette.**

**7) Learn how to download an example and implement it.**

### What are Virtual Instruments?

The software designed in the **LabVIEW** language is called **(Virtual Instruments (VIs)).** These are files with a **.Vi** extension.

The basis for working in the view lab program is to configure **Virtual Instruments** or

defines **VIs**, and each contains two main parts are the **front panel** or what can be called the **control panel**, and the second part is the block diagram the third part is **icon and connectors.**

## 1- Front Panel: The User Interface

This interface is used to interact with the user as all the different application requirements are set on it, as they are control and control elements (icon control), Graphic and illustration items (Waveform), Contact item, and Other. In figure 2.1 that shown the front panel.

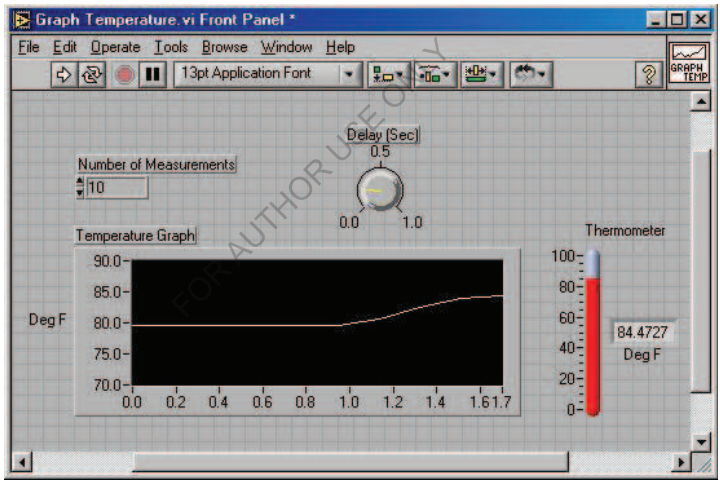



**Figure 2.1: Front Panel**

The front panel is built by controls and indicators, as shown in figure 2.2.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

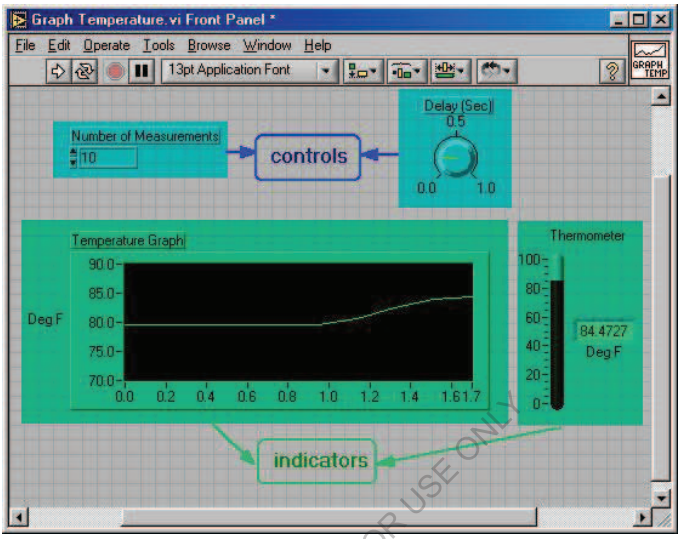



**Figure 2.2: Controls and indicators in Front Panel**



## Controls:

These are the input modules in VI such as knobs, push buttons, dials, etc. It is the same as the **input** units in the real electronic devices. In figure 2.3 that shown the controls types.

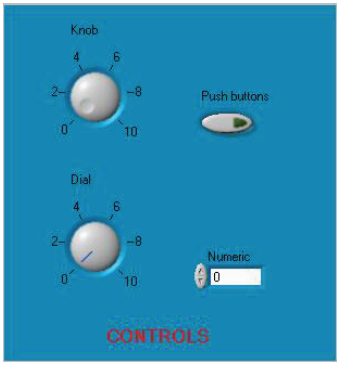



**Figure 2.3: Controls Types**

## Indicators:

These are VI output or display units such as graphs, leds, etc. It is the same as the **output** and display units in real electronic devices. In figure 2.4 that shown the indicators types.

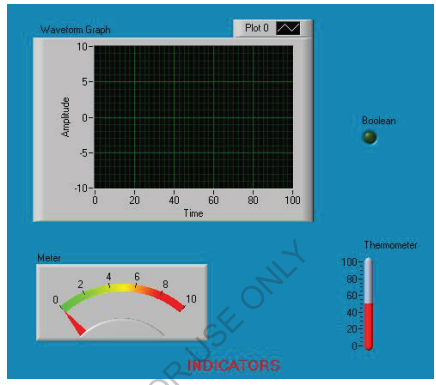



**Figure 2.4: Controls Types**



And what happens in the program is that the data moves from Controls to Block Diagram so that the program code is executed on it and then the results appear from Block Diagram to Indicators, as shown in figure 2.5.

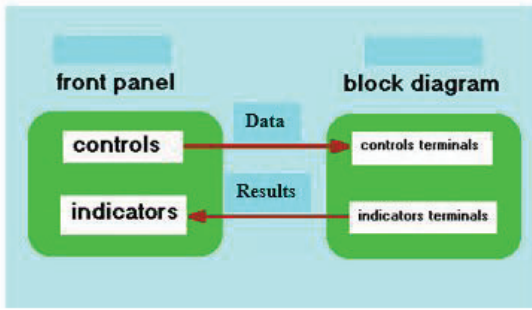



**Figure 2.5: Program Execution**

## 2- The Block Diagram: The Graphical Code

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

Chart contains the basic elements of the **user interface** (UI) components that have been inserted, and represents the main part which binds to and determine the mechanism of action of these elements, in terms of how (planned) are connected to some elements of some and determine its properties, which is what corresponds to the code in programming languages But it is a **graphical code**, or jobs that are already prepared, you should only recommend in order to complete your work.

Who distinguishes the elements of the scheme are very similar to those of the technical and engineering components and components. That professionals use completely not only in the form but also in the form and function. In figure 2.6 that shown the block diagram.

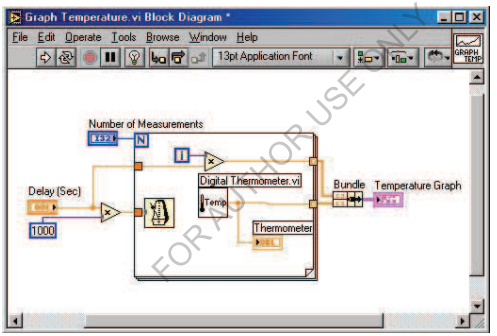



**Figure 2.6: Block Diagram**

Each **Control** or **Indicator** is located on the Front Panel, opposite to which there is a terminal in the **Block Diagram**, as shown in figure 2.7. This terminal is automatically placed in Block Diagram once Control or Indicator is placed in **Front Panel**. When any Control or Indicator is deleted from the Front Panel, its terminal is automatically deleted. You can only delete the terminal by deleting Control or the corresponding Indicator.

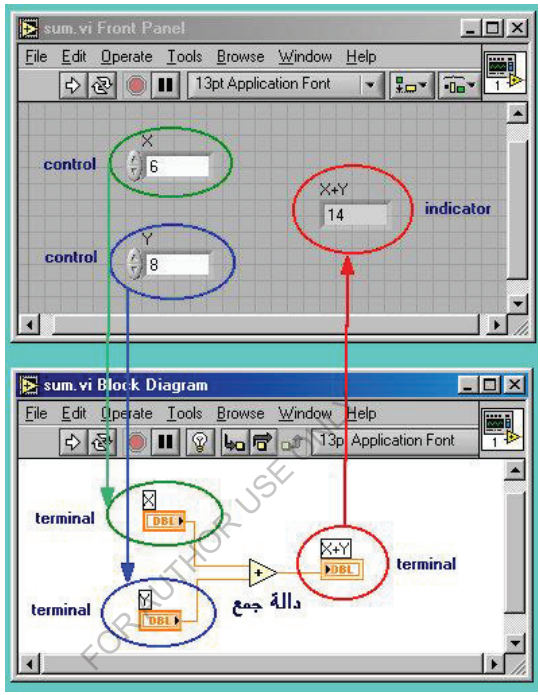



**Figure 2.7: Control and indicator in Front Panel and Opposite Terminal Block Diagram**

The **Block Diagram** in addition to terminals contains **SubVIs**, **Functions, Constants**, **Structures**, and **Wires** that make up the data path.

### 3- The Icon and Connector Pane

These connections are used if the project that we are preparing is connected to an external terminal to receive or send data from outside the PC and are on the upper left of the Diagram or panel user interface. It represents the basis for the part that is the nature of the work of the project, which we are preparing. Also have the advantage of excellent too, is the ability to use a virtual front subset are called when it is needed in the event that the project contains many of the fabrication

parts that we link Some are completely the same is the idea of subprograms in programming languages. This Icon can contain drawing, writing, or both, as shown in figure 2.8.

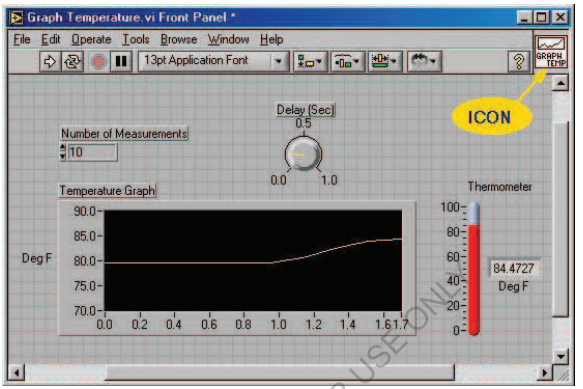



**Figure 2.8: VI Icon**

This Icon represents VI when used as a function in another VI and the VI used then is called **SubVI**, as shown in figure 2.9.

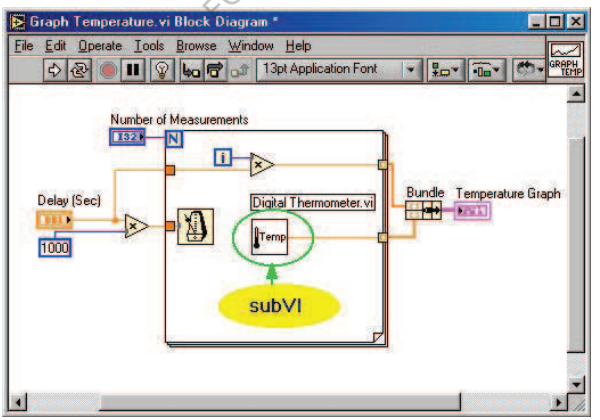



**Figure 2.9: SubVI Icon**

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

Basically, the idea that gives the program **LabVIEW** great strength is the ability to configure **complex applications** in tasks and a **high degree of efficiency** is very easy steps and flexible. In order to access this configuration **sub-applications** or interfaces subset partial tasks required in order to create an integrated application based on assembly These are partial applications together, this is the concept of **SubVIs.**

## Connectors Pane

It is a **set of links** that indicate and specifies the way SubVI is connected in **Block Diagram**. These connections are the **inputs** and **outputs** of the SubVI, as shown in figure 2.10.

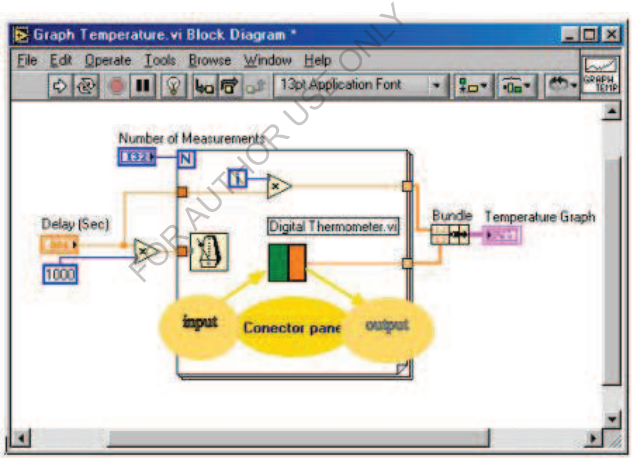



**Figure 2.10: Connector Pane**

LabVIEW's strength lies in its software architecture. Since each VI can be SubVI in another VI. There is no specific limit to the number of SubVI found in VI, as shown in the example of figure 2.11.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

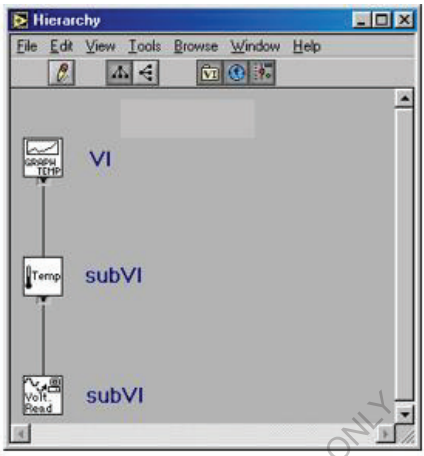



**Figure 2.11: Number of SubVI**



## Toolbar in Front Panel

It is used in the **implementation** and **design** of VI.


To run the program. And be like this . If there is an error in the program it appears like this .

For continuous implementation of the program until the program is stopped or pressed again and appears during the continuous implementation of the program like this .

To stop the program completely and immediately, it is only available during the implementation of the program.

To stop the implementation of the program temporarily and complete the implementation of the program by clicking on it again.

**Note:** In the event of a pause, LabVIEW shows where the program stopped in Block Diagram.

Dropdown menu for changing text properties: color, size, font type.

 Dropdown list to align units together.

 Dropdown list for organizing the distances between units.

 A pull-down list to arrange the units on top of each other, it determines who appears in front and who is in the back.

## Toolbar in Block Diagram


 **Highlight:** Clicking on it shows how the program is implemented using data transmission, and it looks like this  .

 **Step Into:** One step: To implement the program. With it you can enter subVI or loops to implement it step by step.

 **Step Over:** To implement the program, one step with each pressure, considering that SubVI or Loop is implemented in one step without entering it.

 **Step Out:** Exit loop or SubVI in one step.



## Tools Palette

They are opened from:

**Window ➔ Show Tools Palette**

As shown in figure 2.12.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

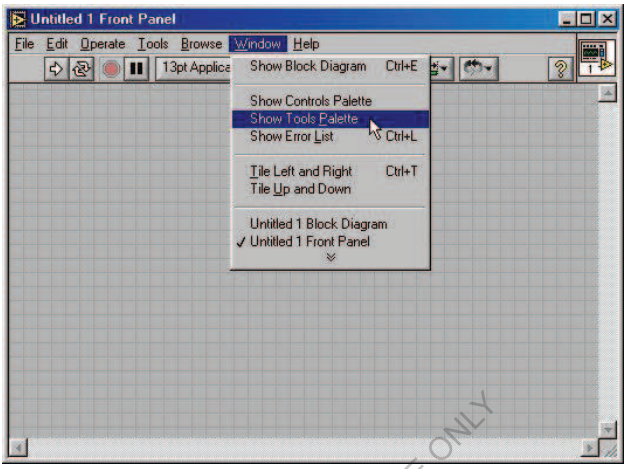



**Figure 2.12: Open the Tools Palette**

In figure 2.12 that shown the tools palette.


**Figure 2.13: Tools Palette**

It is a set of essential tools in the LabVIEW environment for VI design, testing and implementation.

**Operating Tool:** Change the values of Controls or typed text. Usually used during program implementation.

**Positional Tool:** Selecting, moving, and resizing units (Objects).

**Labeling Tool:** To edit any text or make text anywhere.

**Wiring Tool:** Connect units in Block Diagram.

**Object Shortcut Menu:** Symmetry the right-click on the cursor.

**Scrolling Tool:** To make a sliding window without using the sliding bar.

**Breakpoint:** To make a breakpoint at which the program stops execution. Used to test the program.

**Probe Tool:** It is used to create a data display point (Probe) to display the values while the program is executed for testing or handling errors.

**Color Copy Tool:** To take color values from any position in the window for use By Coloring Tool

**Coloring Tool:** To color the units.



## Controls palette

It opens from the Front Panel window.

**Window ➔ Show Controls palette**

In figure 2.14 that shown the controls palette.

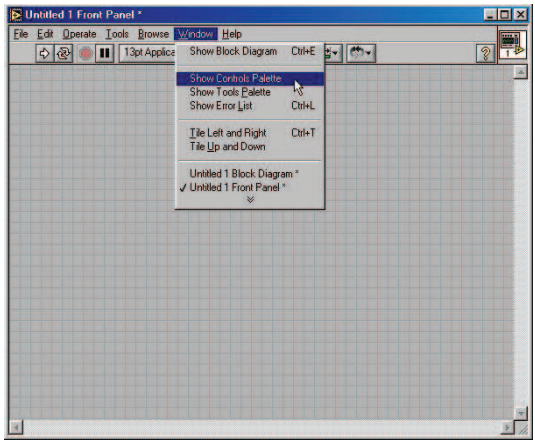



**Figure 2.14: Open the Controls Palette**

In figure 2.15 the controls palette in front panel.

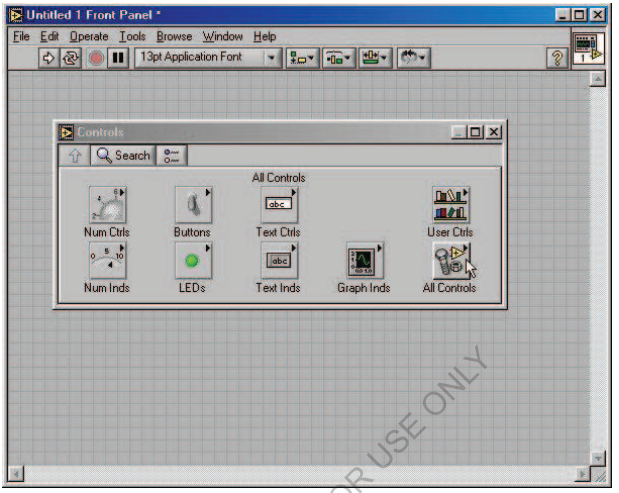





**Figure 2.15: Controls Palette in Front Panel**

In figure 2.16 that shown the controls all.

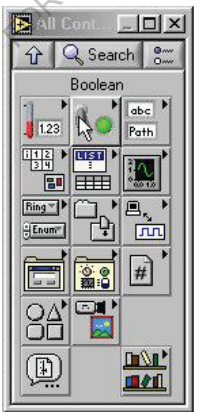



**Figure 2.16: Controls All**

It can also be opened by right-clicking on the Front Panel window, as shown in figure 2.17.

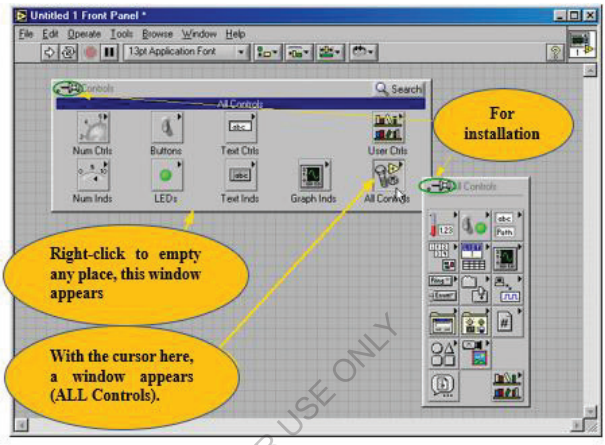



**Figure 2.17: Controls Palette**



## Functions Palette

It opens from the block diagram window.

**Window ➔ Functions palette Show**

In figure 2.17 that shown the functions palette.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

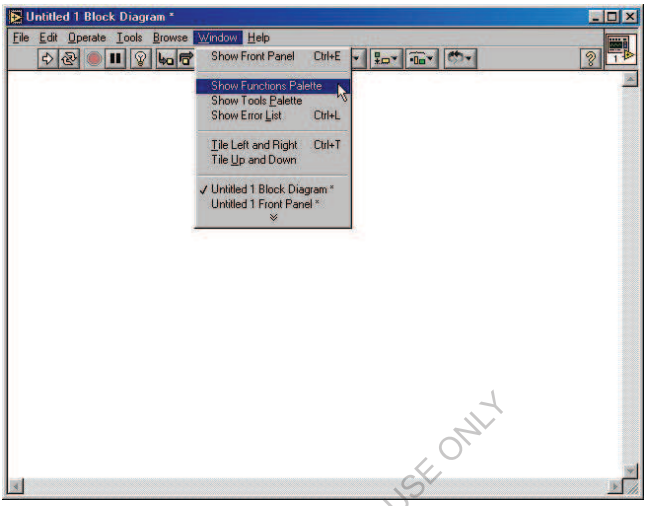





**Figure 2.17: Functions Palette in Block Diagram**

In figure 2.18 the controls palette.


**Figure 2.18: Functions Palette**

It can also be opened by right-clicking on a position in the Block Diagram window, as shown in figure 2.19.

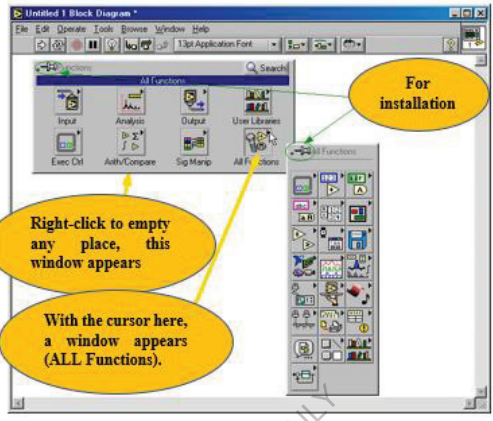



**Figure 2.19: Functions Palette**

CHAPTER TWO
LABVIEW LEARNING LESSONS

# Lesson Two: Create VI

## Aim of the lesson

**1) Tools Palette.**

**2) VI Basics Elements (Front panel, Block Diagram, and The Icon and Connector).**

**3) Dataflow Programming.**

**4) Create LabVIEW Program File.**

**5) Save LabVIEW Program File.**

## Tools Palette

As we learned from the previous lesson, we are getting the Tools Palette from **Window ➔ Show Tools Palette.**

The tool selection can be in one of two ways:

**Automatic:** LabVIEW automatically changes the tool to the appropriate tool.

**Manual:** The programmer selects the tool he wants by clicking on the tool in Tools Palette.

To make Automatic or Manual click the Automatic Tool Selection icon above Tools Palette, as shown in figure 2.20.



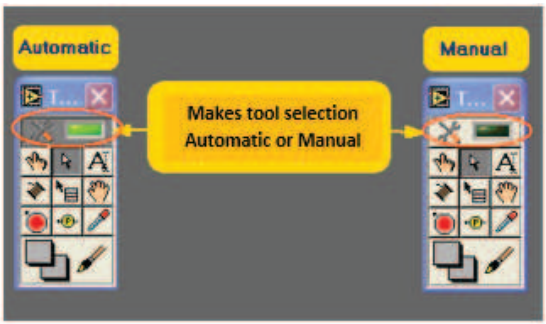



**Figure 2.20: Tools Palette**

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

As we mentioned in the previous lesson, the VI consists of three main components:

**The Front Panel**, **The Block Diagram** and **The Icon and Connector Pane.**

## 1- The Front Panel

The Front Panel is built by Controls and Indicators.

The Controls panel is used to place Controls and Indicators on the Front Panel.

We will show some of the most used examples of Controls and Indicators.

- **Numeric Controls and Indicators**

  Of the most used units.

### Numeric Controls

In figure 2.21 that shown the numeric control list.



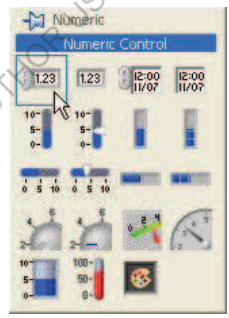



**Figure 2.21: Numeric Control List**

**Numeric Control** contains several properties such as **title**, **value**, and **arrows** to change that value, as shown in figure 2.22.

The value of Numeric Control can be changed by the program user while executing the program to enter the values for the program.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

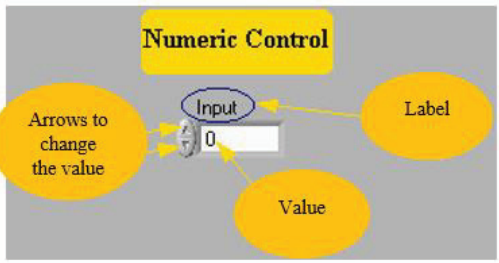



**Figure 2.22: Numeric Control (Input)**

To change the value of Numeric Control:

- Shares can be used by the Operating Tool , as shown in figure 2.23.

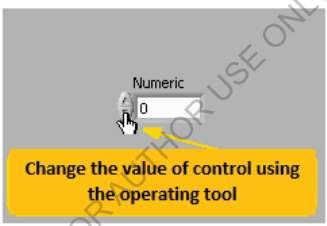





**Figure 2.23: Change the value of control using the operating tool**

- By double-clicking on the Control value by Labeling Tool or Operating Tool and then typing the new value. Then press the <**ENTER**> key or click on the icon at the top of the **Toolbar** or by clicking anywhere outside Control.

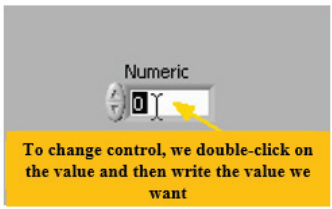



**Figure 2.24: Change Control Value**

## Numeric Indicators

In figure 2.25 that shown the numeric Indicator list.

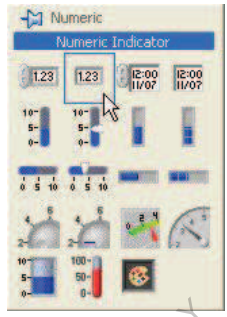



**Figure 2.25: Numeric Indicator List**

**Numeric Indicator** contains several properties such as **title** and **value**, as shown in figure 2.26.

The value of **Numeric Indicator** cannot be changed by the user during the implementation of the program, but changes through **Block Diagram**.



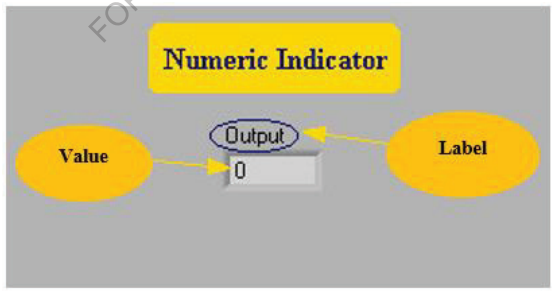



**Figure 2.26: Numeric Indicator (Output)**

- **Boolean Controls and Indicators**

It is used to enter and display the **True** or **False** binary values which are the same as **Switches**, **Push Buttons** and **LEDs**, as shown in figure 2.27.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

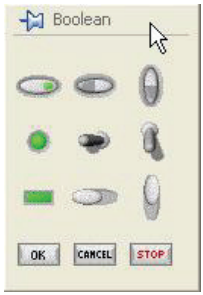



**Figure 2.27: Boolean Controls and Indicators**

Among the most used units are **Vertical Toggle Switch** and **Round LEDs**, as shown in figure 2.28.




**Figure 2.28: Vertical Toggle Switch and Round LEDs**

**Note:**

**Indicators** and **Controls** properties can be changed by pressing the right click on the unit, whether Indicator or Control, a drop-down menu appears showing us what can be changed, as shown in figure 2.29.

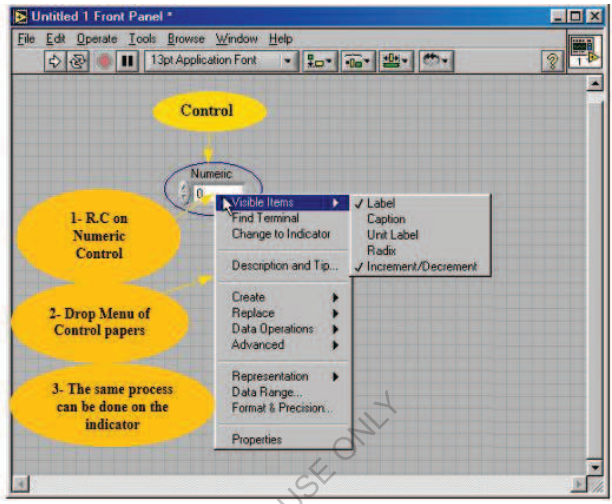



**Figure 2.29: Change Properties of Control**



## 2- The Block Diagram

The block diagram consists of three components: **Nodes, Terminals, and Wires**. In figure 2.30 that shown the block diagram components.

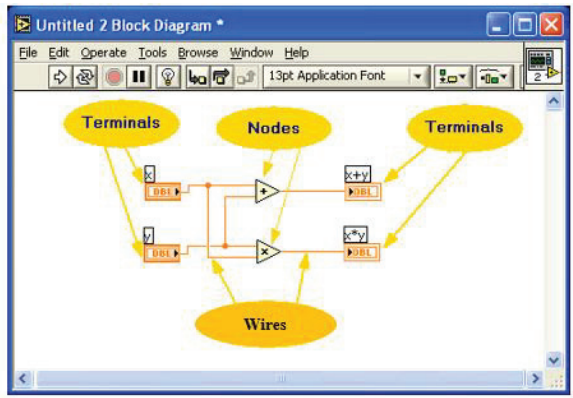



**Figure 2.30: Block Diagram Components**

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

## Nodes

It is any object in Block Diagram that has **entries** or **exits,** or **both**, and it performs an operation while executing the program.

The types of Nodes are: **Functions**, **SubVI**, and **Structures**.

### Functions:

These are ready-made basic functions built into the LabVIEW environment.

It can be obtained from **Functions Palette**, as shown in figure 2.31.




**Figure 2.31: Function Palette**

### SubVI

It is a pre-built VI used in another VI Block Diagram and is identical to **Subroutine** in other programming languages.

### Structures

They are process controls. In figure 2.32 that shown the structures examples.

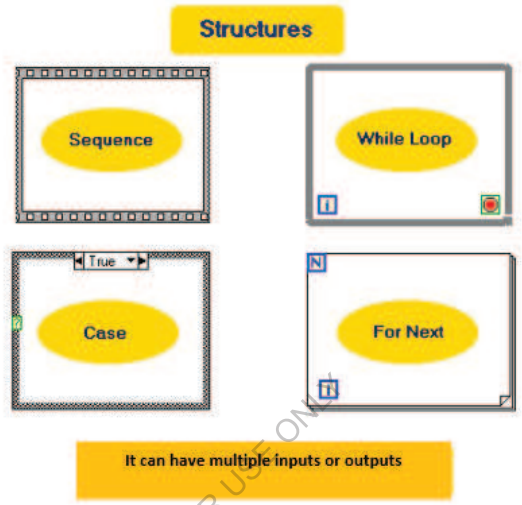



**Figure 2.32: Structure Examples**



## Terminals

They are of two types: **Indicator or Control Terminal** and **Node Terminals**.

### Indicator or Control Terminal

As we explained before for each Control or Indicator present in the Front Panel there is corresponding Terminal in the Block Diagram. This Terminal has Data Type for Control or Indicator.

Data Type determines the data type as well as each of the storage capacity of the value and thus the extent of this value. Each Data Type has a specific color, as shown in figure 2.33.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

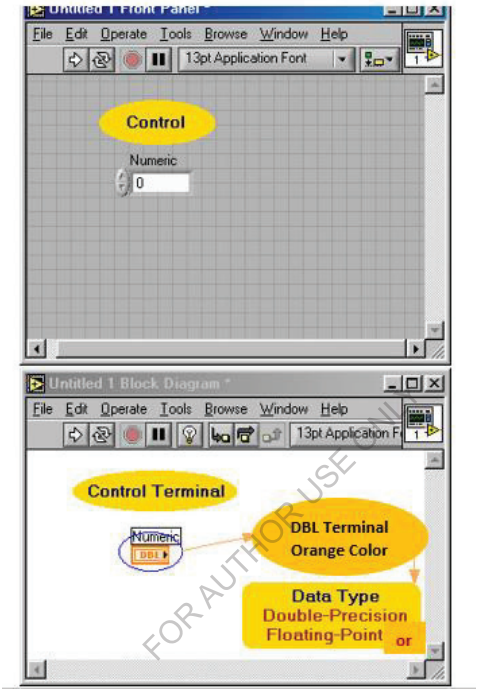



**Figure 2.33: Control Terminal Color**

## Node Terminals

Each Node has a Connector Pane that is a set of connections that shows and specifies how it connects to any Node's input and output locations. These connections are **Node Terminals**. To display the Connector pane of Node

**Right-click on the Node ➔ select Visible Items ➔Terminals**

This is shown in figure 2.34.

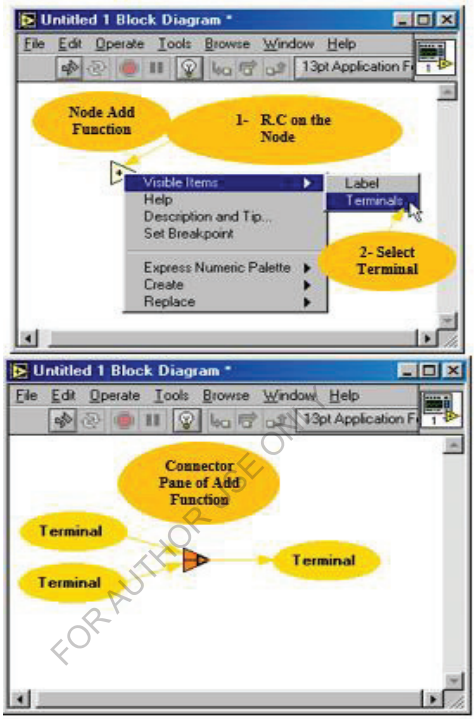



**Figure 2.34: Display Connector Pane of Node**

## Wires

It is the data path between the Block Diagram (Object) units.

Wire connects to a single data source, but this data can be accessed by more than one entry for a **function** or **SubVI**, as shown in figure 2.35.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

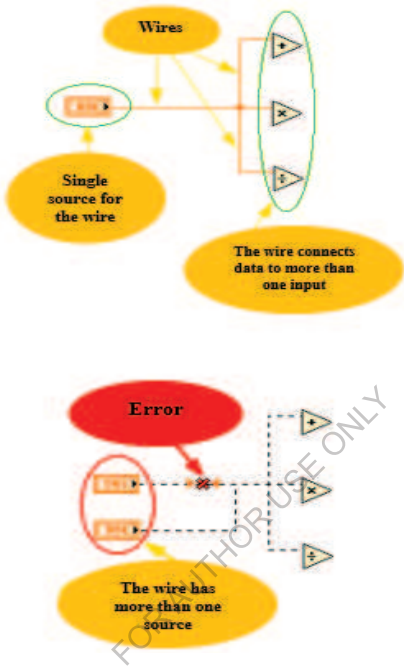





**Figure 2.35: Wires**

Each link has a **color**, **shape** and **thickness** based on the link's **Data Type**, as shown in table 2.1.

**Table 2.1: Different Style, Color, and Thickness Of Wires**

| | Scalar | 1D array | 2D array | Color |
|---|---|---|---|---|
| Floating-point number | | | | orange |
| Integer number | | | | blue |
| Boolean | | | | green |
| String | | | | pink |

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

## Dataflow Programming

The implementation of **LabVIEW** software is dependent on **Dataflow**.

Node executes immediately when data is ready on its inputs. Upon completion of Node implementation, the outputs immediately exit their outputs to be ready for the next Node in the data flow path. This is in contrast to other programming languages that rely on **Control Flow**, where instructions are executed successively according to the order in which they are written in the program

## Create LabVIEW Program File

**Steps:**

1- Download **LabVIEW Program**.

2- Download a **new file**.

3- Press. **New VI.**

4-Select **Blank VI** for **new VI.**



## Save LabVIEW Program File

You can save the program as a separate file in the extension **vi**, or as part of a library that contains several programs and that is in an **.llb** extension.

1- **Save the file in a separate file:**

**Choose File ➔ Save**

This is shown in figure 2.36.

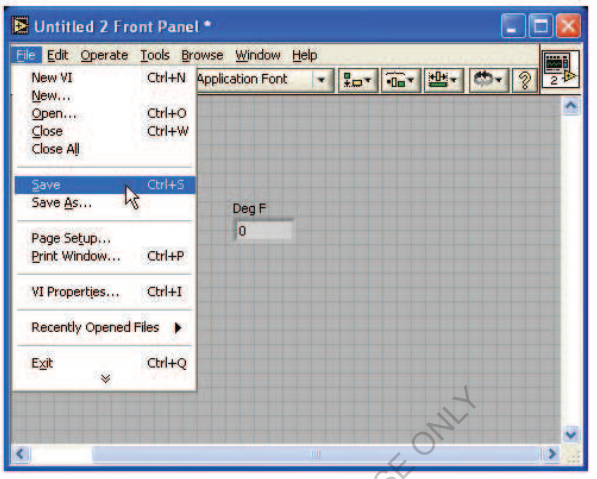



**Figure 2.36: Save LabVIEW File**

Specify the save location and file name, as shown in figure 2.37.

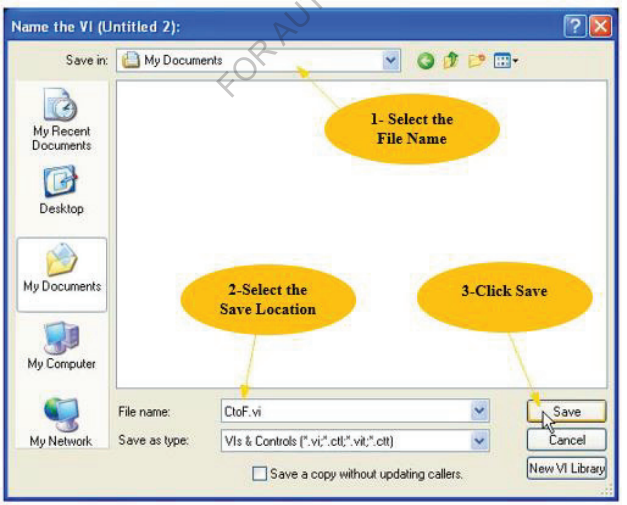



**Figure 2.37: Select the Save Location and File Name**

2-**Save the program in a library**

Select **(File ➔Save)** if this is the first time you have stored it.

Or, select **(File ➔Save as)** if you have stored it before and want to store it under another name or elsewhere. In figure 2.38 that shown the Save in Library.

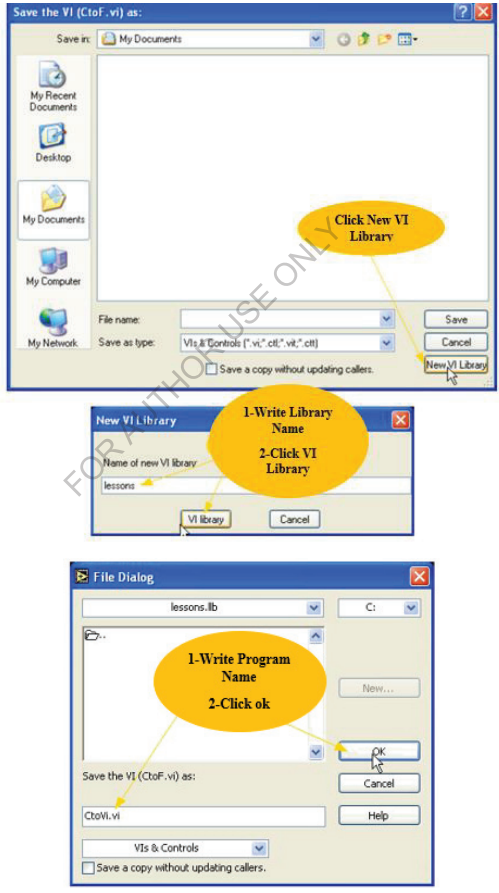



**Figure 2.38: Save in VI Library**

CHAPTER TWO
LABVIEW LEARNING LESSONS

# Lesson Three: Edit VI

## Aim of the lesson

**1) Learn about all operations of VI editing.**

**2) Learn about using the Help types available with the LabVIEW version.**

**3) Learn about ways to track errors to fix them to get to actionable VI.**

**4) Learn about the different ways to test implementation VI.**

## Edit VI

### Adding new Objects

We learned from previous lessons that we are using the Controls Palette to add Controls or Indicators to the Front Panel. Also, you can add Controls, Indicators, or Constants by right-clicking on the Node Terminal in Block Diagram and choosing Create, as shown in figure 2.39.



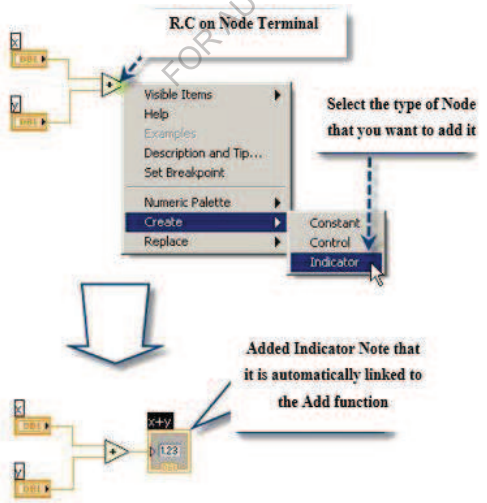



**Figure 2.39: Adding New Objects**

## Select Objects

The **Positioning tool** is used to **select** and **move** units in the Front Panel or Block Diagram, as shown in figure 2.40.

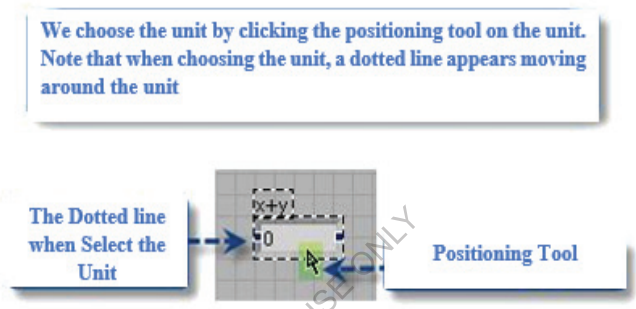



**Figure 2.40: Position Tool to Select and Move units**

To select more than one unit, press the **Shift key** while pressing the tool on the units to be selected.

You can also click the tool anywhere and pull the cursor while holding down to select all the units that we want.

In figure 2.41 that shown the selection of objects.

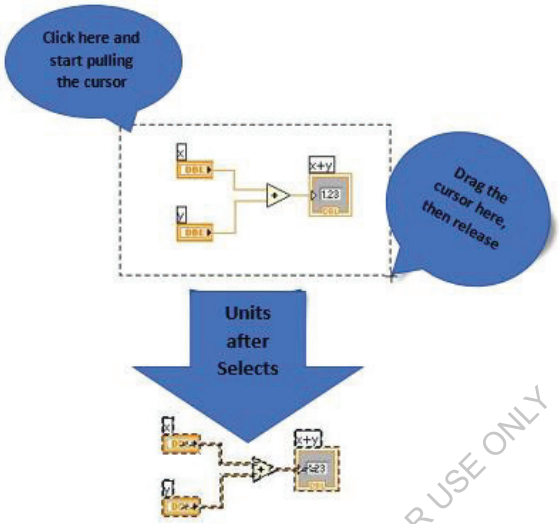





**Figure 2.41: Selection Objects**

## Moving Objects

The tool  is used to select and move units.

After selecting the unit, you can drag it anywhere, or use the arrows to move the selected units.

When using the arrow in the move, you can press the **Shift key** with the **arrow** to speed up the move.

## Delete Objects

Objects are deleted first by selecting the **tool** and then pressing **Delete** or **Edit >> Clear**.

**(Undo / Redo) Steps:**

If you took some steps and want to cancel them, use **Undo** from the **Edit menu** and to return those steps after canceling them, select **Redo** from the same list.

## Copy Objects

If you want to obtain more than one copy of any unit.

Choose the object with the tool  then choose **Edit >> Copy** and then choose **Edit >> Paste**.

Or choose the object with the tool then press the **Ctrl** key and then drag the unit to any other place while holding the key down, as shown in figure 2.42.

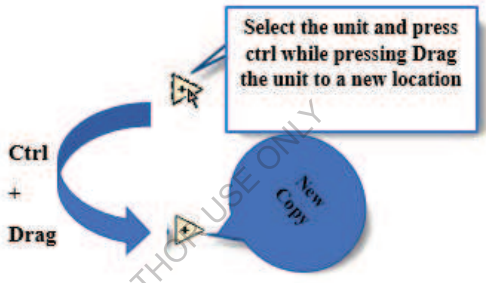





**Figure 2.42: Copy Object**

## Insert Label

The label is used to identify objects, as shown in figure 2.43.


**Figure 2.43: Insert Label**

There are two types of labels:

- **Owned Labels**

Each object has a **label**. This label can be changed individually, but if you move the unit it will move with its label. The label of the object can be shown or hidden, as shown in figure 2.44.

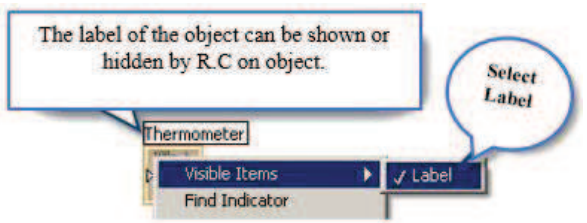



**Figure 2.44: Hidden and Showing the Label**

- **Free Labels**

It is not associated with any object and can be **added**, **moved** or **deleted** without any association with anything else.

It is used for writing anywhere in the program, whether on the Front Panel or Block Diagram.

To create a free title, press the Labeling tool on an empty space and type what you want and then click on any other place or press on the **Toolbar** or press the **Enter key**, as shown in figure 2.45.




**Figure 2.45: Make Free Label**

## Select and Delete Wires

Any horizontal or vertical part of the link shall be called **Wire Segment**, as shown in figure 2.46.


**Figure 2.46: Wire Segment**

The meeting point of two Wire Segment is called **Bend**, as shown in figure 2.47.

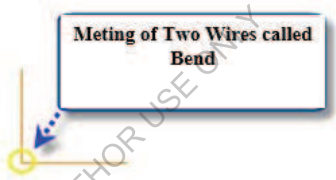





**Figure 2.47: Bend**

However, if the meeting point is between three or more Wire Segment called **Junction**, as shown in figure 2.48.

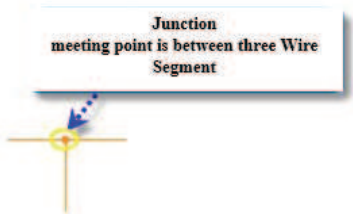



**Figure 2.48: Junction**

**Wire Branch** is defined as all **Wire Segment** that reaches Junction to Junction or Junction to Terminal or Terminal to Terminal without any Junction between them, as shown in figure 2.49.

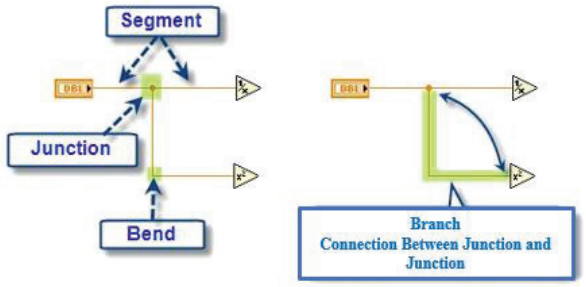



**Figure 2.49: Wire Branch**

To **select Wire Segment**, we press the one-click on the link by using the tool  , as shown in figure 2.50.



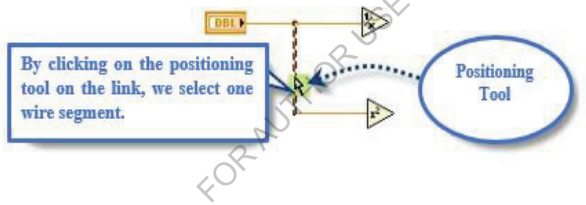



**Figure 2.50: Select Wire Segment**

By double click, select Branch, as shown in figure 2.51.

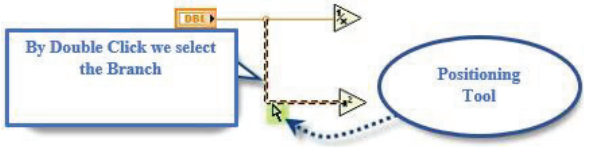



**Figure 2.51: Select Branch**

By clicking three times in a row, we choose the whole link, as shown in figure 2.52.

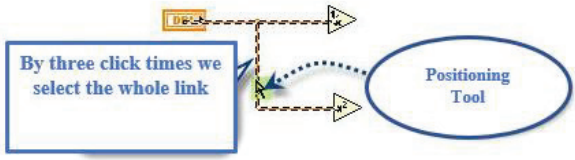



**Figure 2.52: Select Whole Link**

After selecting any part, it can be deleted by pressing the **Delete key.**

## Broken Wires

It appears in the form of a **dashed line** and it means that there is an error in the connection. This error can occur for several reasons, including for example that two units are not consistent in the data type, as shown in figure 2.53.



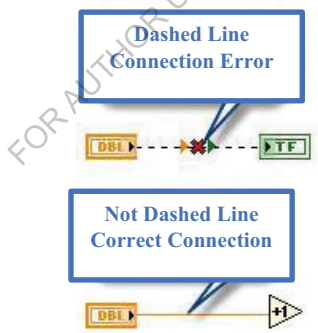



**Figure 2.53: Connection Wire**

By moving the tool by mistake a box appears explaining the cause of the error, as shown in figure 2.54.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

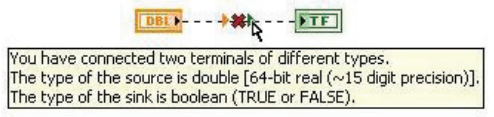



**Figure 2.54: Box Appears Explaining the Cause of The Error**

The broken links can be deleted in the normal way by selecting the link by pressing three consecutive presses and then pressing the key **Delete.**

Or, click **Remove Edit Wires** from the **Edit menu**, also click **Ctrl + B** to delete all the broken links.

**Warning:** Be careful when deleting all broken links, because sometimes some of the broken links appear because you have not yet finished connecting the rest of the links.



## Change Font Properties

**Font**, **Style** and **Font Size** can be changed for any text by choosing the text for which you want to modify the font properties and changing the properties from the **Text Setting drop-down menu**, as shown in figure 2.55.

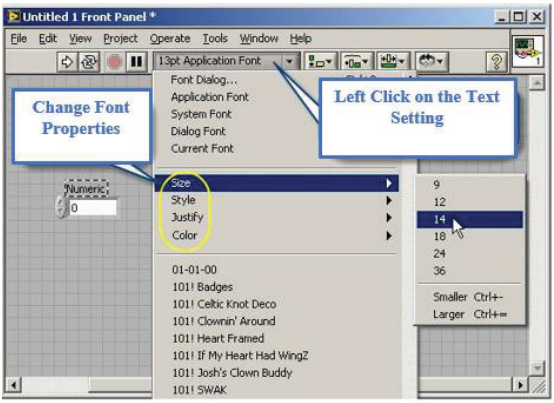



**Figure 2.55: Change Font Properties**

Some Controls or Indicators have more than one text.

Like **Graph** has a **title**, **display indicator**, **X-Axis**, and **Y-Axis**, as shown in figure 2.56. Font properties for each text can be modified individually by highlighting the text with the tool and choosing Edit from the Text Setting menu.

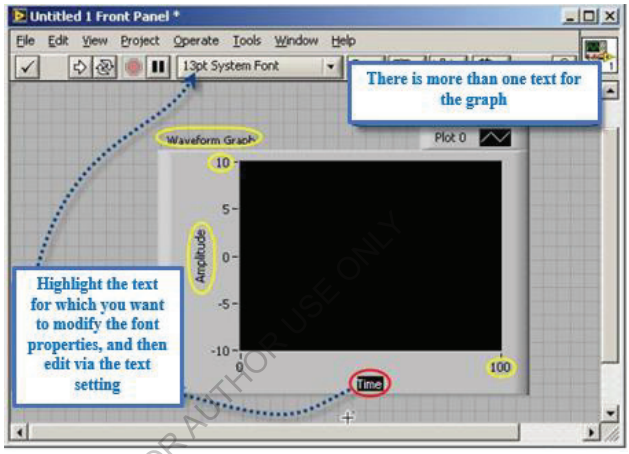



**Figure 2.56: Graphic has more than one Properties**

## Align Objects

If you want to align an array of units, first select them and then choose the alignment type from the Align list, as shown in figure 2.57.

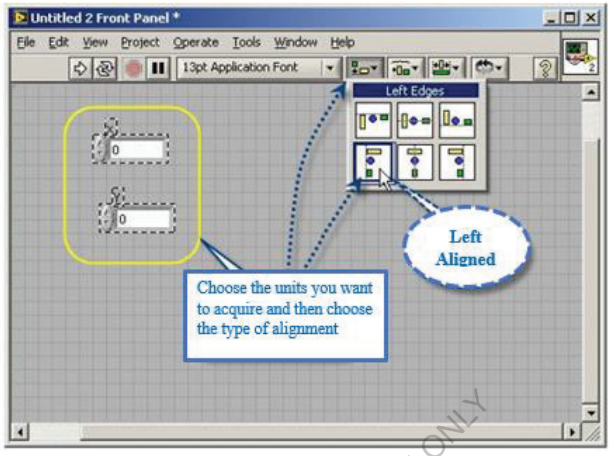



**Figure 2.57: Left Aligned Object**

## Setting the distances between Objects

Select the units to which you want to set the **distances**, then choose how to set the distances from the **Distribute Objects menu**, as shown in figure 2.58.



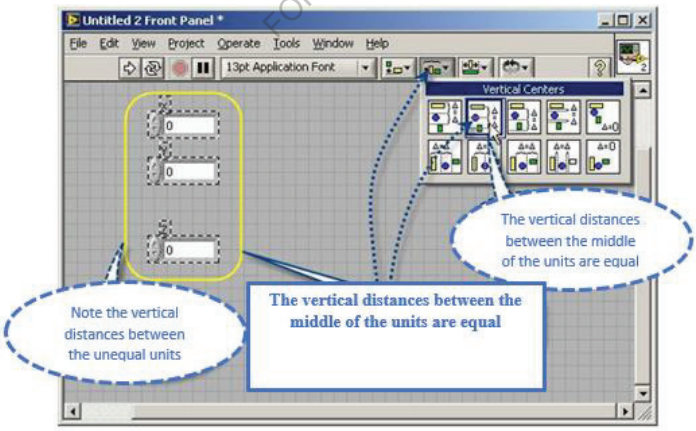



**Figure 2.58: Setting the Distance between Objects**

## Copy Objects between VIs and other Applications:

Units of VI can be copied to another using **Edit >> Copy** and **Edit >> Paste**. Images or text from other applications can also be copied to the Front Panel or Block Diagram.

## Change Color

The **color** can be changed for many objects, but not all. **For example**, you cannot change the color of Terminals in Block Diagram because it is a function of **Data Type**.

To change the color, choose the **Coloring tool** , right-click on the object you want to change its color, and then choose the color you want, as shown in figure 2.59.

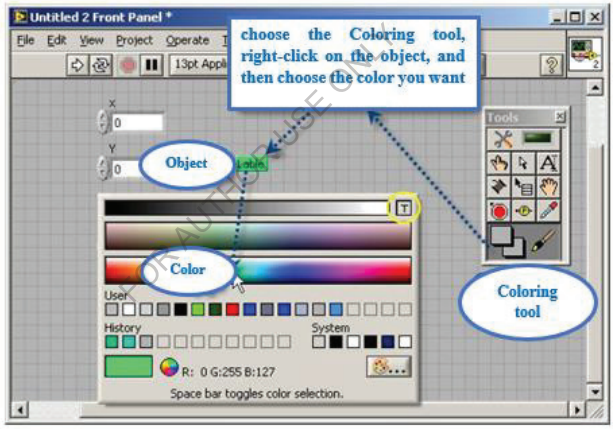



**Figure 2.59: Change Color**

To make the units transparent, choose T from the Color Palette (located in the yellow circles above).

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

## LabVIEW Help

Consider the importance of using **LabVIEW Help** in creating and testing VI and identifying errors in the program. We will know how to use **LabVIEW Help** before knowing how to test VI.

### Context Help window

This window shows the basic information about the objects on the Front Panel or the Block Diagram.

By moving the cursor over any object, basic information about that object appears in the **Context window**, as shown in figure 2.60.

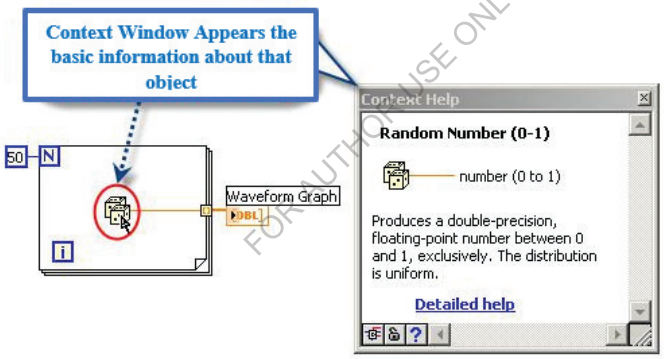





**Figure 2.60: Context Help Window**

To show or hide this window, select **Help>>Show Context Help**, as shown in figure 2.61.

Or press **Ctrl + H** or by clicking (Show Context Help) in Toolbar.

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

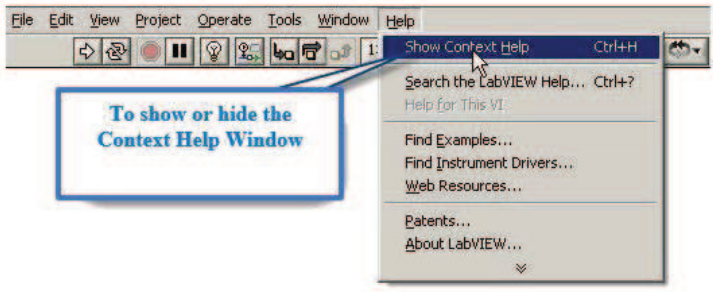



**Figure 2.61: Show or Hide the Context Help Window**

When you move the cursor over any object in the **Front Panel** or **SubVi** (Block Diagram, Function, Constant, Control, or Indicator), this object appears in **Context Help** as an **icon** with the **terminals** and their **labels displayed**, as shown in figure 2.62.



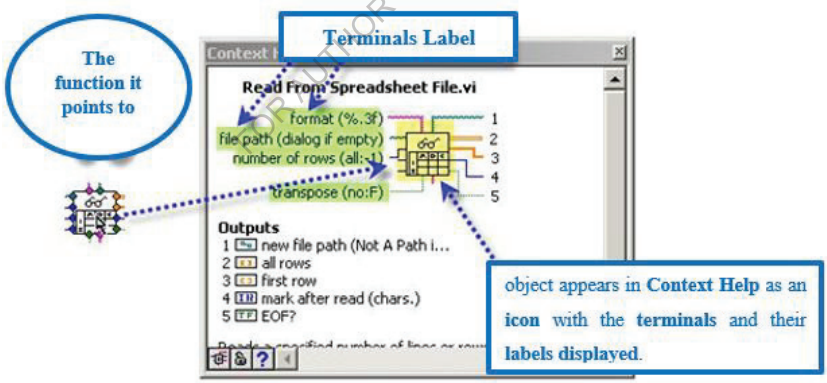



**Figure 2.62: Object information in Context Help**

There are three types of Terminals:

### 1- Required Terminals

You must connect to it and if you do not connect to it, the program will not be executable and the **Run key** will look like this  .

The **titles** of these Terminals appear in the Context Help window in **Bold line**, as shown in figure 2.63.


**Figure 2.63: Required Terminal**



### 2- Recommended Terminals

It is preferred that you connect to it (that is, if you do not connect to it, these Terminals will take the **default values**) and the labels of these Terminals will appear in the normal line (**Plain text**), as shown in figure 2.64. VI can be executed without connecting these Terminals, so they will be taken.

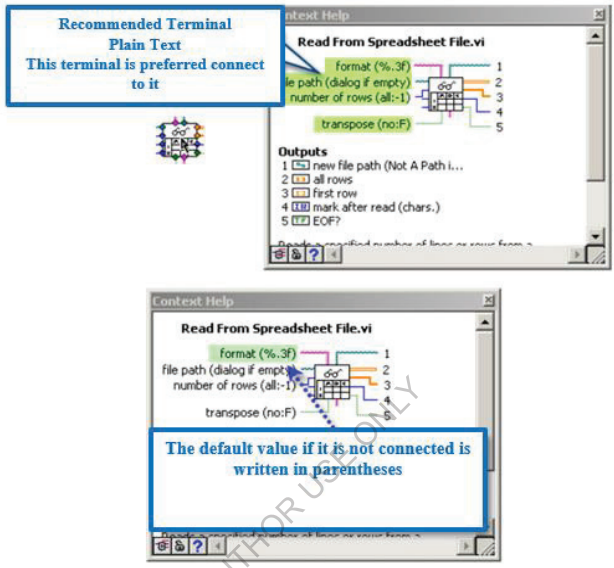



**Figure 2.64: Recommended Terminal**

### 3- Optional Terminals

**Optional connection** the labels of these Terminals appear in **faded font**, as shown in figure 2.65.

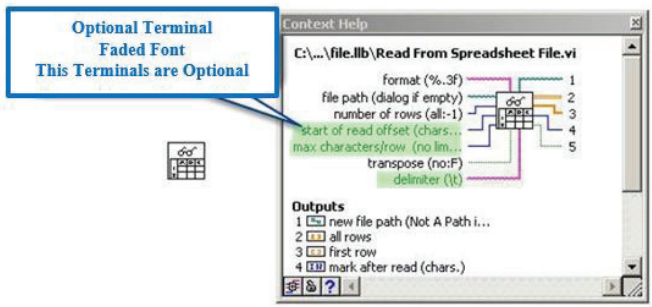



**Figure 2.65: Optional Terminal**

There are three keys in the **Context Help window**:

### 1- Hide Optional Terminals and Full Path

This key hide or displays optional Terminals, and it only displays the path of the file in which the function is stored or SubVI, or shows the name of the function, as shown in figure 2.66.

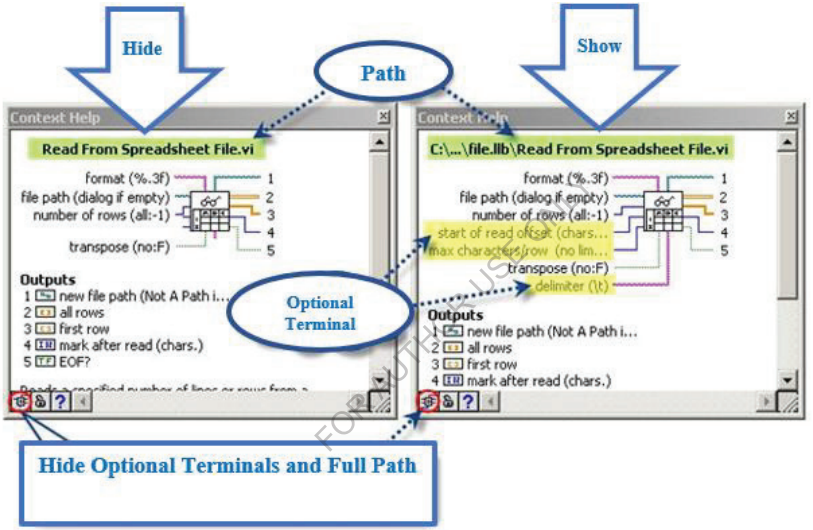



**Figure 2.66: Hide Optional Terminals and Full Path**

### 2- Lock Context Help

This key installs the contents of the **Context Help window**. In this case when we refer to the cursor on other units, the window contents do not change, as shown in figure 2.67.

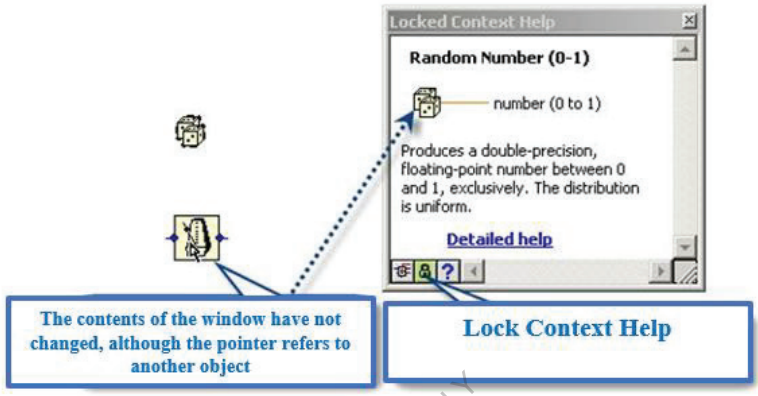



**Figure 2.67: Lock Context Help**

### 3- Detailed help

If there is more information in the LabVIEW Help for the unit displayed in the Context Help window, this information can be viewed by clicking on this key or by clicking on the word "**Detailed Help**" at the bottom of the window, as shown in figure 2.68.



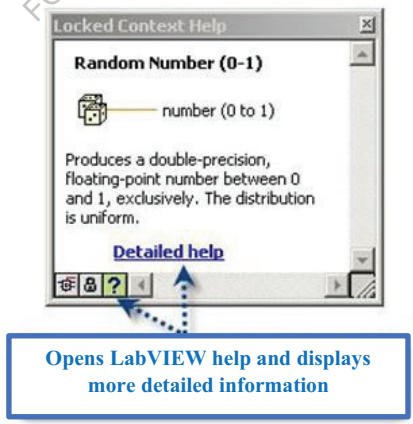



**Figure 2.68: Detailed Help**

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

## LabVIEW Help

With LabVIEW, you can find complete Help files for anything you want to program with LabVIEW.

We have seen that LabVIEW can be opened through the Context Help window.

LabVIEW Help can also be opened from the menu **Help >> Search the LabVIEW Help**, as shown in figure 2.69.

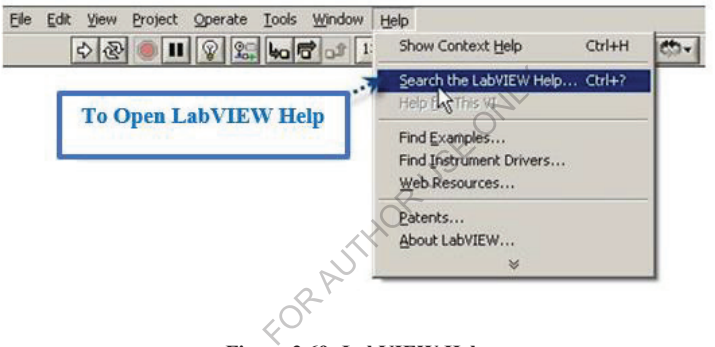





**Figure 2.69: LabVIEW Help**

Also, right-click on any object and choose **Help**, as shown in figure 2.70.

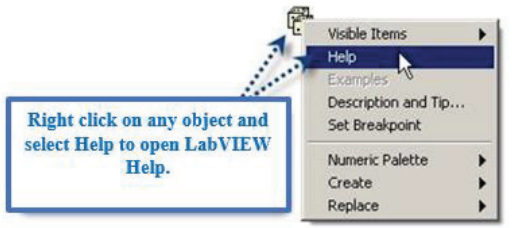



**Figure 2.70: LabVIEW Help**

## NI Example Finder

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

With **LabVIEW** there are many examples that you can modify to suit the application you want and some parts of it can be copied for use in your application and it helps in understanding many of the topics in the **LabVIEW** program.

The **NI Example** Finder is used to search and find the example you want, whether it is located with a copy of **LabVIEW** or on the **NI company website**.

The **NI Example** Finder is called by pressing **Find Examples** on the Getting Started **LabVIEW** screen, as shown in figure 2.71.

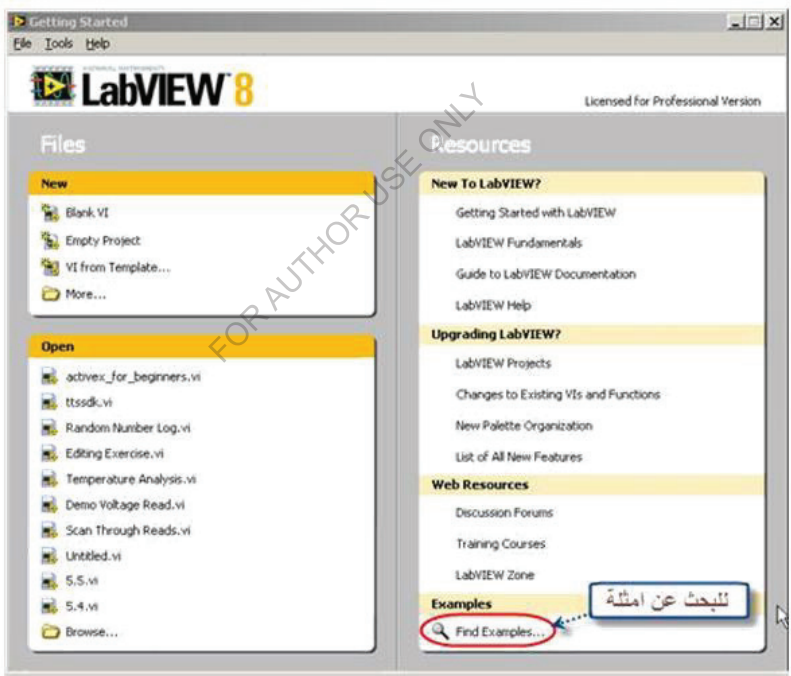



**Figure 2.71: Find Example**

Or by select **Help>>Find Example**, as shown in figure 2.72.

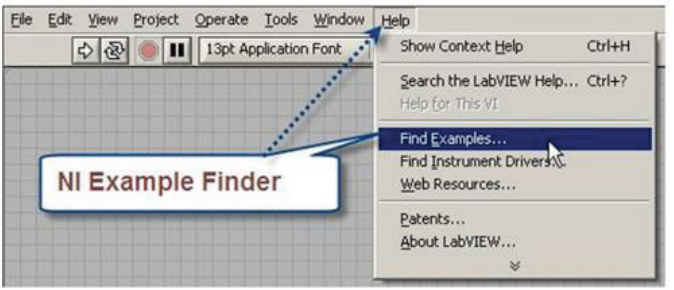



**Figure 2.72: NI Example Finder**

The NI Example Finder is easy to use. After selecting the example you want, you click on the mouse two consecutive presses for LabVIEW to load the example.

## Fixed Errors in LabVIEW

When there is a mistake in linking the units in VI, the implementation key appears in this image  and this means that VI contains errors and is not executable.

To implement VI, errors must be fixed first. To learn about errors and find their causes, press the key  or choose **View >> Error List**, as shown in figure 2.73.

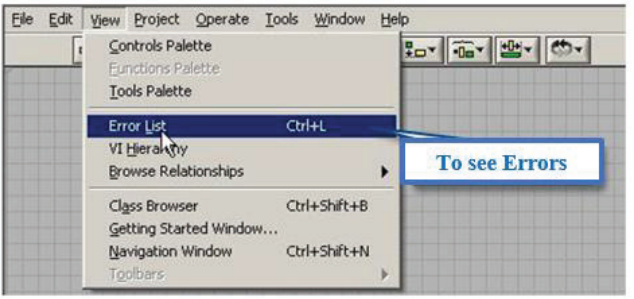



**Figure 2.73: Error List**

This is to display a window with an Error list, as shown in figure 2.74.

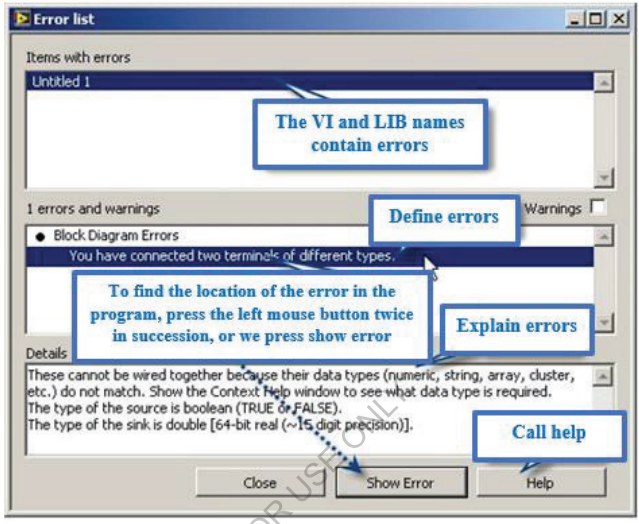



**Figure 2.74: Error List Window**

By clicking on the Help key in the previous menu, we display the topic in **LabVIEW Help**, which explains the error in detail and how to fix it step by step.

One of the most common errors is to link two different objects in the **Data Type**, as shown in figure 2.75.

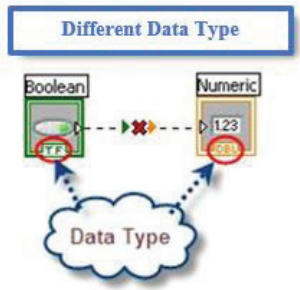



**Figure 2.75: Different Data Types Objects**

All errors must be fixed and all links must be verified so that the implementation switch becomes this way  that is, VI is enforceable.

## VI Test

If all errors are corrected and the VI becomes executable but the VI is not implemented as expected or it gives an error in the results then the **VI implementation** must be **tested**. LabVIEW provides us with several tools to track program implementation and testing, including:

### 1- Executing the program using. Highlight Execution

By clicking on the key  in the Toolbar in Block Diagram (turns to  ) and then by pressing the Execute key  , the program execution movement will be displayed slowly with the flow of data flow during the program execution, as shown in figure 2.76.



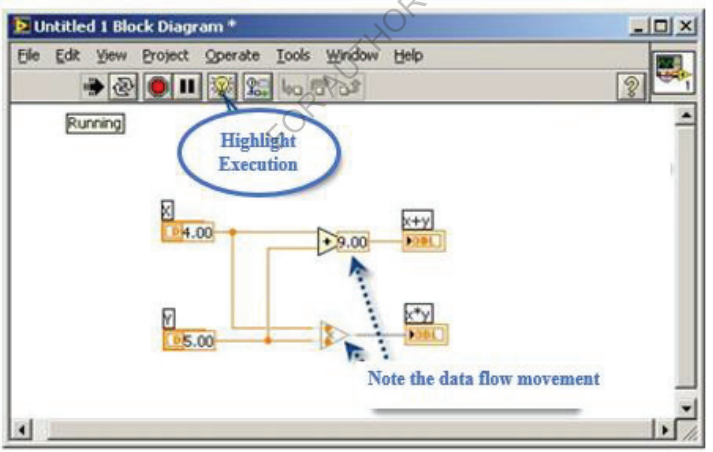



**Figure 2.76: Highlight Execution**

### 2-Implementation with the single-stepping system:

Sometimes you need to implement Block Diagram as a step that is executed in every single Node step and this is known as **Single-Stepping** or a **step-by-step system** implementation.

The **Step Into** , **Step Over** , and **Step Out** keys in the Toolbar in **Block Diagram** are used step by step.

To start the program implementation in the step system, press one of the previous keys.

By clicking the **Step Into** and **Step Over** keys, the current Node is executed and the next Node is ready to be executed in the next step.

When you press the **Step Over** key, Node is executed in one step and is not entered. For example, if Node is a SubVI, it will execute completely in one step and move to the next Node in the next step without entering into the internal details in executing that SubVI.

Either by pressing the **Step Into** key, you will be entered into SubVI, step by step.

**Step Out** is used to end **Block Diagram**, **Structure**, or **VI** that is currently executing Press the key to execute at the next step or complete the execution completely.

By moving the mouse pointer over those keys at any point, we will see text indicating what will happen if we press these keys.

During the **step-by-step system** implementation, the **Pause** key can be pressed to complete program execution in the normal way.

In figure 2.77 that shown the Step Over Execution.

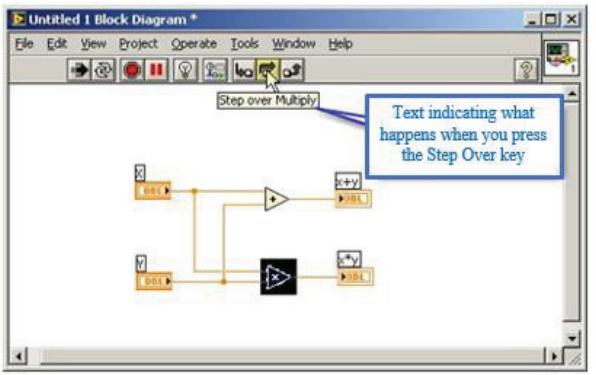



**Figure 2.77: Step Over Execution**

When performing **step by step** and activating the **Highlight key**, an arrow appears on the **SubVI** icon when it is in the execution status, as shown in figure 2.78.



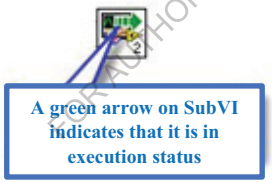



**Figure 2.78: Execution Status**

## 3- Use Probe

Probe is used to know the data on the tracks at the time of program implementation.

Probe can be set using the tool  , as shown in figure 2.79.

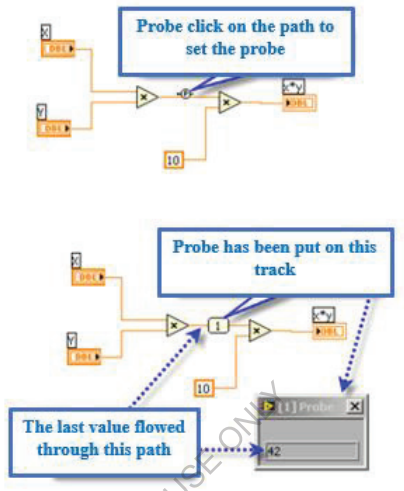



**Figure 2.79: Use Probe**



- It uses Probe in the event that your program is **complex** and **produces incorrect results**.
- Knowing the data on any path in the program during the implementation of the program
- More than one Probe can be placed in the program. Whether you use the Front Panel or the Block Diagram, the Probe is always visible in the foreground.
- Probe can be used with Highlight implementation, step-by-step implementation, or with Breakpoints (we will explain it in the next paragraph) to find out where the error is in the program.
- The values that appear in a window cannot be modified. Probe
- The implementation of VI is not affected by the existing Probes.

**There is more than one type of Probe:**

## 1- Generic Probe

Used to display data that flows along a path. In figure 2.80 that shown the way to get the Generic Probe.

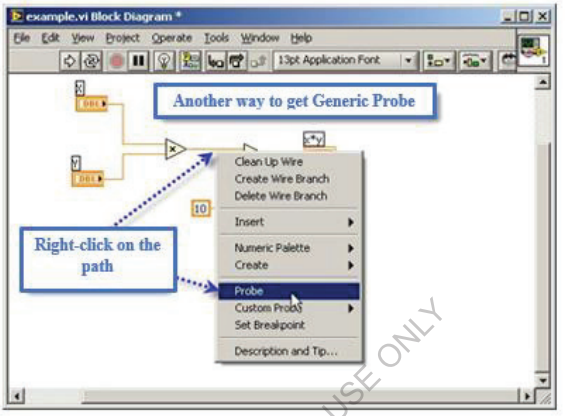



**Figure 2.80: The Way to Get the Generic Probe**

**Generic Probe** can be obtained by pressing the right click on the path and choosing Probe. This is if it was not previously defined by Custom Probe (it will be explained in this lesson) for the data type on this path, as shown in figure 2.81.

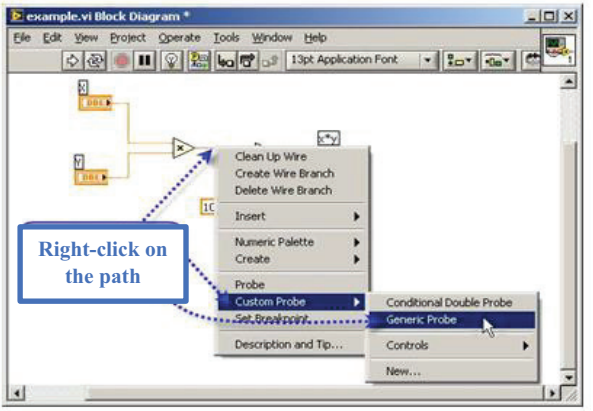



**Figure 2.81: Select Generic Probe**

## 2- Use the Indicator in Probe

Indicator in Probe can be used to display data on tracks during program execution.

In this case we use the appropriate Indicator for the data.

For example, a chart can be used in Probe to see data for a signal on a path. To put the Indicator in Probe, press the right mouse button on the path and select **Custom Probe >> Controls**, then choose the desired one, as shown in figure 2.82.

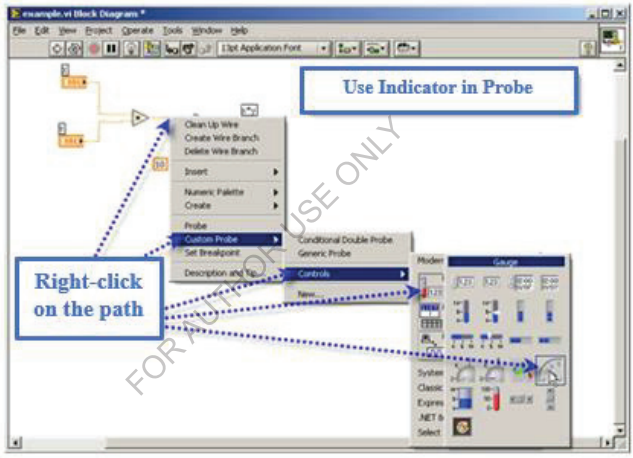



**Figure 2.82: Indicator in Probe**

In Figure 2.83 it shows the done probe.

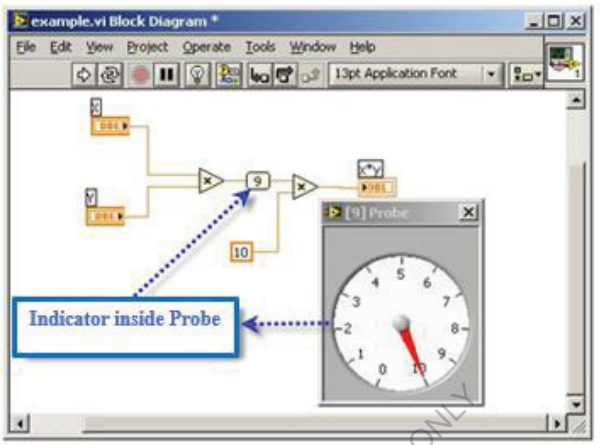



**Figure 2.83: Indicator Inside Probe**



### 3- Supplied Probe

It is an VI that gives more comprehensive information about the data on the path. For example, **VI Refnum Probe** gives information about the **VI name**, its **storage path**, and its **Hex value**.

This Probe can interact with the software. For example, a condition can be set when the data on the path is met and the program execution is stopped, as shown in figure 2.85(a,b). **Supplied Probe** appears at the **top of the Customer Probe** list, as shown in figure 2.84. It varies depending on the type of data on the path.

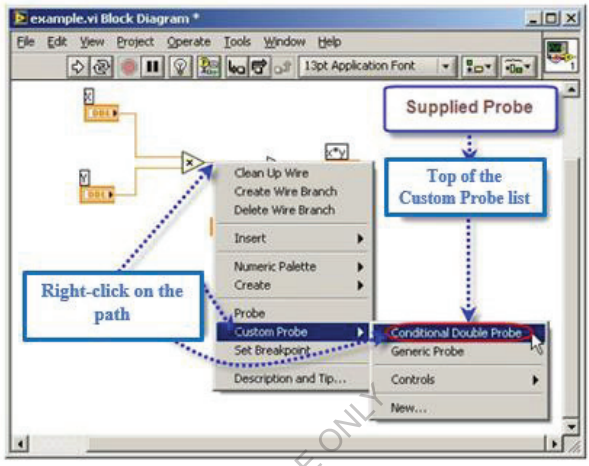
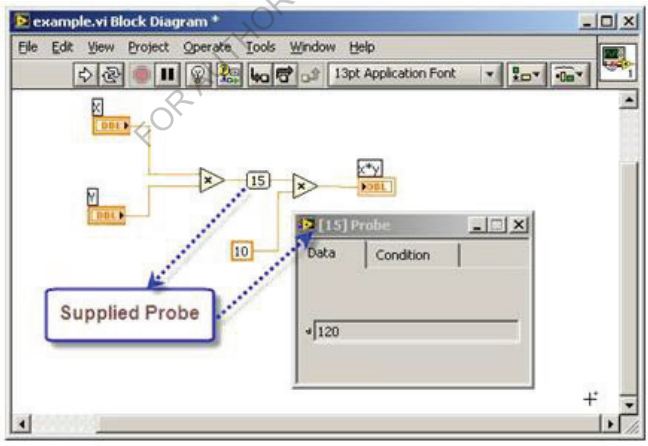
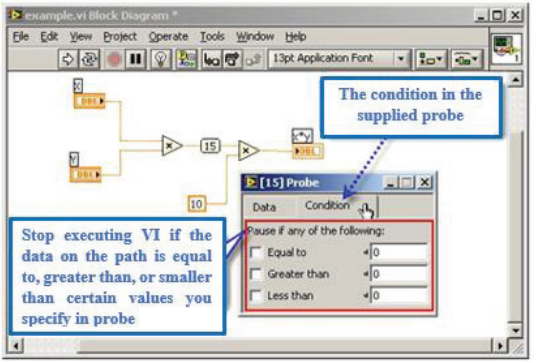



**Figure 2.84: Supplied Probe**




**(a)**




**(b)**

**Figure 2.85: The Condition in the Supplied Probe**



## 4- Custom Probe

You can make your own **Custom Probe** by modifying an existing Probe or creating a new Probe. To create a Probe, **right-click** on the track and then choose **Custom Probe >> New**, as shown in figure 2.86. It shows us a set of windows that define the Probe specifications that we want.

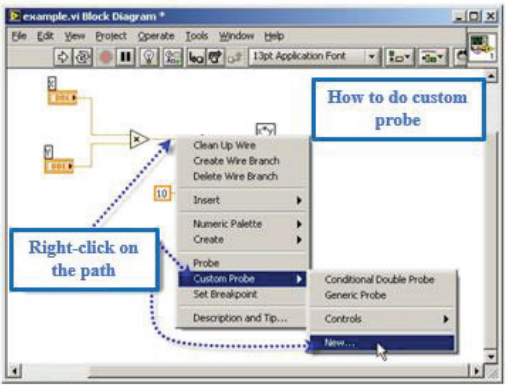



**Figure 2.86: How to do custom probe**

**Note:** When a special Probe is done for a specific data type, this becomes the **default Probe** for this type of data that appears when we press the **right button** on the path and choose **Probe**.

## 4- Breakpoints

The breakpoints are set by the  **Breakpoint tool** in **Block Diagram** to **stop VI execution** when execution reaches these points, as shown in figure 2.87.

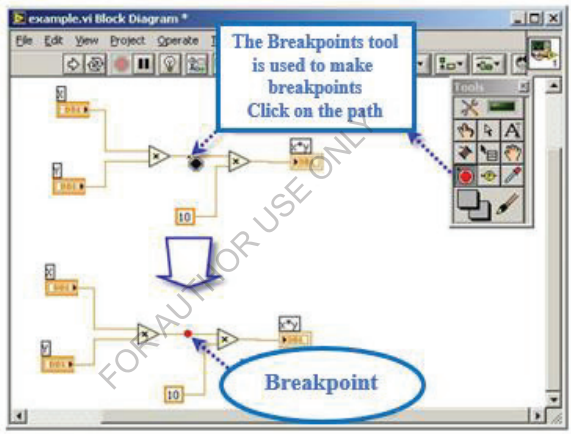



**Figure 2.87: Breakpoints**

Breakpoint can be set to **VI**, **Node**, or **Wire path** in Block diagram, as shown in figure 2.88.

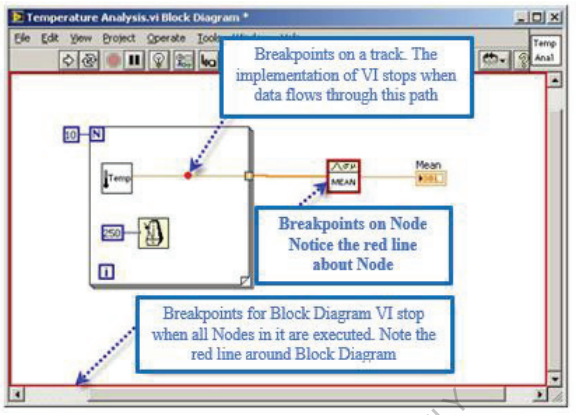



**Figure 2.88: Breakpoints Set to VI, Node, and Wire path**

When VI stops at a breakpoint, the **Block Diagram** window appears in the foreground and a path flash or Node containing the breakpoint occurs, as shown in figure 2.89.

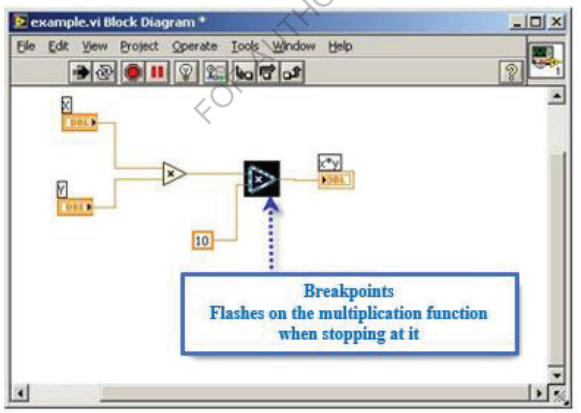



**Figure 2.89: VI Stops at a Breakpoint**

When you move the Breakpoint tool to an existing stop, this tool appears as such , which means that there is a Breakpoint stop by clicking this tool on the breakpoint, that point is removed, as shown in figure 2.90.

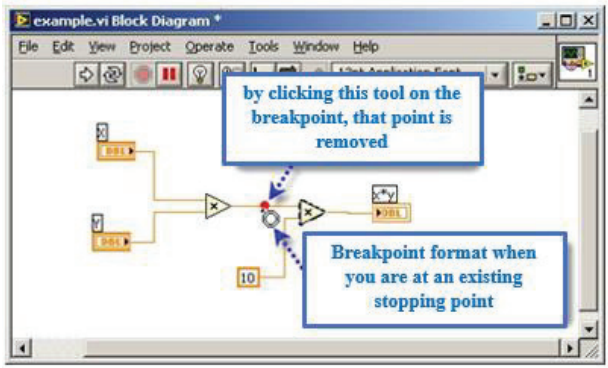



**Figure 2.90: Existing Breakpoint**

When VI stops at Breakpoint breakpoint The Pause key in Toolbar is displayed as this .

In this case, one of these steps can be taken:

1- Completing the program implementation with the **step-by-step** system using the keys in the Toolbar.

2- Use Probe to check path values.

3- Changing the values of the Controls in the Front Panel

4- Press the Pause key to complete the program execution until the program reaches another stopping point or until the program has finished executing.

CHAPTER TWO
LABVIEW LEARNING LESSONS

# Lesson Four: Create a SubVI

## Aim of the lesson

**1)Learn how to create and use SubVI**

**2)Learn about the VI structural structure**

**3)Learn how to use the Icon Editor**

**4)Learn how to setup Connector Pane**

**5)Learn how Documentation of the VI works**

## What is SubVI?

It is a complete VI for use in other VI Block Diagrams.

It is the same as Subroutine in other programming languages like C, as shown in figure 2.91.



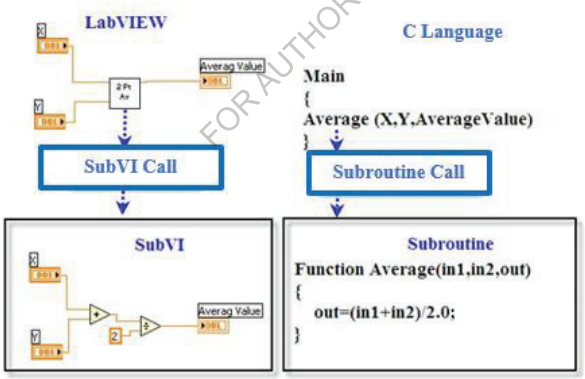



**Figure 2.91: Subroutine in LabVIEW and C Language**

This is one of the features of LabVIEW, so any VI you have created can be reused in another VI to create a larger function.

VI can contain an unlimited number of SubVIs.

This SubVI also can contain many SubVIs.

This is called the Hierarchical Nature in LabVIEW, so all SubVI can also contain SubVIs.

## The Icon and Connector

After building VI's Front Panel and Block Diagram it should be built its Icon and Connector Pane for it to use as SubVI.

### 1- The Icon

We learned from the first lesson that every VI has an Icon. This icon is in the upper-right corner of the Front panel or Block Diagram window, as shown in figure 2.92.

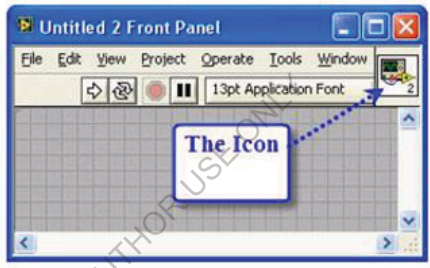





**Figure 2.92: The Icon**

This icon represents the VI and this icon can contain a graphic or writing, or both.

The initial form of the icon is  and the number on the icon indicates the number of the new VI that has been opened since the operation of the program.

LabVIEW in the sense that if you do another new VI, the icon will be like this  . **Note that the number has increased by one**.

This icon can be modified to the shape you want:

• **Right-click** the icon and choose **Edit Icon**.

• or by **double click** on the icon.

In figure 2.93 that shown the icon modifies.

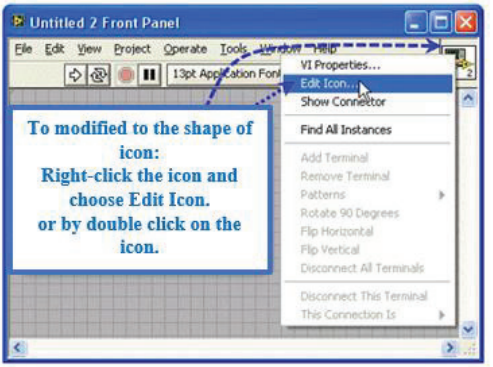



**Figure 2.93: The Icon Modifies**

• or by choice **File >> VI Properties**, then choose General and press Edit Icon, as shown in figure 2.94.



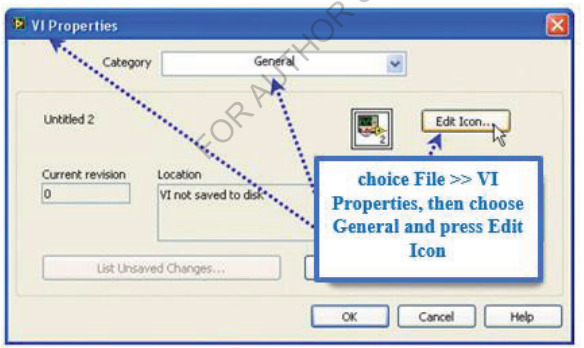



**Figure 2.94: The Icon Modifies**

All of the above options lead to the opening of Icon Editor. This program allows modifying the appearance of icons.

## 2- Icon Editor

With this program you can design the Icon you want, as shown in figure 2.95.

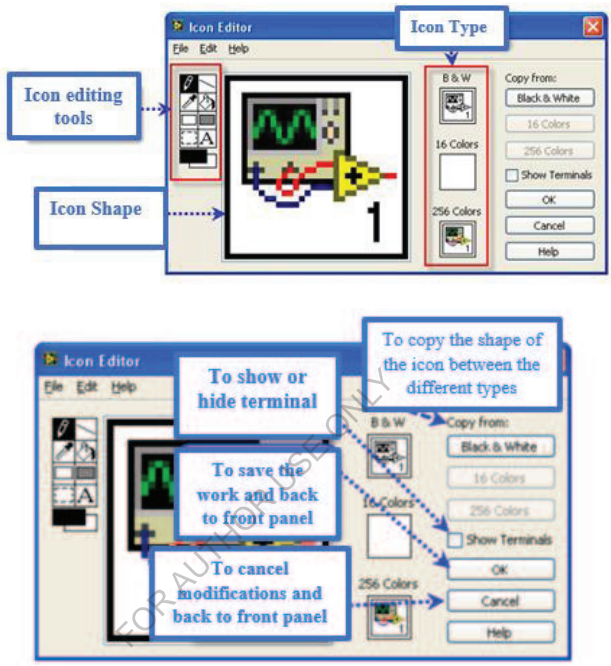



**Figure 2.95: Icon Editor**

There are several tools in the Icon Editor program, as follows:

Pencil tool to draw or erase point by point.

Line tool to draw lines. If you want the line to be horizontal or vertical, press the Shift key while drawing the line.

The Color Copy tool to copy a color in Icon to be the writing color.

Fill tool to fill a specific area of writing color.

Rectangle tool to draw a blank writing rectangle. By pressing this tool two consecutive presses, an outer window is drawn for the entire icon writing color.

Filled Rectangle tool for drawing rectangular writing color and background color shading.

Select tool to choose a specific area to erase, copy, or move to another location.

By pressing this tool two consecutive presses and then pressing the Delete key, the entire icon content is erased.

Text tool to write on the icon. By pressing two consecutive presses a window opens with the properties of the line and these properties can be modified.

To show and choose the writing color and the background color (the writing color is the front rectangle and the background color the rectangular back) and by pressing which two rectangles the color is chosen.



### 3- Connector Pane

In order to use **VI** as a **SubVI** the Connector Pane must be set up for it. **Connector Pane** is a set of **Terminals**. These Terminals represent Controls and Indicators found in VI. The **Connector Pane** determines the inputs and outputs of the VI to be used as SubVI. To display the Connector Pane for VI, press the **right click** on the icon and choose **Show Connector**, as shown in figure 2.96.

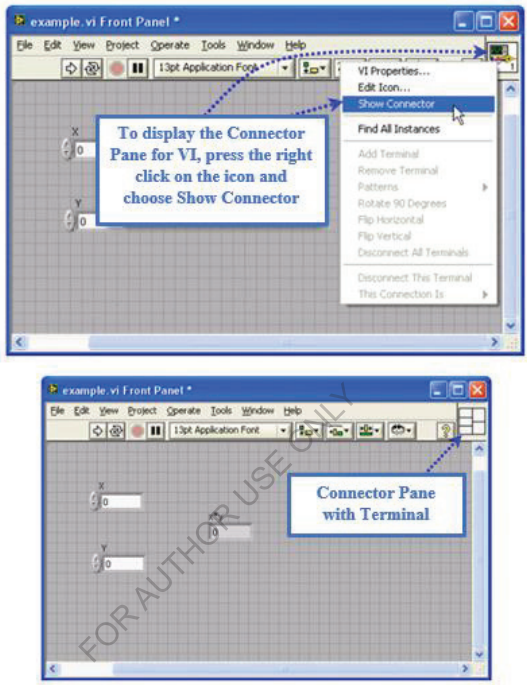



**Figure 2.96: Connector Pane**

Each rectangle in the Connector Pane that expresses terminal. This can represent an input or output depending on whether it is connected to the Control or Indicator as we will see. In figure 2.97 shown the visualization can help zoom the image.

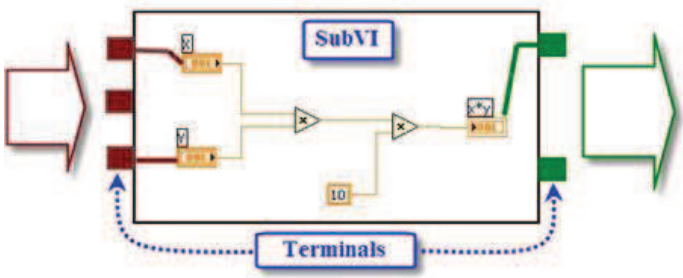



**Figure 2.97: Visualization Can Help Zoom the Image**

Note from the previous figure that Terminals may not be attached to any Control or Indicator. Also, there can be a Control or Indicator that does not have a corresponding terminal in the Connector Pane.

## Terminals Patterns

The organizational form for Terminals or the number of Terminals can be changed by **right-clicking** on the Connector pane and then selecting **Patterns** and then selecting the Pattern you want, as shown in figure 2.98.



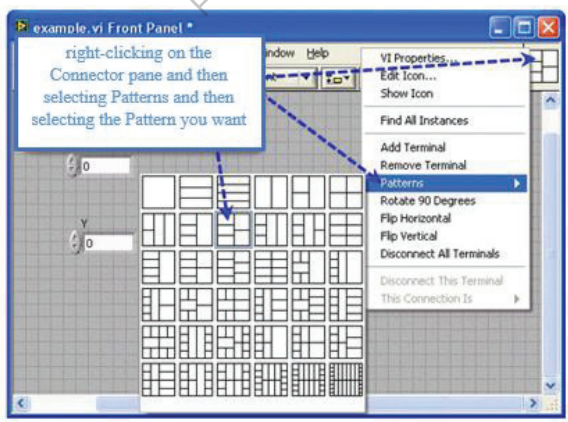



**Figure 2.98: Change Terminal Patters**

It is preferable to choose a pattern so that there are additional terminals that are not associated with any Control or Indicator at this time until the modification is easy if you need to add any Control or Indicator in the future. The maximum number of Terminals is **28**, but it is preferable to use more than **16**. Terminals arrangement can be changed in the Connector Pane by pressing the right mouse button and choosing **Flip Horizontal** or **Rotate the 90 axis** or **Rotate 90 degrees**. Terminal can also be added or deleted, as shown in figure 2.99.

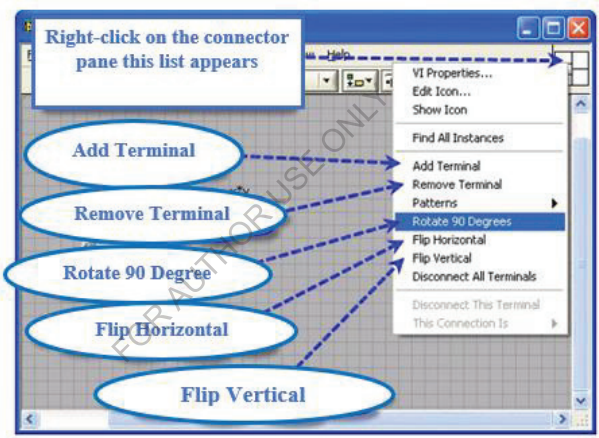



**Figure 2.99: Connector Pane List**

## Connect Terminals to Controls and Indicator

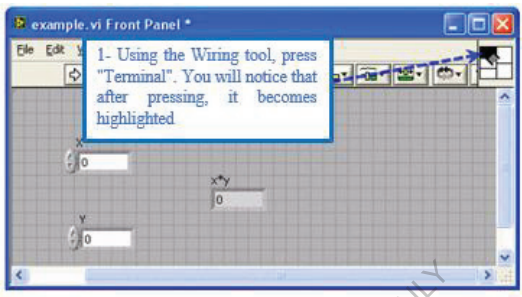
After the Connector Pane is set to VI, the Terminals in the Connector Pane should be linked to the Controls and Indicators on the Front Panel.

Before explaining how the link works, we confirm that it is preferable that the **Inputs** be to the **left** in the Connector pane and the **Output** to the **right** to be easier to use and avoid complication.

How to link:

Show the Front Panel window then

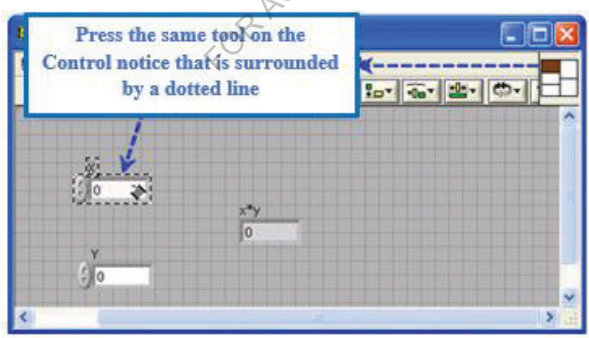
1- Using the Wiring tool, press "Terminal". You will notice that after pressing, it becomes highlighted, as shown in figure 2.100 (a).




**(a)**



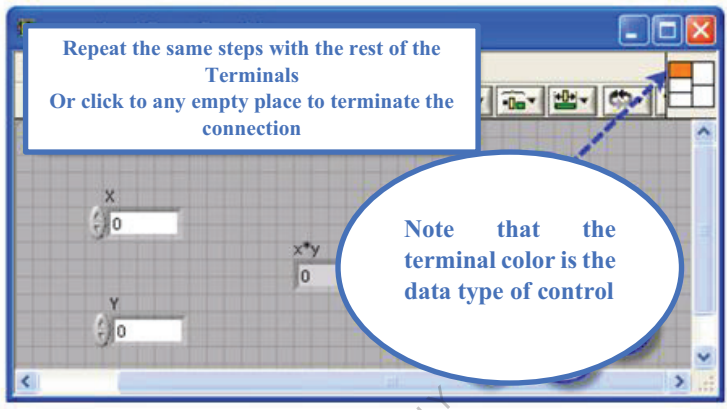
2- Press the same tool on the Control or Indicator you want to link to. You will notice that the Control or Indicator is surrounded by a dotted line, thus Terminal is linked, as shown in figure 2.100 (b).




**(b)**

3- Repeat the same steps with the rest of the Terminals, as shown in figure 2.100 (c).

4- To end the fastening process, press anywhere empty.




(c)

**Figure 2.100: Connection Terminals to Controls and Indicators**



**Notes:**

• Although the Wiring tool is used in the linking, there is no line that connects Terminals with Controls or Indicators

• In the linking process you can choose Control or Indicator first and then Terminal

• If an error occurs in the linking process, you can remove all links between Terminals and Controls or Indicators by pressing the right button and choosing Disconnect all Terminals.

• Or, you can remove the link for one Terminal by pressing the right button and selecting "Disconnect This Terminal."

In figure 2.101 that shown the remove all terminal or single terminal.

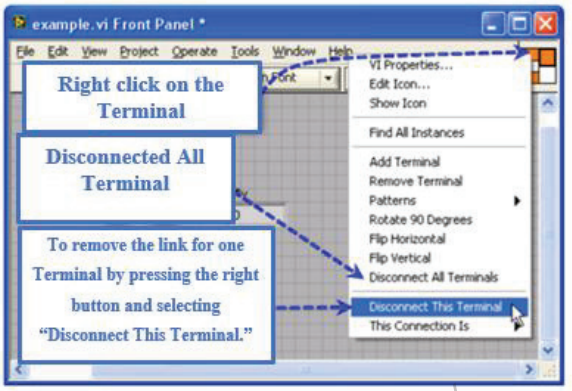



**Figure 2.101: Remove All Terminal or Single Terminal**

To return from the **Connector Pane** to the icon shape, press the **right click** and choose **Show Icon**, as shown in figure 2.102.

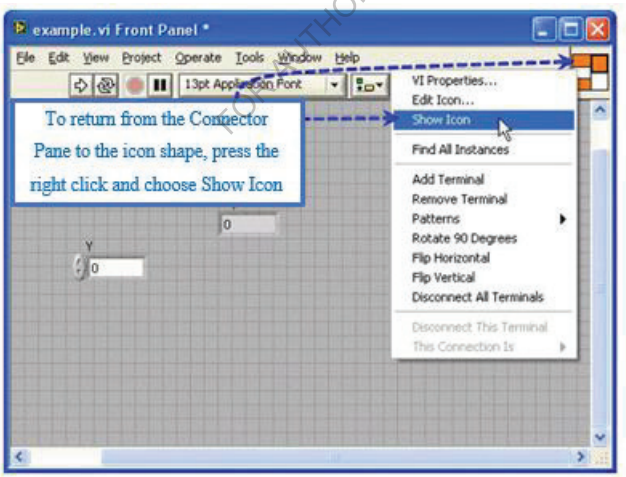



**Figure 2.102: Return to Icon Shape**

## The Help Window

When the Context Help window appears and moves the cursor on SubVI, the information available for this SubVI appears. This information is the same as the definition of SubVI and Terminals present, and is this Required Terminals, Recommended, or Optional. The window also shows some information about this. Terminal. This is all we have seen in the previous lesson. But what matters to us now is how to prepare that information for a new VI that we will use later as SubVI.

### 1- Specify the type of terminal

Terminal type can be selected by pressing the right click on the terminal in the Connector Pane and choosing This connection is then choosing the **Terminal type**.



**Note:**

The default choice for Terminal type is **Recommended**.

To make connection of this terminal necessary for VI to be executable make its type **Required**.

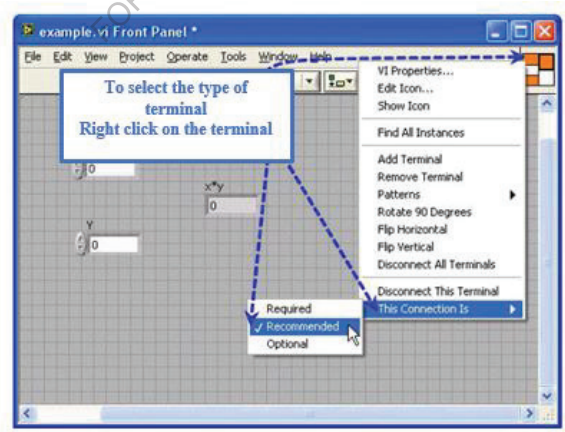



**Figure 2.103: To Select the Type of Terminal**

## 2- Write a Description (Documentation) of the VI

To write a VI description, press the right click on the icon, then select VI Properties, as shown in figure 2.104.

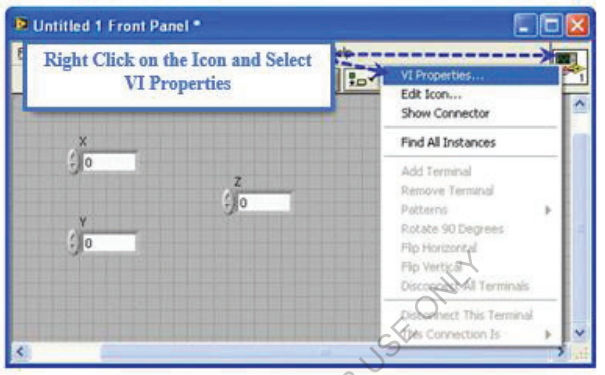



**Figure 2.104: VI Properties**

To show us this window in figure 2.105.

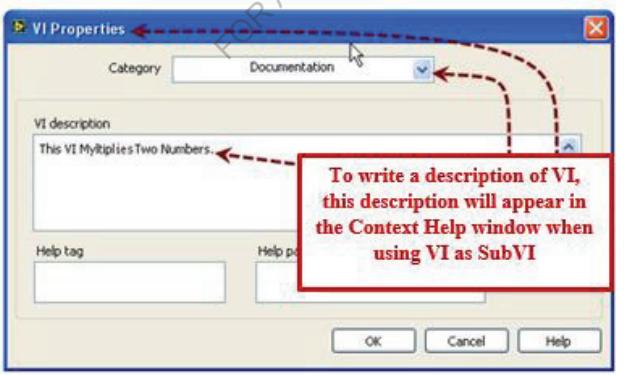



**Figure 2.105: VI Properties Window**

In figure 2.106 that shown the context help of subVI.

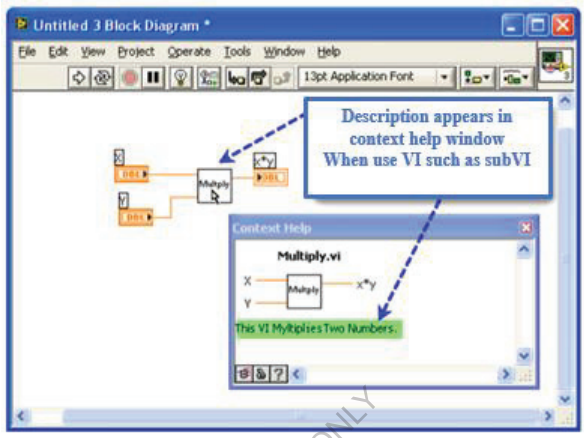



**Figure 2.106: Context Help of SubVI**



## 3- Writing a description for the Control and Indicator

When designing a new VI, it is best to do a description for each Control or Indicator in it. By right-clicking on Control or Indicator and choosing Description and Tip, as shown in figure 2.107.

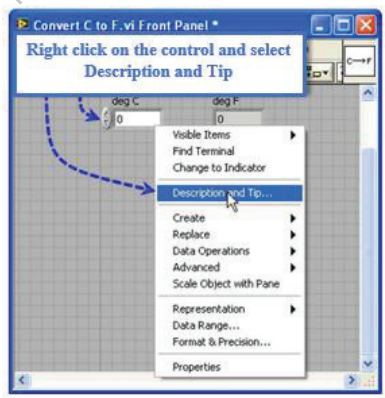
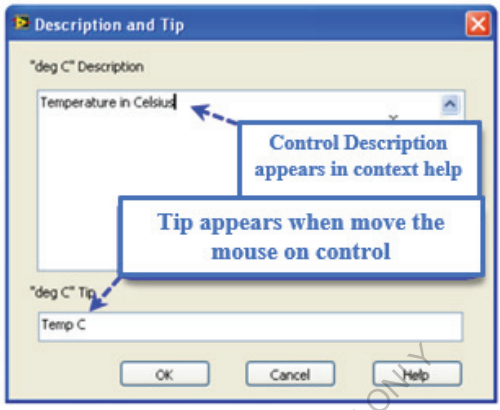
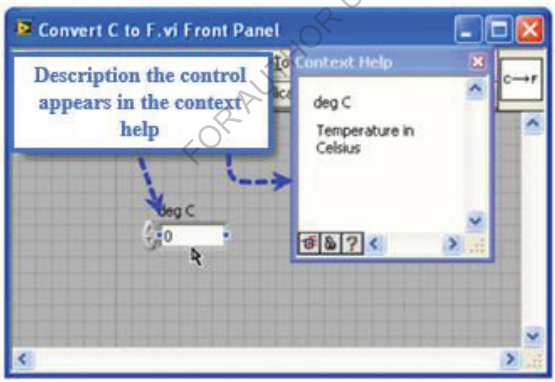



**Figure 2.107: Description and Tip**

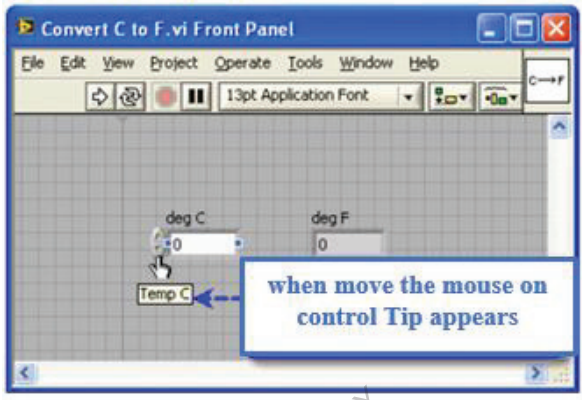
In figures 2.108 (a, b, c) that shown the Description and Tip design.




**(a)**




**(b)**




**(c)**

**Figure 2.108: Description and Tip Design**

CHAPTER TWO
LABVIEW LEARNING LESSONS

# Lesson Five: Structures

## Aim of the lesson

**1) Learn about Repetitive Functions While Loop and For Loop**

**2) Get to know the Shift Registers and Feedback node**

**3) Learn about Case Structure**

**4) Learn about Dialogs and the different ways to output messages to the user during program execution.**

**5) Learn about Flat and Stacked Sequence Structure**

**6) Learn ways to make time delays and calculate time in programs.**

**7) Learn about Formula Node and Expression Node to write and calculate mathematical equations within a program**



## Introduction

Structures is one of the most important types of Nodes used in Block Diagram

In this lesson, we will learn about basic structures in LabVIEW:

- For Loop
- While loop
- Case Structure
- Sequences Structure
- Formula Node

## Loops

Loops are used to duplicate part of the code. LabVIEW has two main types of Loops: For Loop and While Loop.

**For Loop** It is to repeat the execution of a specific part of the code a specified number of times. **Example:** Repeating two numbers five times

**While Loop** It is to repeat a specific part of the code until a certain condition is met, that is, until this condition is True or False, depending on the type of condition used as we will see.

**While Loop** and **For Loop** are listed from **Function Palette >> Programming >> Structures**, as shown in figure 2.109.

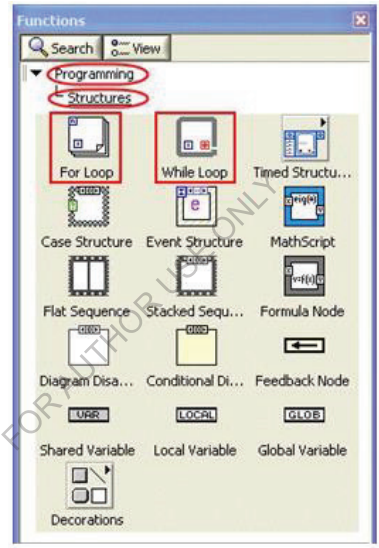



**Figure 2.109: Structure List**

## For Loop

For Loop is used to repeat the code inside of it a specified number of times. The code inside of For Loop is called **Subdiagram**.

With for loop there are two terminals:

**Count Terminal** [N]**:** It determines the number of executions inside a For loop, and the number is connected to it from outside. If the number is connected to zero, it will not execute inside.

**Iteration Terminal i:** It contains the number of times it has been executed. The first time that the code is executed, the value is zero, and the second time it is one value, and so on until it reaches the number N-1.

This takes the values from 0 to N-1 where N equals the number connected to Count Terminal (the number of times the code will execute). In figure 2.110 that shown the for loop.

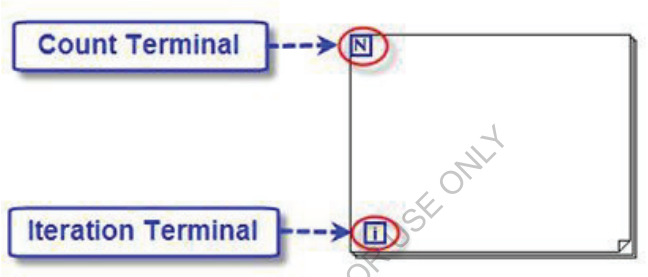




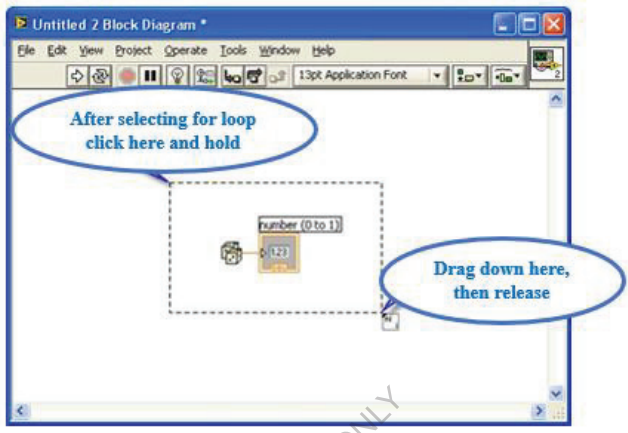

**Figure 2.110: For Loop**

**Note:**

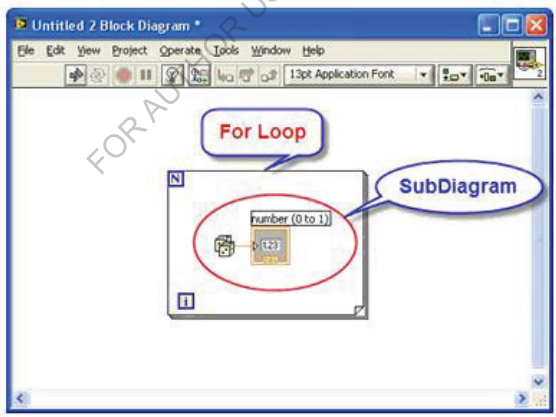
The Data Type of each N and i which is Long Integer, they take the correct values from zero to 232-1. Note that they are blue.

## How to include For Loop

As we mentioned there is For Loop in **Function Palettes >> Programming >> Structures.** Choose for Loop for the cursor to look like this . Then start the for loop drawing where you want it, as shown in figure 2.111.




(a)




(b)

**Figure 2.111: How Include For-Loop**

• **For Loop dimensions** can be changed by using the Positioning tool by moving them on the Loop frame to make the pointer an arrow shape so we press the pointer and drag the frame in the direction we want, as shown in figure 2.112.

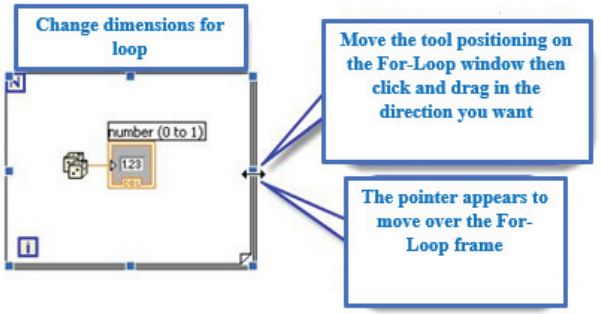



**Figure 2.112: Change Dimension For-Loop**

• Any function, SubVI, or any Structure can be inserted into For Loop, as shown in figure 2.113.

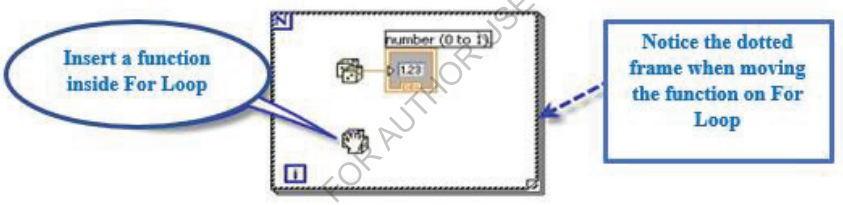





**Figure 2.113: Insert a Function Inside For-Loop**

• **For Loop** can also be moved and moved by **Positioning Tool**.

• When moving **For Loop**, it is moved together inside.

• **Note** that if you move For Loop to cover SubVI or a function, a shadow will appear for this function to warn you that it is not inside **For Loop**, as shown in figure 2.114.

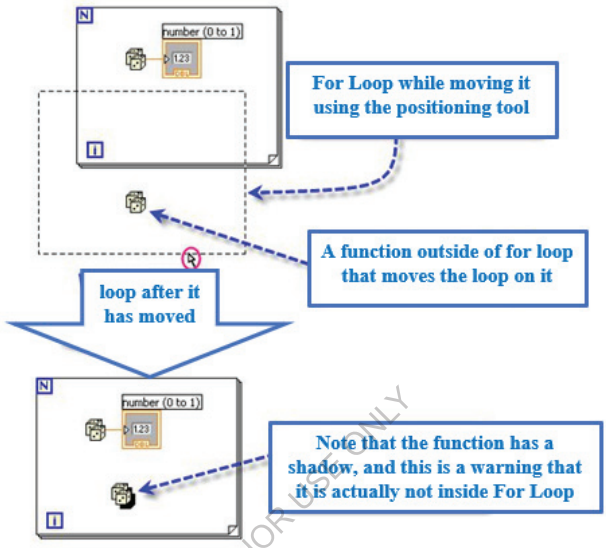



**Figure 2.114: Function outside the For-Loop**



## Auto Grow feature

Each Structure, including For Loop, has an **Auto Grow** feature, which is that **For Loop automatically** expands to nine everything inside it, and it's all visible.

This feature can be activated or deactivated by pressing the **right click** on the Loop window and selecting or deselecting **Auto Grow**, as shown in figure 2.115.

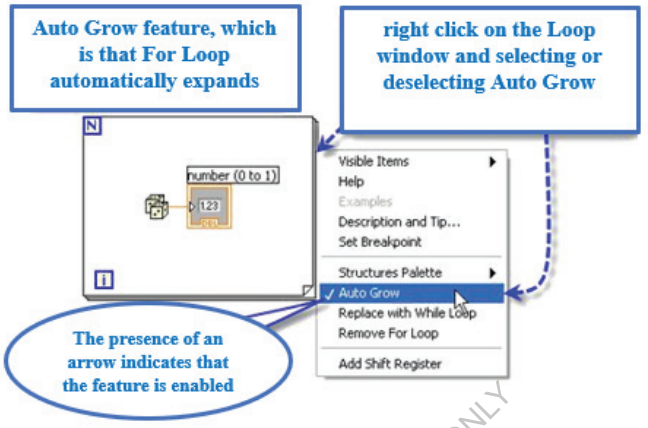



**Figure 2.115: Auto Grow Feature**



## Remove For-Loop

• If you want to delete For Loop inside it, choose For Loop and press Delete.

• To delete For Loop only and keep it inside, press the right click on the For-Loop window and choose Remove For-Loop, as shown in figure 2.116.

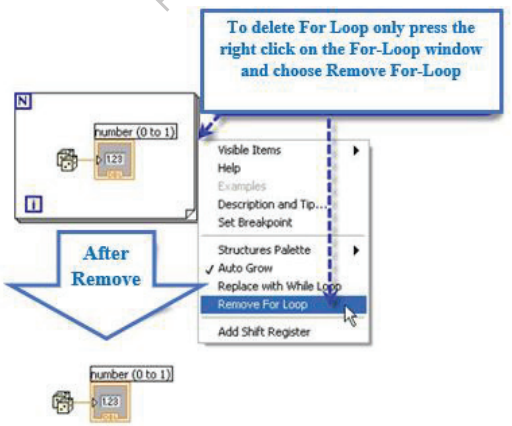



**Figure 2.116: Remove For-Loop**

## Numeric Conversion

We mentioned before that the **Data Type** for Count Terminal is **Long Integer**, i.e. a valid number between **0** and **232-1**. There are other types of **Data Type** for Control or Digital Indicator, such as **Double-Precision** and **Single-Precision**. If you connect two different Terminals in a Data Type, LabVIEW converts the numeric values between the two different types. Then LabVIEW places a point on the converted Terminal and this point is called **Coercion dot**, as shown in figure 2.117.

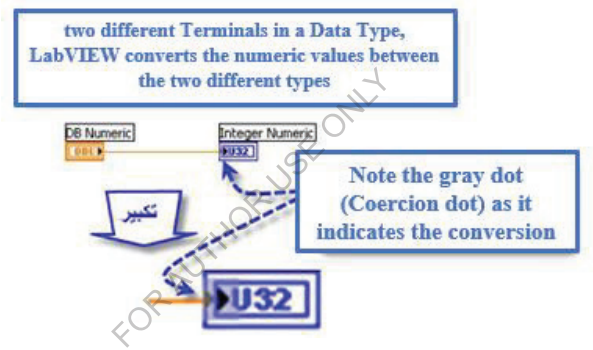




**Figure 2.117: Two Different Terminate in Data Type**

Also, if you connect a **Terminal** that has a different **Data type** to **Count Terminal** N, LabVIEW will perform the conversion, as shown in figure 2.118.

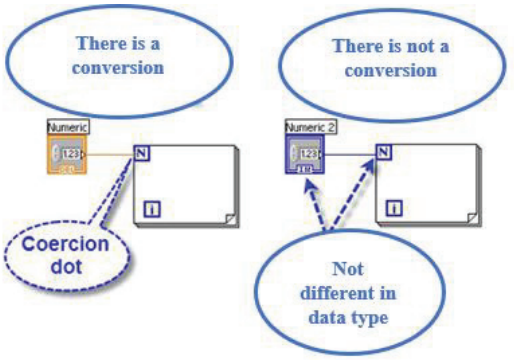



**Figure 2.118: For-Loop Different Data Type**

**To avoid this**

1- Change the terminal type data connected to the Count Terminal by right-clicking, selecting **Representation**, and **selecting I32**, as shown in figure 2.119.



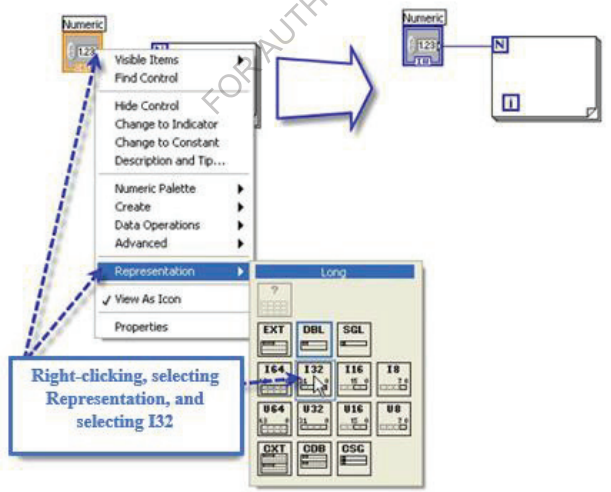



**Figure 2.119: Change Type of Terminal**

2- You can **create Constant** or **Control** that is automatically connected to the **Count Terminal** and has the same Data type by pressing the right click on the Count Terminal and choosing **Create >> Constant** or **Create >> Control**, as shown in figure 2.120.

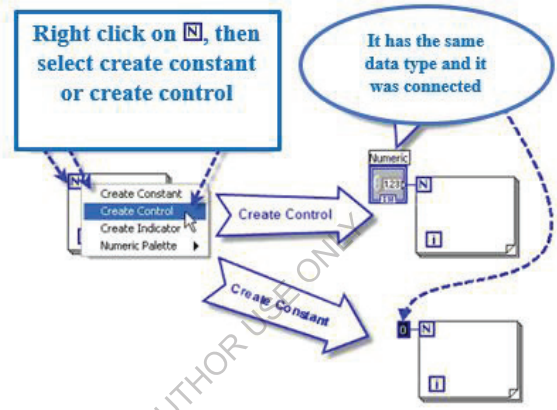





**Figure 2.120: Create Control or Create Constant to Count Terminal**

## While Loop

It is a Structure that repeats the implementation of the code inside it until a condition is met. In figure 2.121 that shown the while loop block.

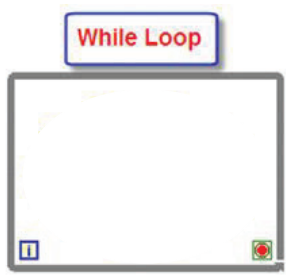



**Figure 2.121: While Loop**

While Loop has two terminals which are **Iteration Terminal** and **Conditional Terminal**.

### 1- Iteration Terminal

It is like the Iteration Terminal found in For Loop and it contains the number of times the code was executed inside While Loop. The first time that the code is executed, the value is zero, and every time the code is re-executed, it increases one.

### 2- Conditional Terminal

This takes the Boolean or False values. The execution is done inside While Loop until the entry to this terminal is either True or False, depending on the type of Conditional Terminal.

There are two types to this terminal:

#### a- Stop If True

The execution of an execution will stop inside While Loop if the internal value of this Terminal is True.

Each time what is inside is executed while Loop, then the value inside of this Terminal is checked, if True exiting while loop and if false repeat the execution of what inside while loop, as shown in figure 2.122.



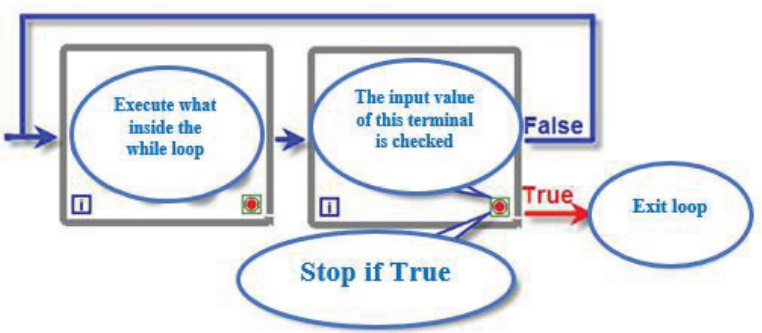



**Figure 2.122: Stop the Loop if True**

### b- Continue if True

Repeats what is inside **While Loop** if the value in this terminal is **True**.

Each time what is inside is executed **while Loop** then the internal value of this terminal is checked, if it is **True**, what is inside is executed **While Loop**, and if the value is **False** then exit while Loop, as shown in figure 2.123.

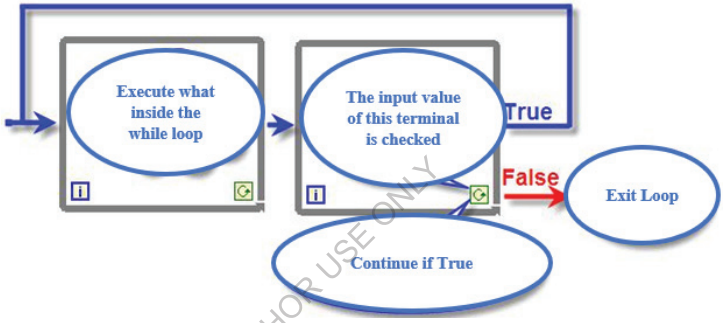



**Figure 2.123: Continue the Loop if True**



**Note that**:

• In any case, what is executed inside the loop is executed at least once.

• You can switch between two Terminals by **right-clicking** on terminal and choosing the other type, as shown in figure 2.124.

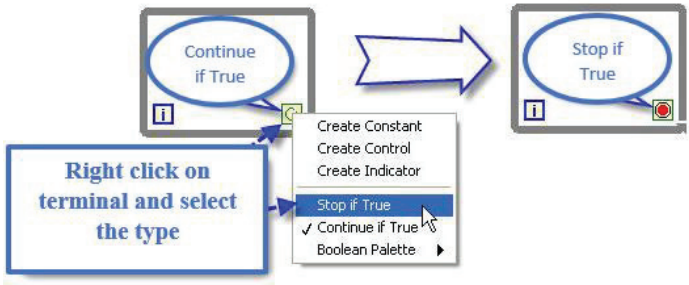



**Figure 2.124: Switch Between two Terminals**

**How to include While Loop:**

While Loop can be listed from

**Function Palettes >> Programming >> Structures**

Or from

**Function Palettes >> Express >> Execution Control**

In figure 2.125 that shown the include while loop.

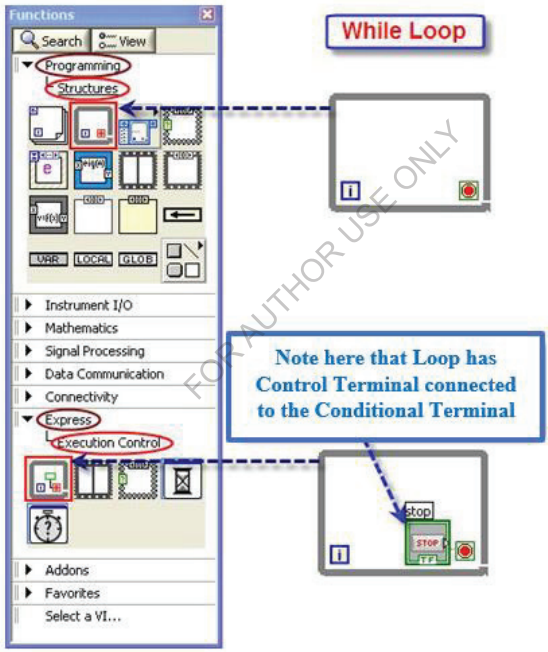





**Figure 2.125: Include While Loop**

Like **For-Loop** to insert **While Loop**, we select it from the **Function Palette** so the cursor appears like this and then we draw **While Loop**, as shown in figure 2.126.

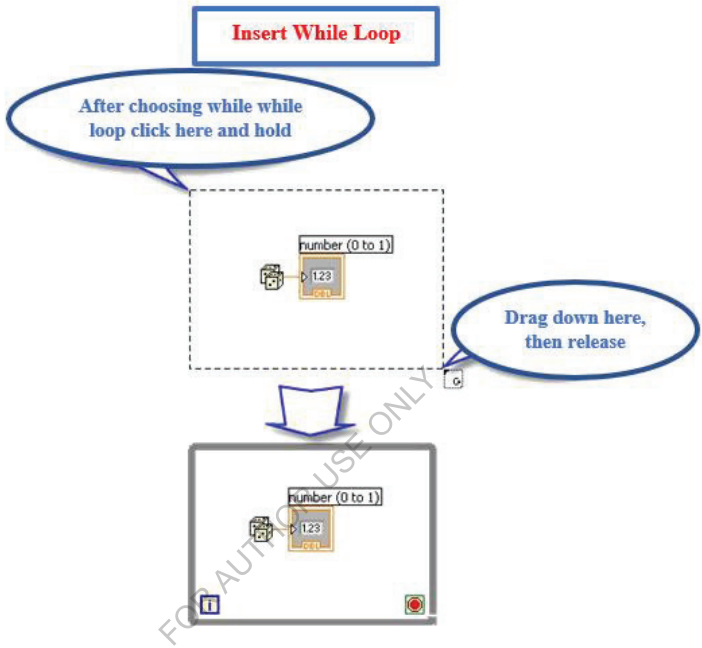





**Figure 2.126: Insert While Loop**

**While Loop is consistent with For Loop in:**

1- How to move, transmit and change the dimension of Loop.

2- Loop Deletion Method.

3- Auto Grow feature.

**Notes on using Terminals with Loops:**

Data is **entered** and **exited** to and from Loops through a small box called **Tunnel**, as shown in figure 2.127.

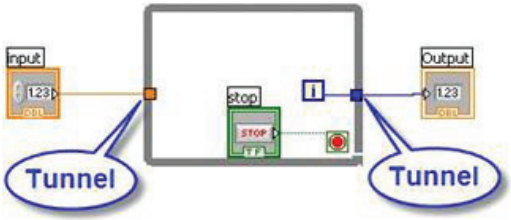



**Figure 2.127: Tunnel in While Loop**

According to the **Dataflow** concept, the data inside the Loop is entered into the Loop before the execution of what is inside the Loop for the first time. The data is exited from the Loop after the execution of the inside of the Loop last time, as shown in figure 2.128.

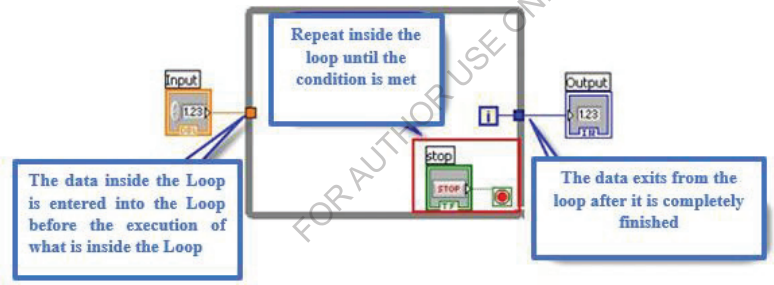





**Figure 2.128: Data Output from While Loop**

## Shift Registers and Feedback Node

When dealing with Loops, either for For-Loop or While Loop, you will find that you need to transfer data from the previous times that what has been done inside the Loop has been executed until the time that what is inside Loop is executed.

To clarify, we will call the implementation of what is inside the Loop as the name of an implementation, and we will give it a number indicating its order in implementation. For example, the first time that what is implemented inside the Loop is called implementation 1 and the second time we call it implementation 2 and so implementation 3, etc.

Often, we need to transfer data from one implementation to the next, such as transferring data from implementation 1 to implementation 2 and so on.

There are two ways to obtain data from previous implementations: **Shift Register** and **Feedback Node**.

## Shift Registers

**Shift Register** moves data from one implementation for For-Loop or While Loop to the next one. The **Shift Register** consists of these two formats ⏶ and ⏷, as shown in figure 2.129.

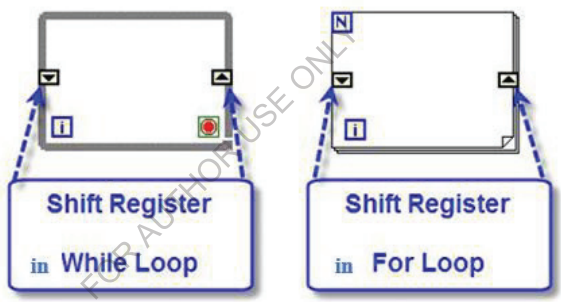





**Figure 2.129: Shift Register**

Data is transferred ⏷ from the previous implementation to the current implementation, or data is transferred from outside the Loop to the first implementation.

The data is transferred ⏶ from the current implementation to the next implementation, or the data is transferred from the last implementation to the outside of Loop.

In figure 2.130 that shown the shift register work.

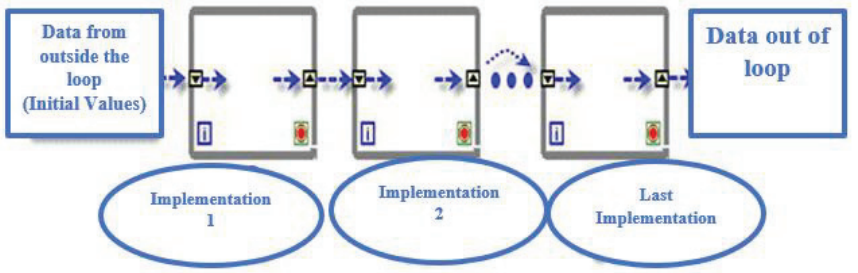



**Figure 2.130: Shift Register Work**

## How to add Shift Register

**Shift Register** is added by **right-clicking** on the Loop window and select Add, as shown in figure 2.131.

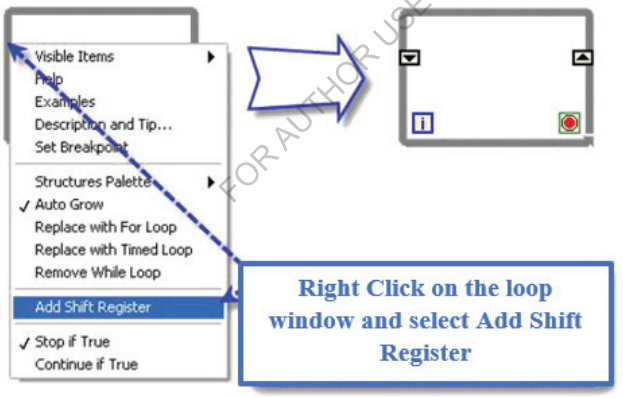





**Figure 2.131: Add Shift Register**

More than one Shift Register can be added in the same way, as shown in figure 2.132.

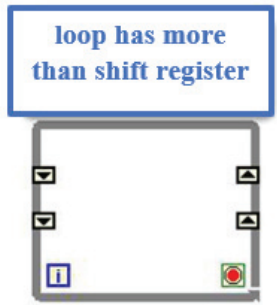



**Figure 2.132: Insert more than one shift Register**

**Shift Register** can transfer any type of data **number**, **Boolean**, **text** or **array**, but the data type must be connected to ▼ and ▲ one. In figure 2.133 that shown the double precision data type connected to shift register.



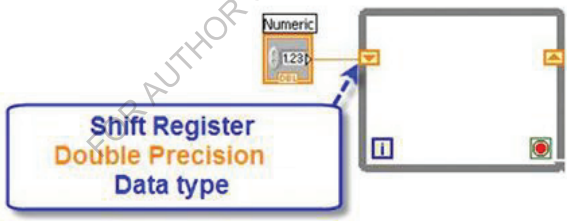



**Figure 2.133: Double Precision Data Type Connected to Shift Register**

## Initial values for the Shift Register

It is the Shift Register values before entering Loop Execution.

### How to set the Initialize Shift Register values

You can set the **Initial Shift Register** values by **Constant** or **Control** in the Terminals on the left end, as shown in figure 2.134.

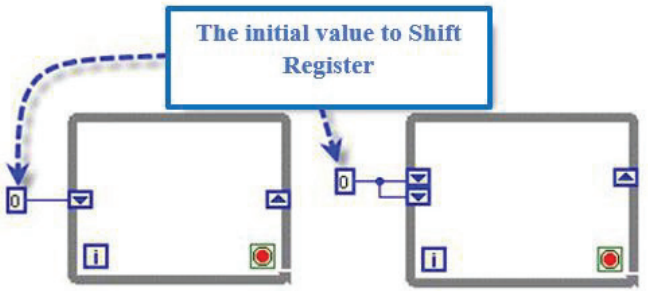



**Figure 2.134: Add Initial Value to Shift Register**

**What happens if the Shift Register's initial values are not specified?**

The Shift Register will take default values for the type of data the Shift Register transmits, for example if that data is Numeric, the default values will be zero and if the data are Boolean numbers, then the value will be **False**.

The **Shift Register** takes these initial values before the program is executed for the first time, but after the program is completely finished, the **Shift Register** retains its **last values** for it to be the initial values when the program is executed for the second time.



## Feedback Nodes

The **Feedback Node** is used in **While Loop** or **For-Loop** when we want to link a **SubVI** output, a function or a SubVIs group to the same input as the **SubVI**, **function**, or **SubVIs group**, and **functions**.

As we know from the **Dataflow concept**, we will not be able to connect an output to an input for the **same function** or **SubVI**, because it will not execute and output data on its output unless the data is ready on its inputs and the input of the source does not occur, as shown in figure 2.135.

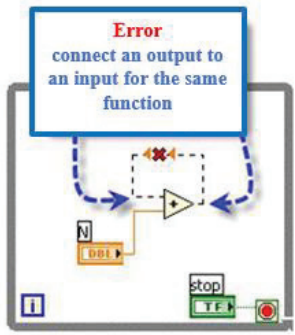



**Figure 2.135: Error to Connect the Output to Same Input in Same Function**

In this case, we use the **Feedback Node** to make the value that was output to one of the outputs of a function or SubVI the value entered into one of the inputs to that function or SubVI the next time that this function is executed or SubVI, as shown in figure 2.136.



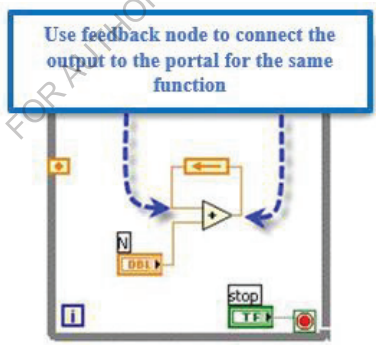



**Figure 2.136: Feedback Node**

It is like a **Shift Register** that holds its value after the end of the current implementation and moves it to the next implementation. It can transmit any type of data. Terminal is found with the Feedback Node; this connects to the initial value of the Feedback Node. This initial value will be the value of the portal connected to the Feedback Node the first

time that you implement it. This Terminal is set automatically by just listing the Feedback Node, as shown in figure 2.137.

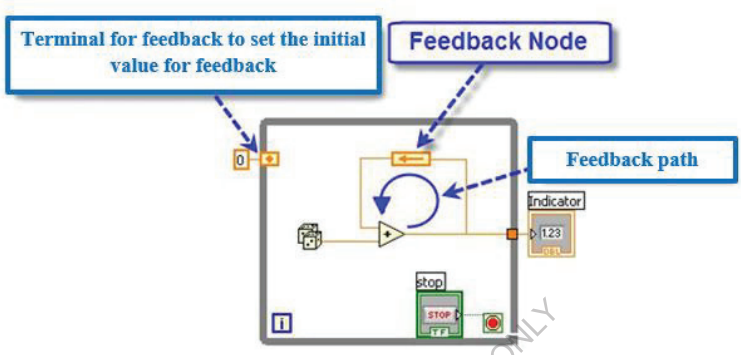



**Figure 2.137: Initial Value of Feedback**



As shown, the direction of the arrow on the Feedback Node determines the flow of the data.

## Include Feedback Node

Feedback Node is listed from **Function Palette >> Programming >> Structures**, as shown in figure 2.138.

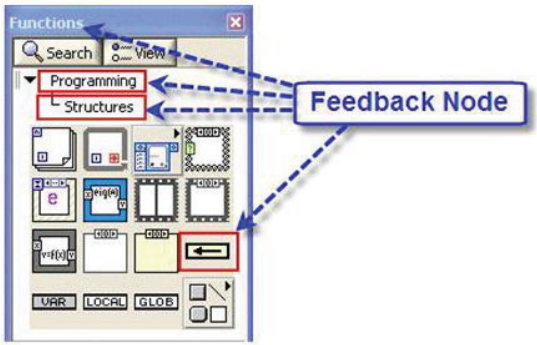



**Figure 2.138: Include Feedback Node**

**What happens if the initial values for the Feedback Node are not specified?**

If you set the initial value for the Feedback Node, this value will be the value. **Feedback Node** Before entering the first implementation in Loop, this is whether the program is executed for the first time, the second time, or any time. However, if the initial value of the Feedback Node is not specified, Feedback Node will take the initial value of the type of data that it transmits before the first time it is executed. Where Loop and so on. This is similar to what happens in **Shift Registers**.

## Case Structure



It is one way of doing conditional sentences and is equivalent to If… then… else in other programming languages.

Case Structure is listed from **Function Palette >> Programming >> Structures** or from **Function Palette >> Express >> Execution Control**, as shown in figure 2.139.

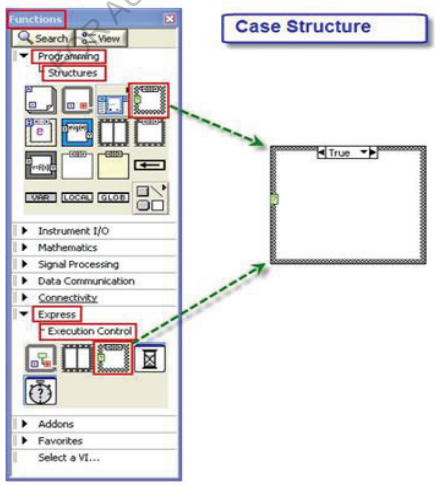



**Figure 2.139: Insert Case Structure**

The **Case Structure** consists of several states and an entry called **Selector Terminal**. Each case in which this code is written is called **Subdiagram**. When Case Structure is executed one case is executed and the value that the input takes is determined by the state to be executed, as shown in figure 2.140.

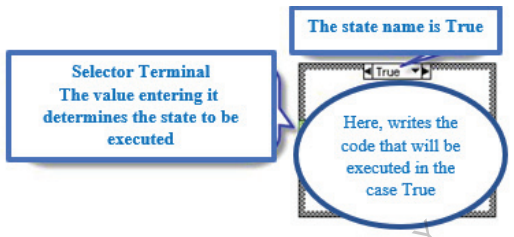



**Figure 2.140: Case Structure**



Each case has a name, which is the value at which the case will be executed. The case name is written in **Selector Label** and can be modified using the **Labeling Tool**, as shown in figure 2.141.

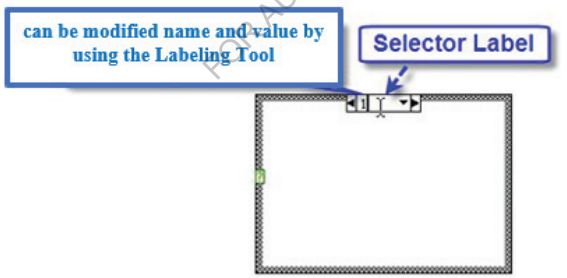



**Figure 2.141: Labeling Tool to modified the Selector Label**

**Case Structure** usually contains several states but only displays one case content

To view the contents of the following case, we use the **right arrow** ▸. To view the contents of the previous case we use the **left arrow** ◂. It is possible to navigate to any situation by pressing the arrow and selecting the case to be displayed. Or, **right-click** on

the Case window, choose **Show Case**, and choose the case you want, as shown in figure 2.142.

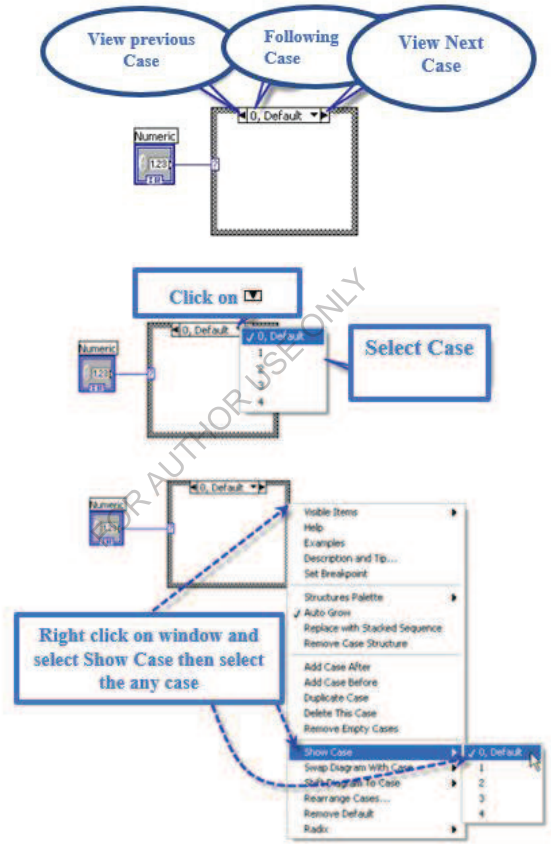



**Figure 2.142: Select Case in Case Structure**

**How to identify cases with different types of values connected to the Selector Terminal:**

• If the input type is **Boolean** then there are two states, **True** or **False**, as shown in figure 2.143.

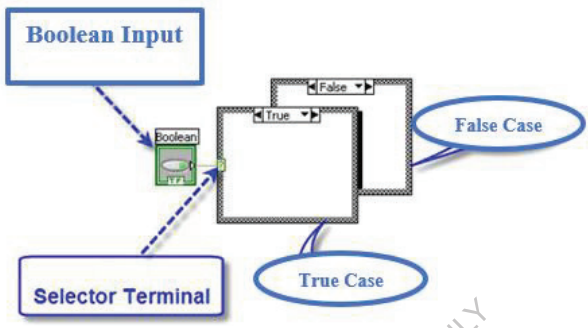



**Figure 2.143: Boolean Input Case Structure**

• If the input is **Numeric type**, there can be a final number of values. In this case, **Case Structure** contains only **two cases**, and you can add the cases you want, as shown in figure 2.144.



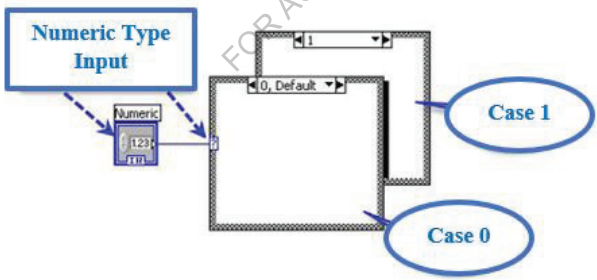



**Figure 2.144: Numeric Input Case Structure**

The case can have more than one value, and to do so, write the values between them are commas, as shown in figure 2.145.

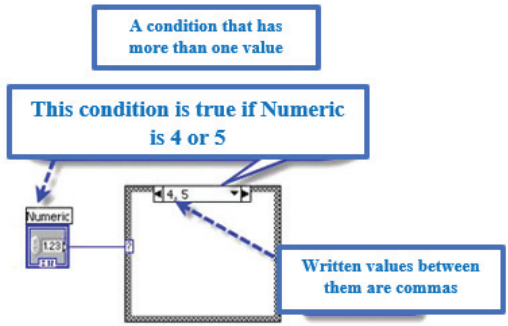



**Figure 2.145: The Case Can Have More Than One Value**

It can also be the case for a **range of values**. To do this, we use the **two points ..** for expressing the extent, for example **2..20** means that this case for all the correct numbers from 2 to 20 and **3..** means that the case for all the correct numbers equal to or greater than 3, as shown in figure 2.146.



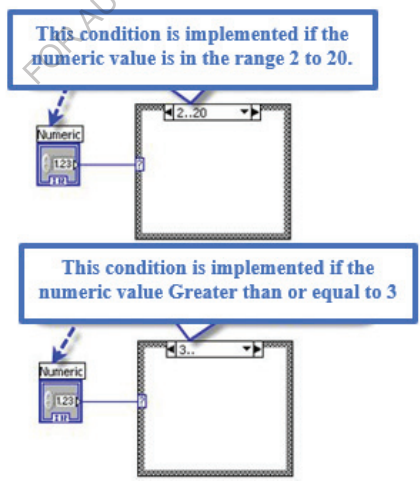



**Figure 2.146: Case for a Range of Values**

### Default Case

The condition to be executed if the value connected to the input is **not assigned a case**. For example, if the existing conditions are for values 1, 2 and 3 only, what would be the case if the income value is 100 for example. **The default case will be executed**. The default state appears next to its assigned value as Default, as shown in figure 2.147.

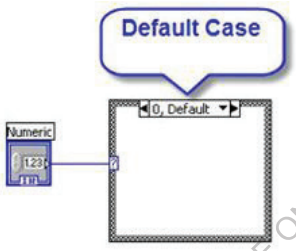





**Figure 2.147: Default Case**

Any case can be made the default case, as shown in figure 2.148.

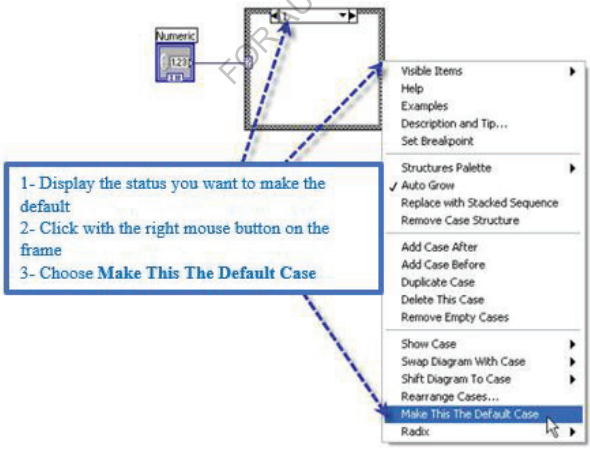



**Figure 2.148: Make the Default Case**

• If the input is **String type**, write the case values between **two quotes** and if you do not put the labels, LabVIEW will set them automatically, as shown in figure 2.149.

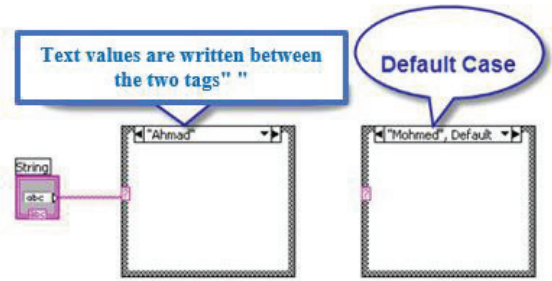



**Figure 2.149: String Type Case**

Note that the word **"Default"** does not exist between two quotes.

The case can have more than one value by writing the values between them with a comma.

**We can make Case Sensitive or Case Insensitive**

- **Case Sensitive** means taking the capital letters and lower-case letters, for example, ALI does not match.
- **Case Insensitive** means that there is a difference between uppercase and lowercase letters for example, ALI matches Ali matches.

In figure 2.150 that shown the select of case insensitive.

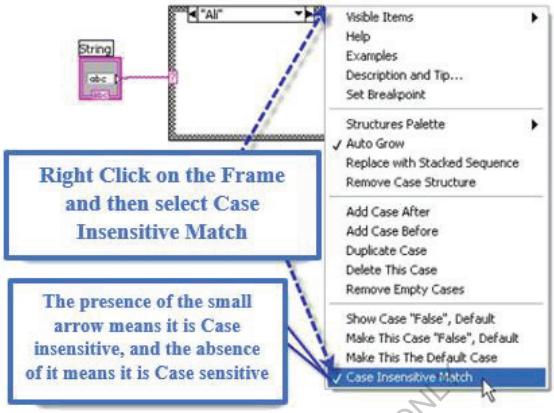



**Figure 2.150: Case Sensitive or Insensitive Select**

**Note that:**

- If a **Floating-Point** number is connected, LabVIEW converts the value to a valid I32 number and then selects the case based on the converted number, as shown in figure 2.151.



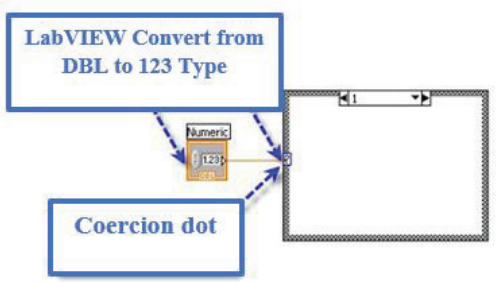



**Figure 2.151: LabVIEW Conversion Type**

- The **Case Selector** can be moved anywhere on the Case Structure frame.

## Insert and Remove Cases

You can add a new case, delete a case, or make a copy of an existing case and other options by pressing the right click on the Case Structure frame, as shown in figure 2.152.

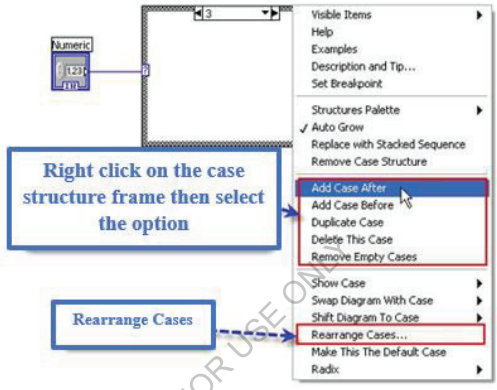



**Figure 2.152: Rearrange Cases**

When choosing to rearrange the cases, a window appears in which we specify the order of the cases, as shown in figure 2.153.


**Figure 2.153: Rearrange Cases Window**

## Inserting and removing from and to case structure

We mentioned that there is a Selector Terminal and its value determines the status to be executed. We can also enter any number of inputs into cases or output any number of outputs through tunnels, as shown in figure 2.154.

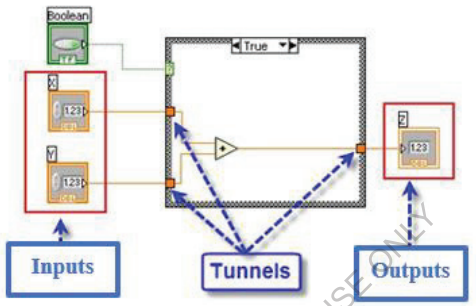



**Figure 2.154: Inputs and Outputs to Case Structures**



It is not a condition that cases have inputs and outputs.

It is not a condition that the condition use all of the inputs.

But if an output is connected in any case, this output must be connected in all cases. First, the program will not be executed and it will give you an error message stating that there is a tunnel to an unconnected outlet, as shown in figure 2.155.

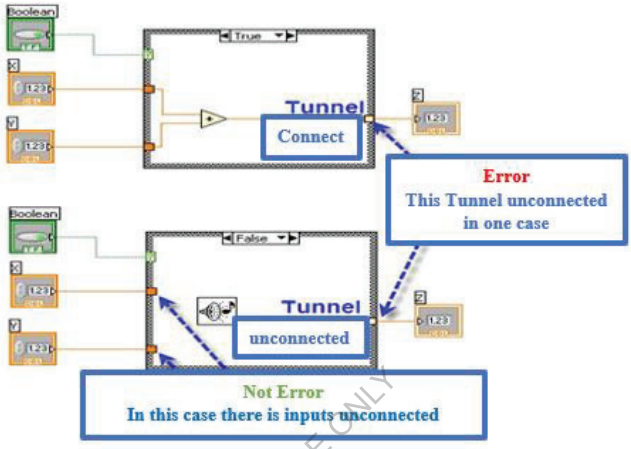



**Figure 2.155: Error Connection in case structure**

To fix this error, we do the following that shown in figure 2.156:

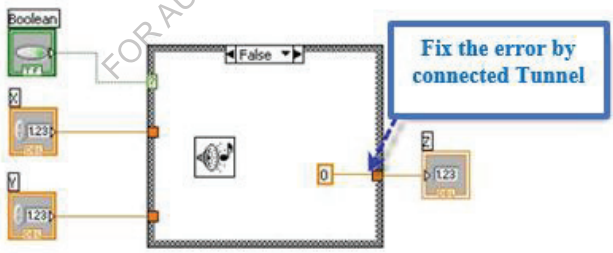



**Figure 2.156: Fix Error Connection**

# The Sequence Structure

The **Sequence Structure** contains a set of codes, each code called **Subdiagram**, which executes them all in the order you find them in **Sequence Structure**. Each code is placed within a frame. The order of this framework is the order in which the code is executed. There are two types of Sequence Structure: **Flat Sequence Structure** and **Stacked Sequence Structure**. They differ only in the way Code view (**Subdiagrams**).

## Flat Sequence Structure

**Frames** or **codes** are displayed next to each other, and the codes inside the frames are executed in order from **left to right**, as shown in figure 2.157.

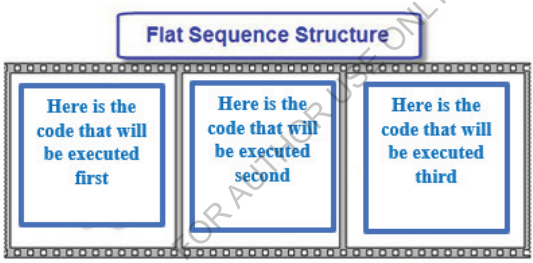





**Figure 2.157: Flat Sequence Structure**

## Stacked Sequence Structure

It is the same as Case Structure in the way the codes are displayed (on each frame), as shown in figure 2.158.

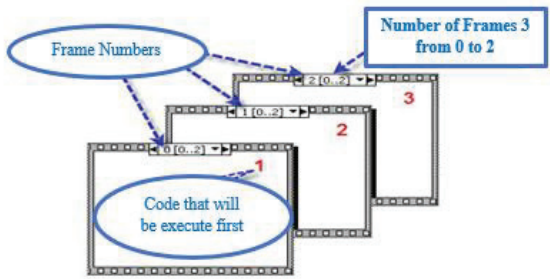



**Figure 2.158: Stacked Sequence Structure**

## Insert Sequence Structure

Sequence Structure is inserted from **Function Palette >> Programming >> Structures,** as shown in figure 2.159.



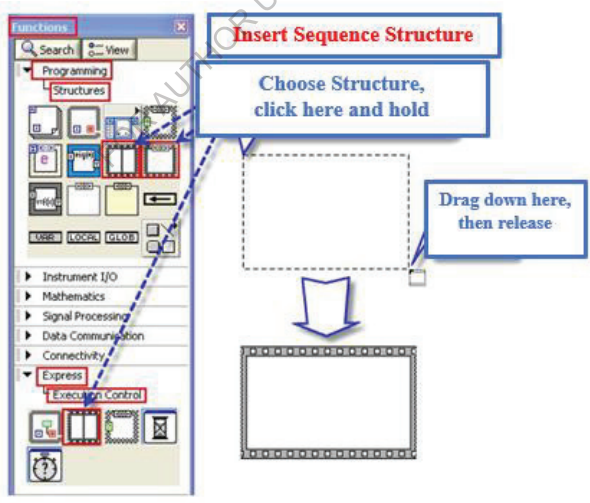



**Figure 2.159: Insert Sequence Structure**

To add a new frame, **right-click** the Structure frame and press **Add Frame** After or Add Frame Before, as shown in figure 2.160 and 2.161.

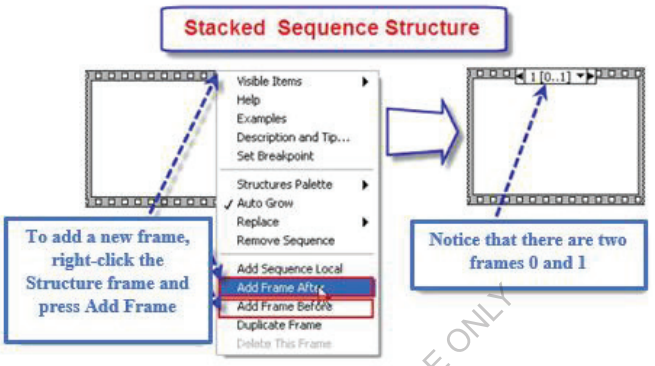



**Figure 2.160: Add New Frame in Stacked Sequence Structure**

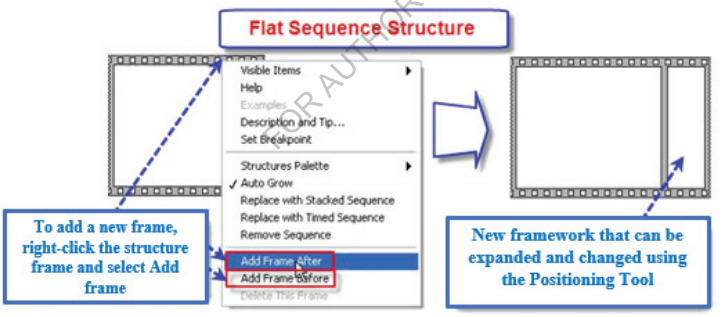



**Figure 2.161: Add New Frame in Flat Sequence Structure**

Data is transferred between frames in the Flat Sequence Structure by linking the codes via **Tunnels**, as shown in figure 2.162.

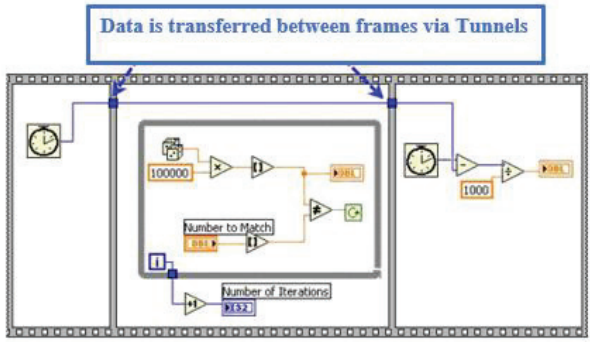



**Figure 2.162: Transferred Data via Tunnel**

In the case of **Stacked Sequence Structure**, **Sequence Local** is used to transfer data between different frames.



## Insert Sequence Local

Right click on the frame and select "**Add Sequence Local**", as shown in figure 2.163.

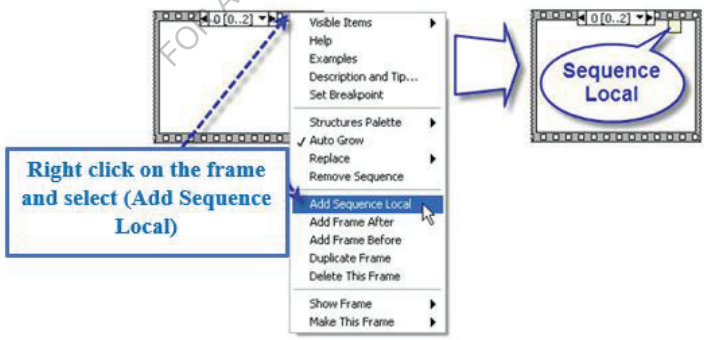



**Figure 2.163: Insert Sequence Local**

We reach a value to **Sequence Local** in one frame and this value is read in the other frames, as shown in figure 2.164.

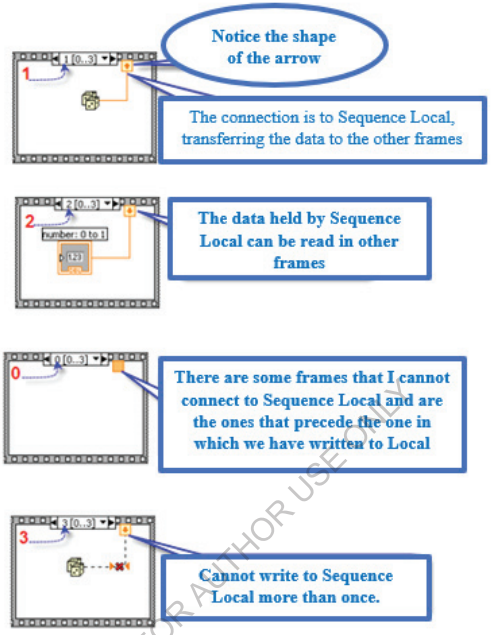



**Figure 2.164: Reach A Value to Sequence Local**

**Note:**

At any time during the program design, it is possible to switch between Flat and Stacked Sequence Structures, as shown in figure 2.165.

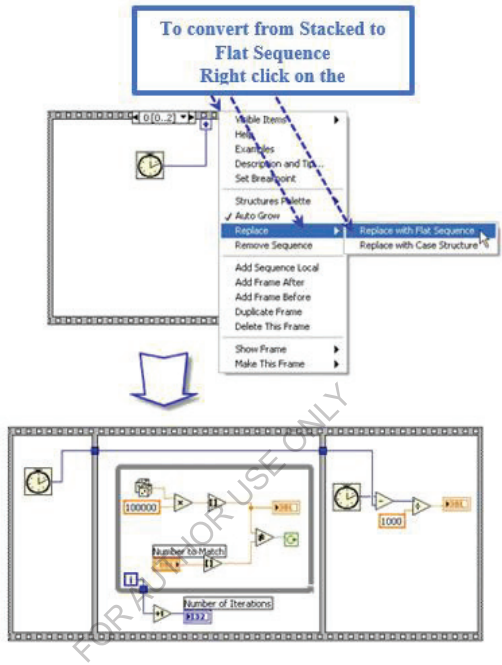





**Figure 2.165: Convert from Stacked to Flat Sequence Structure**

## Timing

**Timers** are of great importance in LabVIEW. They are used to measure time, synchronize between different tasks, and delay work in Loops so that Loops operate at an appropriate rate so that they do not capture the overall processor speed.

There are two types of timers: **basic functions** and **Express Timing VIs**. All timers can be listed from **Function Palette >> Programming >> Timing**, as shown in figure 2.166.


**Figure 2.166: Insert Timing Function**



## Basic functions

### 1- Wait Function (ms)

It makes this VI function wait for a certain number of **milliseconds**, and then the VI implementation continues. That is, it delays in **milliseconds**. If you want the delay to be in **seconds**, multiply the number of seconds by **1000**, as shown in figure 2.167.

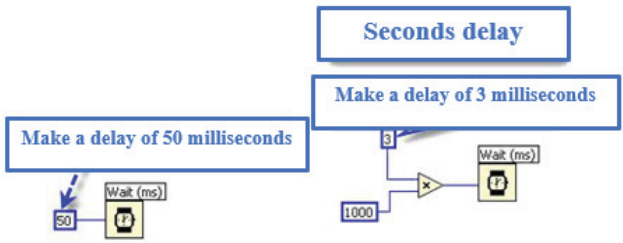



**Figure 2.167: Wait Function (ms)**

## 2- Wait Until Next Function

This makes the VI function wait until the **internal clock** of the **operating system** (for example Windows) is equal to the multiple of the function's internal value (**millisecond multiple**), after which the VI continues. The inner value of the function is in **milliseconds**. In figure 2.168 that shown the wait until next function.

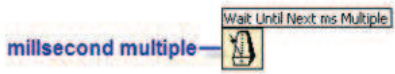



**Figure 2.168: Wait Until Next Function**

The Wait (ms) and Wait Until Next ms Multiple functions are similar, but they differ. The difference is clear from the following example:




If the internal value of Wait until Next ms Multiple  is **10**. The value of an hour or timer for the system is, for example, 112. The function will wait **8 milliseconds** until the system's internal clock is **120**, which means multiples of **10** (the inner value of the function). The next time, you will wait until the time 130 is also a multiple of 10. So, note that **synchronization occurred with the system clock**.

As for the **function Wait (ms)**, if the value entered is **10**. The value of an **hour** or **timer** for the system is, for example, **112**.


The function will **wait 10 milliseconds** for the system's internal clock to be **122**. That is, it has waited **10 milliseconds**, regardless of the system clock value. Thus, you will wait every time you implement **10 milliseconds**, regardless of the **system clock value**.

Therefore, the function is used to make Loops inside activities implemented in specific times. It is also used to make synchronization between events.

Note that when this function is used in Loop, it is the first time that it is executed that you may wait for less than the value entered in it, as in the previous example.

The above two functions are frequently used in Loops so that Loop does not gain processor speed. These functions make the delays necessary to do other tasks, such as the interaction of Front Panel elements with the user.

### 3- Tick Count (ms) Function

This function is used to obtain the **internal timer (hour)** of the operating system, in **milliseconds**, as shown in figure 2.168.

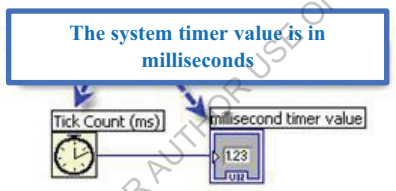





**Figure 2.168: Tick Count (ms)**

This function is used to calculate the **elapsed time** in implementing a specific part of the program.

## Express Timing Function

In addition to the basic functions in the **LabVIEW** environment, there are **Express Timing VIs**, which are **Time Delay** and **Elapsed Time**.

### 1- Time Delay Express VI

It is exactly the same as the **Wait (ms) function** and the difference between them, in this Express VI, we have a **delay time in seconds**. When Express VI is listed, a window appears, specifying the delay time that we want, as shown in figure 2.169.

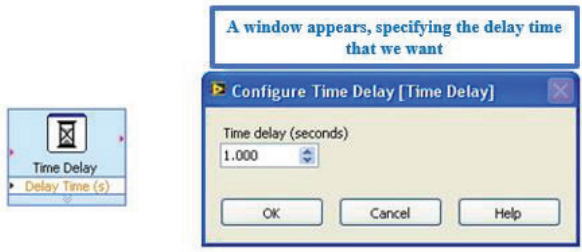



**Figure 2.169: A window to Specify the Time Delay**

The delay value can be connected to Express VI and this value will replace the value that we specified by the previous window, as shown in figure 2.170.




**Figure 2.170: Delay Value Connected to Express VI**

## 2- Elapsed Time Express VI

This **Express VI** lets you know if time has passed (its value has already been determined) or not. When that Express VI is listed, a window appears that specifies the time we want to know whether or not that time has passed, as shown in figure 2.171.

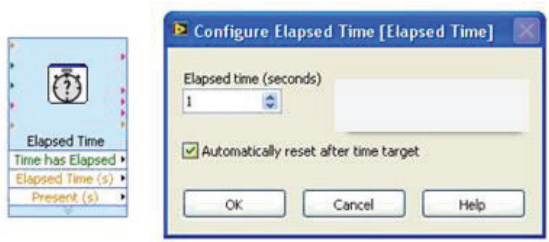



**Figure 2.171: Configuration Elapsed Time**

The Time Has Elapsed value determines whether the time has elapsed or not. If **True** then the specified time has passed. If **False**, the time has not passed yet, as shown in figure 2.172.

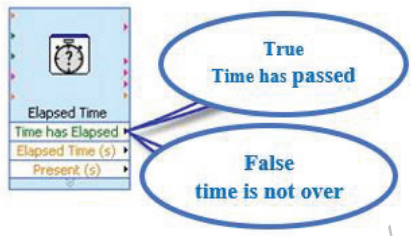



**Figure 2.172: Elapsed Time Output**



## Formula Node

**Formula Node is used to write mathematical equations**.

For example, to apply this equation Y = X2 + X + 1 with the basic functions in LabVIEW, it will look like this shown in figure 2.173.

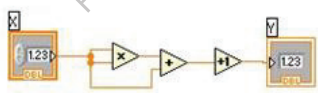



**Figure 2.173: Without Formula Node**

But **with Formula Node** you will be like this shown in figure 2.174.

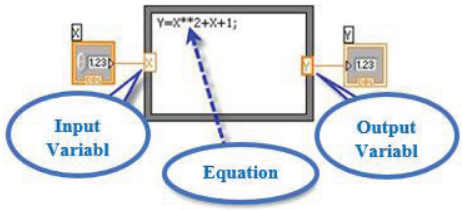



**Figure 2.174: with Formula Node**

## How to insert Formula Node

**Formula Node** is listed from **Factions Palette >> Programming >> Structures**, as shown in figure 2.175.

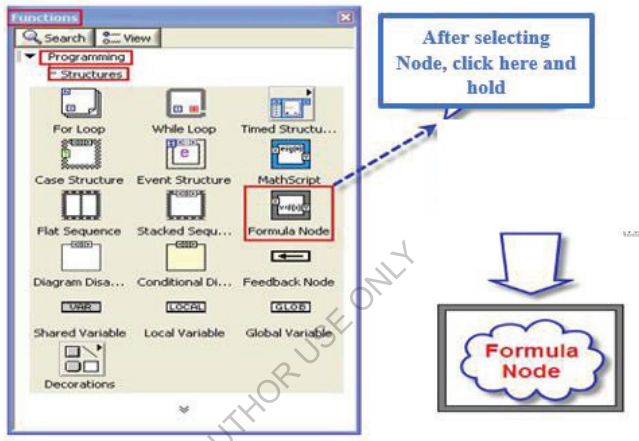





**Figure 2.175: Insert Formula Node**

## Adding Internal and External Variables

You can add any number of variables inside or outside by pressing the right mouse button and choosing:

**Add Input** to add an Input variable.

**Add Output** to add an outside variable.

In figure 2.176 that shown the adding internal and external variables.

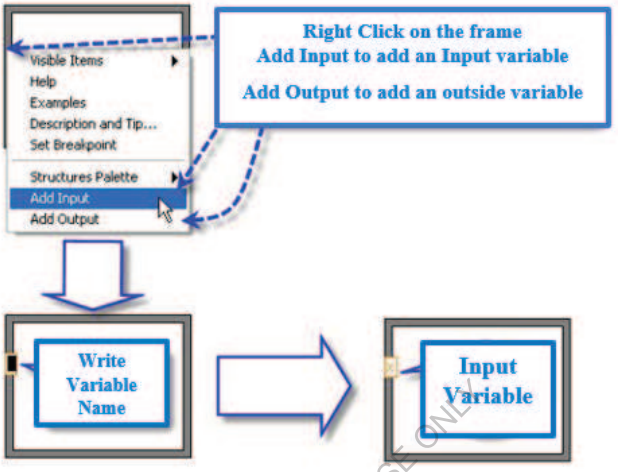





**Figure 2.176: Adding Internal and External Variables**

**Notes**

• There cannot be internal or external variables with the **same name**.

• It can be an outside variable and another variable with the same name.

• Outbound variables are **thicker** than inbound variables, as shown in figure 2.177.

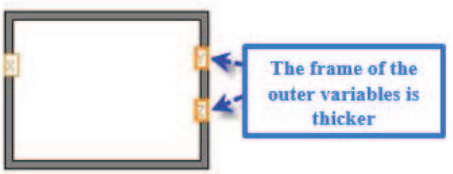



**Figure 2.177: Outbound Variables are Thicker Than Inbound Variables**

• The variable can be changed from inside to outside or vice versa, as shown in figure 2.178.

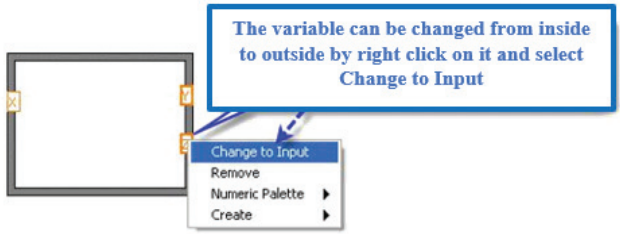



**Figure 2.178: Changed Variable from Inside to Outside Or Vice Versa**

## Write Equations Inside Formula Node

Equations are written inside **Formula Node** in a similar way to writing sentences in **C language**. The equation is written using **functions** and **variable names**. Is placed after each **semicolon equation**. There can be more than one equation in **Formula Node**. Any variable used in equations should be set as an **external variable** or an input to the sides of the Formula Node.



**Note:**

The variables outside the Node do not have to be linked. For example, there are intermediate variables used between equations, as shown in figure 2.179.

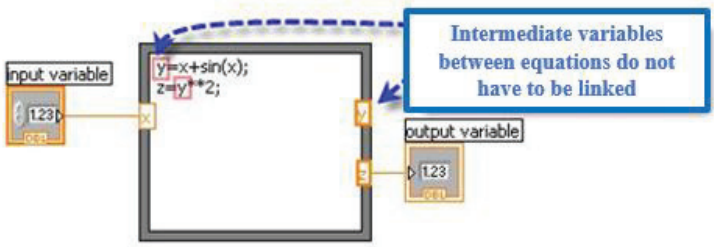



**Figure 2.179: Intermediate Variables Between Equations**

**Note:**

There is no specific number of variables or the number of equations used in **Formula Node**. Formula Node can be enlarged to accommodate any number of equations by using the **Positioning Tool**, as shown in figure 2.180.

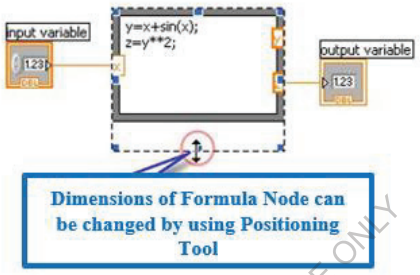



**Figure 2.180: Change Dimension of Formula Node**



A **vertical slider bar** may also appear inside the Node to write as many formulas as possible, as shown in figure 2.181.

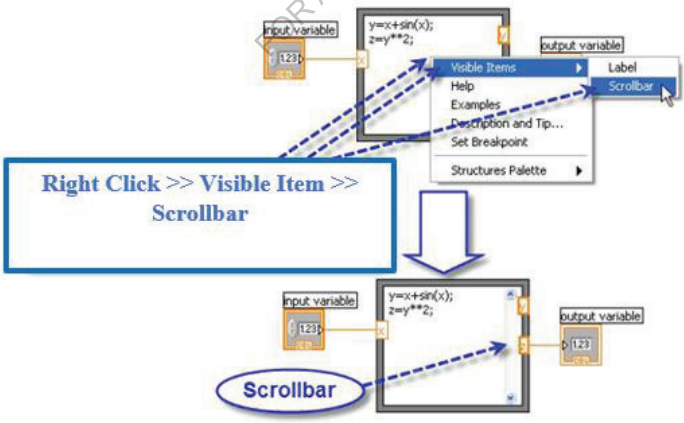



**Figure 2.181: Scrollbar appears**

## Expression Node

It is a simplified form of **Formula Node**. It contains one mathematical phrase, one entry, and one output, as shown in figure 2.182.

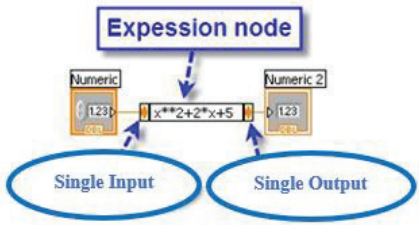



**Figure 2.182: Expression Node**



- As the entrance and the exit are lonely, they **do not have names**.
- The mathematical phrase is written without the mark **;**.
- A **single variable** is used in this mathematical phrase, and this variable can be called by any name.
- This variable expresses the input.
- The output is the product of the mathematical phrase.

**Expression Node** is listed from **Function Palette >> Programming >> Numeric**, as shown in figure 2.183.

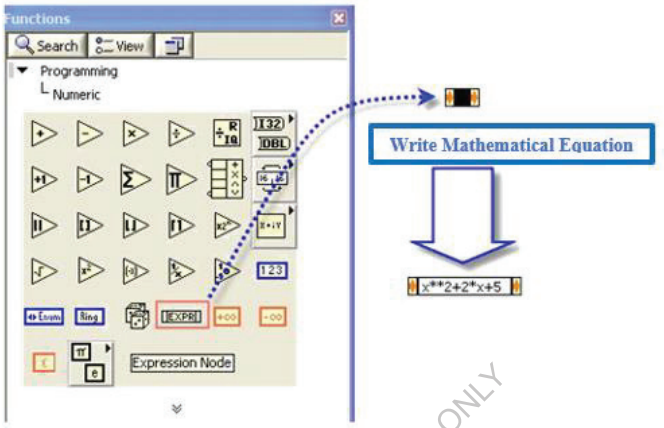



**Figure 2.183: Insert Expression Node**

# Lesson Six: Arrays

## Aim of the lesson

**1) Learn about Arrays and their processes.**

## Introduction

In this lesson, we will learn about a **complex type** of Data Type, which is **Arrays**. The presence of this type facilitates the **storage** and **handling** of data.

We will learn about the many uses of it and it functions in the **LabVIEW** environment.

## Arrays

### Create an array of Controls or Indicators



An array of Controls or Indicators is created in the Front Panel in two steps:

1- The matrix framework is managed and it will contain the elements of the matrix.

It is inserted from **Controls Palette >> Modern >> Array, Matrix & Cluster**, as shown in figure 2.184.

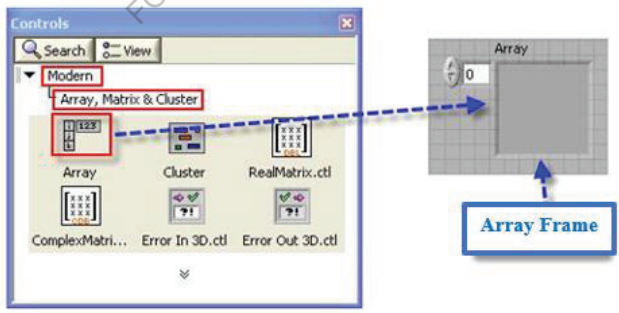



**Figure 2.184: Create Array**

2- The Control or Indicator that we want is inserted into the array window to create an array from it, as shown in figure 2.185.

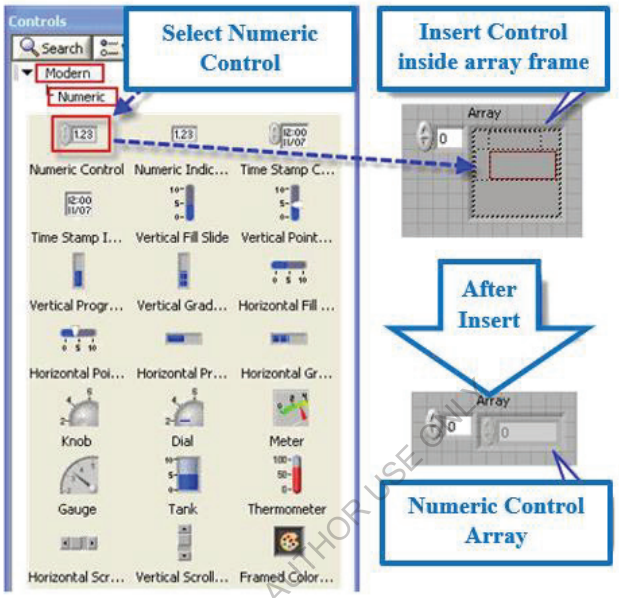



**Figure 2.185: Make Control Array**

Note that matrix elements can be Controls or Indicators, not a mixture of them.

When inserting the array frame and before placing any Control or Indicator inside it the Terminal appears in the **Block Diagram** and the color of this Terminal is black and has two empty brackets until its data type is specified and this is done by setting the Control or Indicator or fixed inside the array frame.

When Control, Indicator, or static are placed within the array frame, the Terminal color changes to the Data Type of the object in which it is placed. Terminal Data is also enclosed in parentheses in Terminal, as shown in figure 2.186.

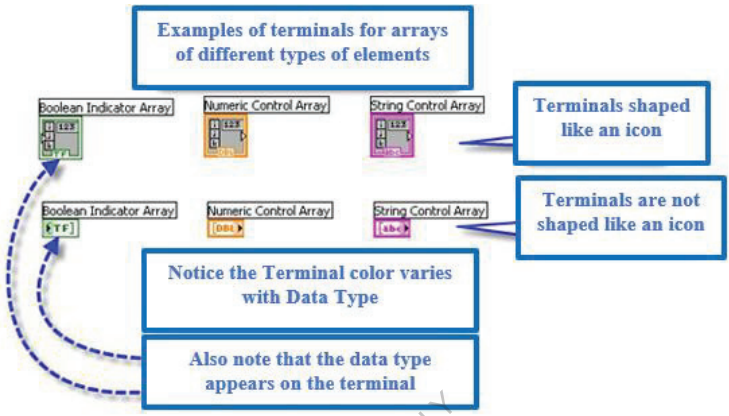



**Figure 2.186: Examples of Array Types**

Remember that Terminal can be in the form of an icon or in the old form and to change between them we press the right click and choose or cancel the selection as **View icon** and this is a change in the shape only and has no other effect, as shown in figure 2.187.

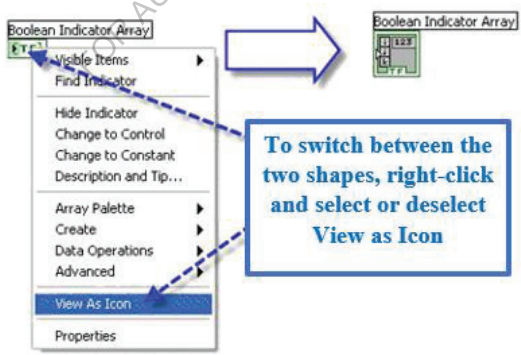



**Figure 2.187: View as Icon**

Dimensioning the element inside the array can be done by moving the Positioning tool on the edge of the element inside the array until the indicator appears as two indexes or ↔, or we press and drag to the dimensions that we want, as shown in figure 2.188.

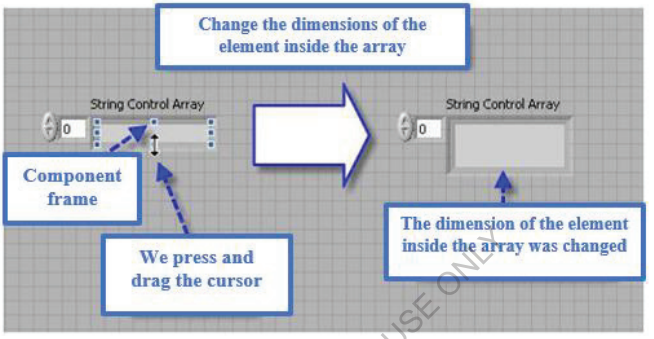



**Figure 2.188: Change Dimension of Array Element**



The number of elements that appear inside the array window can be changed by changing the dimensions of the array frame by moving the Positioning tool on the array frame until the index is in the form of an index with two arrows or ↔ we squeeze and withdraw the arrow, as shown in figure 2.189.

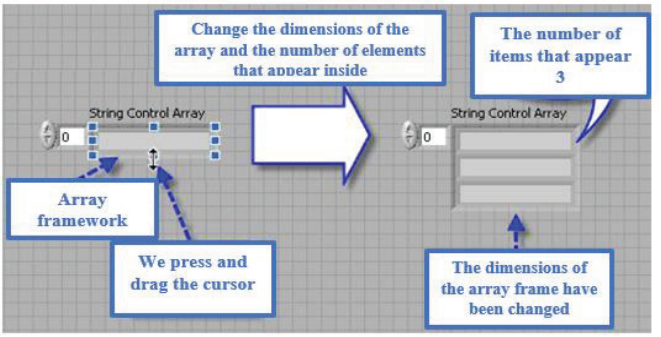



**Figure 2.189: Change the Dimension of Array**

The element inside the array appears **dimmed** if no value is specified for it and the value of the elements is determined by writing the value directly in the element or using the arrows in the event that the element is **Numeric Control** or by clicking on the element if the **Boolean is exiting** the parameter of the parameter, as shown in figure 2.190.

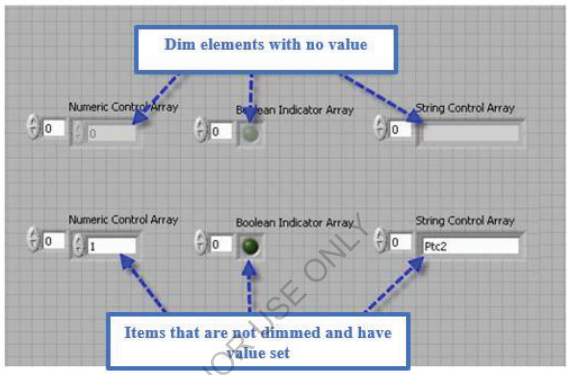



**Figure 2.190: Array with and without Elements**

The value of the array indicates the first element in the element. If you change the value of the directory, you will see an item with a number equal to the value of Index, as shown in figure 2.191.

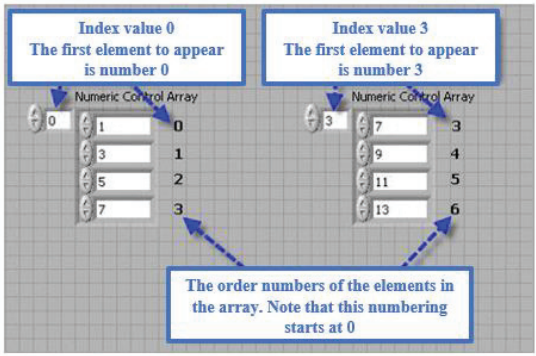



**Figure 2.191: Change Index of Array**

## Create an array of Constants in Block Diagram

In the same way an array of Constants is created in Block Diagram in two steps:

1- We include the Array Constant frame from **Function Palette >> Programming >> Array**, as shown in figure 2.192.

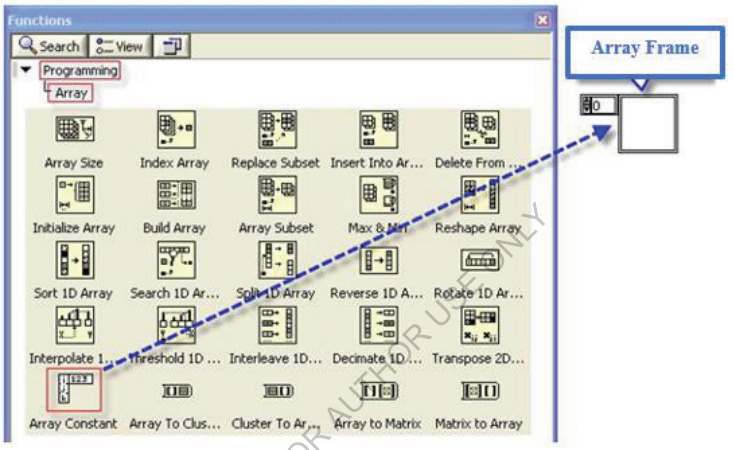



**Figure 2.192: Create an array of Constants in Block Diagram**

2- Insert "Constant" inside the array frame. This parameter can be of type Numeric, String, or Boolean, as shown in figure 2.193.

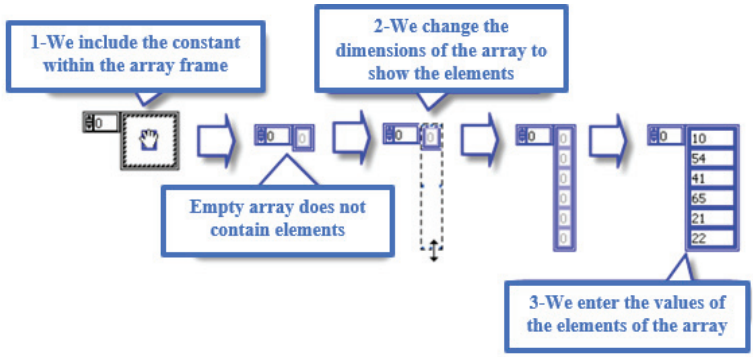



**Figure 2.193: Create Constant Array**

## Delete all elements of the array

3- All values in the array can be deleted and made blank by pressing the **right click on the Index** - not on the Item - and choosing **Data Operation >> Empty Array**, as shown in figure 2.194.



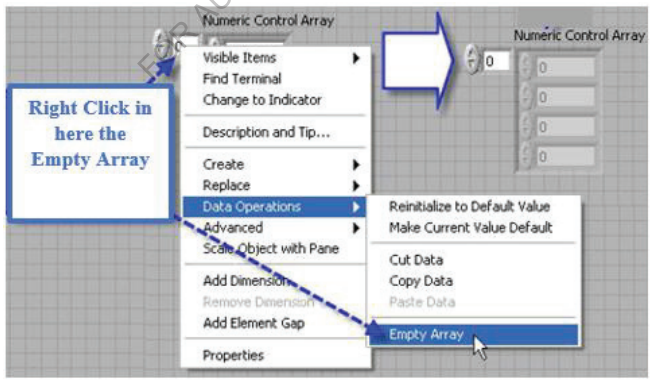



**Figure 2.194: Delete all elements of the array**

## Add or delete an element in an array

You can add an element or delete an element from an array by right-clicking on the element inside the array and choosing

**Data Operation >> Insert Element Before** ➔To add an item

or

**Data Operation >> Delete Element**➔To delete an item

## How a two-dimensional Array (matrix)

We do a one-dimensional matrix and then press Index on the right click and choose **Add Dimension**, as shown in figure 2.195.

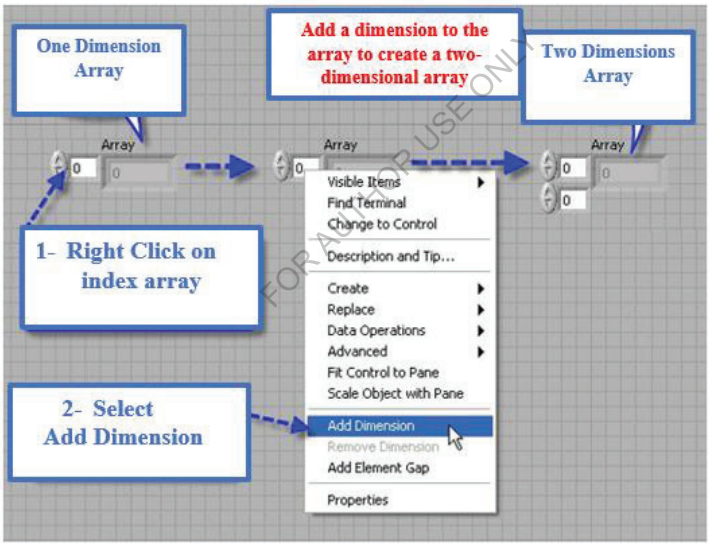



**Figure 2.195: Add Dimension Array**

A greater number of elements can be displayed in the array by moving the Positioning tool on the array window. To turn into a two-way arrow ↕ or ↔, press and move the

frame in one of the vertical or horizontal directions. And re-do this in the other direction, as shown in figure 2.196.

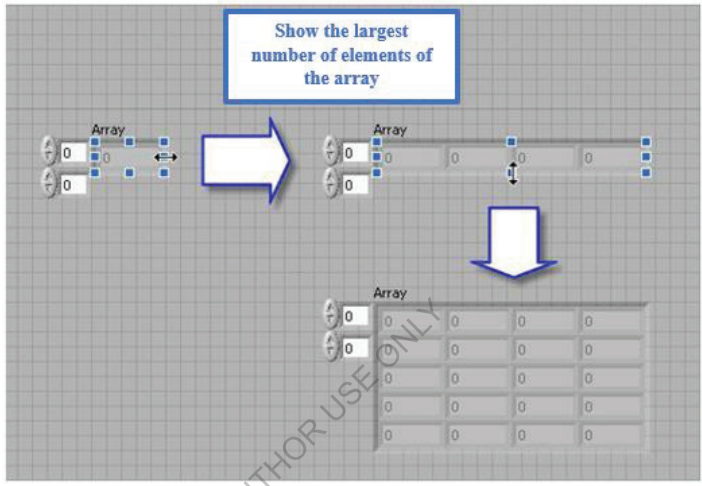



**Figure 2.196: Display Larger Number of Element**



We note that when the matrix is created it is blank and we can write the values that we want directly to it.

In the same way, we can delete one dimension of an array that has more than one dimension by pressing the right button on the index matrix and choosing Remove Dimension.

CHAPTER TWO
LABVIEW LEARNING LESSONS

# Lesson Seven: Graphs

## Aim of the lesson

**1) Learn about Waveform Graph and how to draw on it.**

## Introduction

**Graphics** are of great importance in representing data and obtaining information from it. And LabVIEW is rich in many tools that allow you to draw and represent different types of data in the best picture. In this lesson, we will look at one of the types found in LabVIEW, which is **Graph**.

## Graphs

**Graphs** draw the ready data into an array and draw the data once, and no other data can be added to the drawing except by repainting.

There are many types of graphs such as **Waveform Graphs**, **XY Graphs**, **Intensity Graphs**, **3D Graphs**, **Digital Waveform Graphs**, and other special types. In this lesson we will explain about the **Waveform Graph** type.



## Waveform Graph

**Waveform Graphs** draw data on the **Y** axis and **X** values for this data start from a specific number and increase with a **constant value**, meaning that the difference between the values of X is constant (for example 4,3,2,1,0 or) 20,15,10,5,0, i.e. the distance between the points drawn on the X-axis will be **equal**. In Graph, each value of X has only one value Y.

### Waveform Graph Insert

Waveform Graph is listed on **Front Panel Controls Palette >> Modern >> Graph**, as shown in figure 2.197.

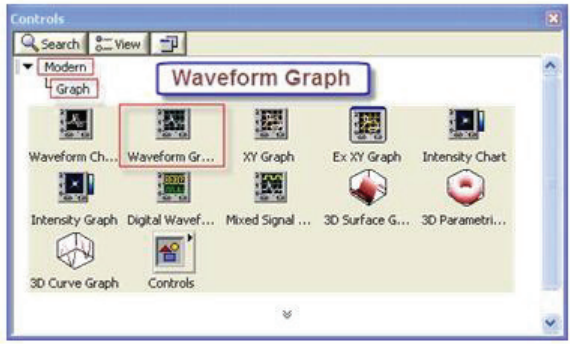



**Figure 2.197: Waveform Graph Insert**

Like any **Indicator**, once it is inserted into the **Front Panel**, terminal appears in the Block Diagram, as shown in figure 2.198.




**Figure 2.198: Waveform Graph Terminal**

## How to draw on a graph

### Waveform Graph contains one graphic

1- The simplest way to draw on a Waveform Graph is to connect the array to the Waveform Graph Terminal directly. The array values will be the Y values. The X values will start from zero and increase to one for each Y value in the array.

That is, X0 (the first value in X values) = 0 and dX (the increase in X values) = 1, as shown in figure 2.199.

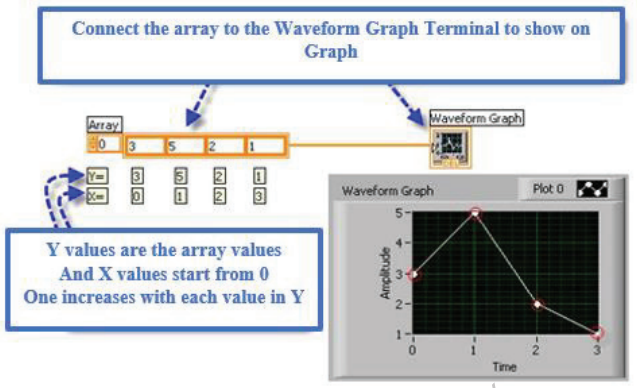



**Figure 2.199: Draw One Graph**



**For example:**

Create a **one-dimensional array** using **For Loop** using the **Auto Indexing** feature and draw the array on the Waveform Graph, as shown in figure 2.100.

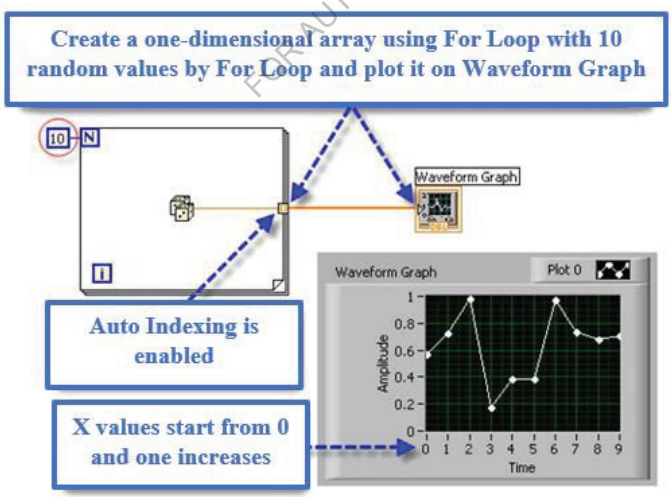



**Figure 2.100: Draw One Graph by For-Loop**

This can be done using the **Bundle function**. In this case, we make a bundle of three entries

**First:** the value of X0 (the value at which the values of X begin).

**Second:** the value of dX (the amount of the increase in the value of X for each value of Y).

**Third:** The one-dimensional matrix we want to draw (Y values).

The Bundle function is listed from **Function Palette >> Programming >> Cluster & Variant**.

This is the previous example, but using the **Bundle function**, as shown in figure 2.101.

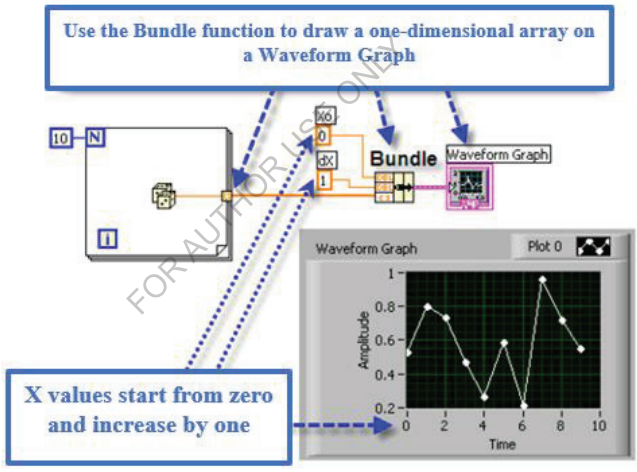



**Figure 2.101: Draw One Graph by using Bundle**

### Graph contains more than one graphic

To draw more than one separate graphic on Waveform Graph we create a two-dimensional array and each row in this array is drawn in a separate drawing. In this case it is **X0 = 0** and **dx = 1**, as shown in figure 2.102.

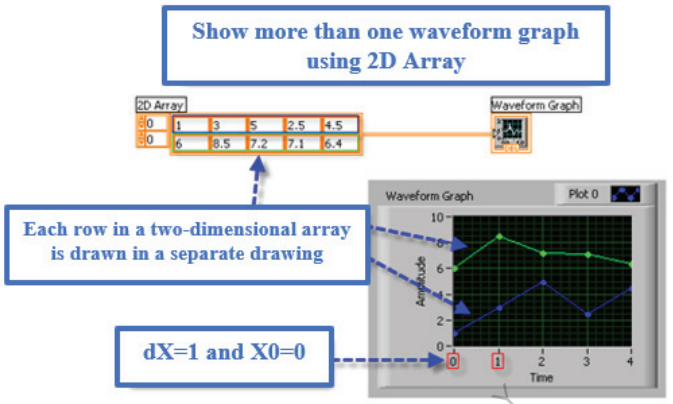



**Figure 2.102: Draw Two Graphs**



More than **one-dimension** array can be grouped into **two-dimensional** array using the **Build Array** function, as shown in figure 2.103.

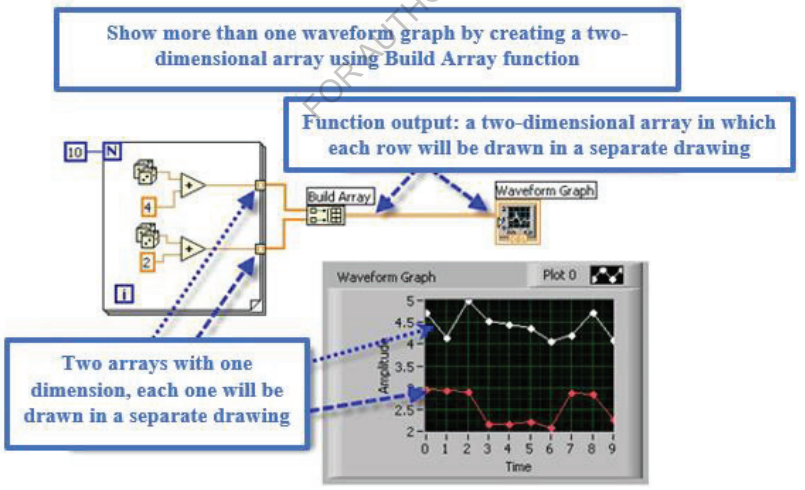



**Figure 2.103: Draw Two Graphs by using Build Array**

To change the value of **X0** (the value by which the values of X) and **dX** (the value of the increase in the values of X) begin each graphic.

We use the **Bundle function** with every **one-dimensional** array and then compile the **Build Array** function, as shown in figure 2.104.

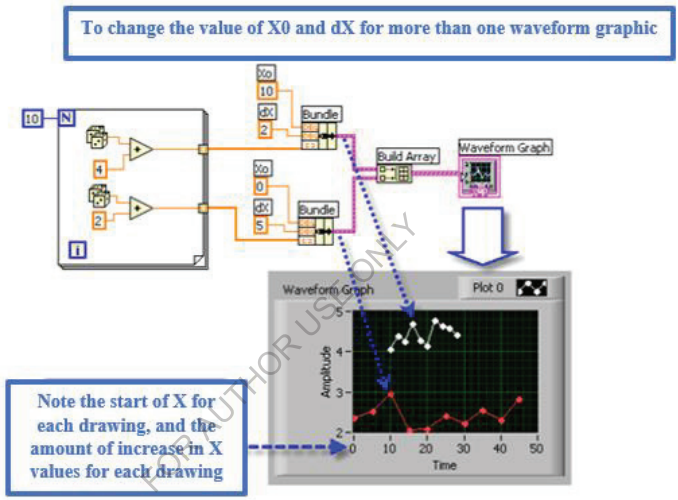



**Figure 2.104: Draw Two Graphs by using Two Build Arrays**

CHAPTER TWO
LABVIEW LEARNING LESSONS

# Lesson Eight: Strings

## Aim of the lesson

**1) Learn about Strings and their features.**

## Introduction

In this lesson we will learn about a very important component of programming language is String and we will learn how to deal with Strings.

## Strings

A **text** or **String** is a set of characters called **ASCII Characters**. Each letter contains a **code** (a number for each letter) called **ASCII Code**, as we know that all data inside the computer is in the form of numbers.

These letters include **upper-** and **lower-case** letters, numbers 0 through 9, and symbols, as well as a non-printable or non-printable letter such as **Space** (New Line) or **Carriage Return,** and **non-printable** characters that look like all of them.

String Control, String Indicator and String Constant are in LabVIEW to show, entering and dealing with Strings. In figure 2.105 that shown the string control.

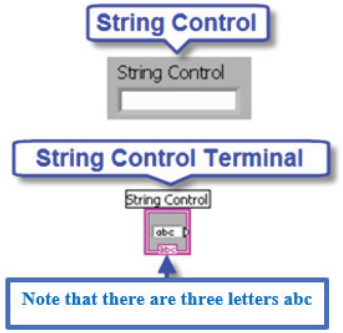



**Figure 2.105: String Control**

**String Control** or **String Indicator** are insert on the **Front Panel** of **Controls Palette >> Modern >> Strings & Path**, as shown in figure 2.106.

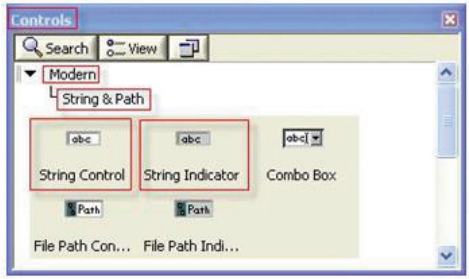



**Figure 2.106: Insert String**

**String Constant** is insert in **Block Diagram** of **Functions Palette >> Programming >> String**, as shown in figure 2.107.



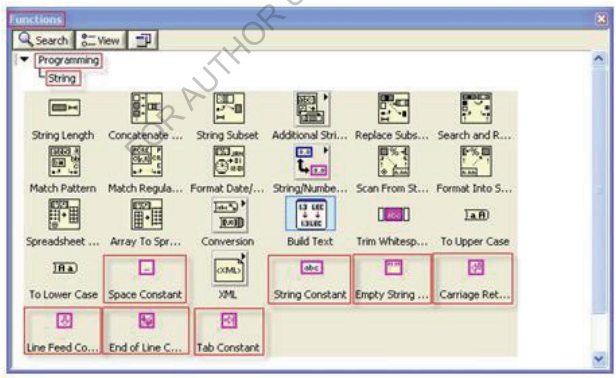



**Figure 2.107: Insert Constant String**

## String Controls properties

Strings have several properties, including:

# CHAPTER TWO
# LABVIEW LEARNING LESSONS

## How to display texts

Texts can be displayed in **String Control** in more than one way. To change the way text appears in String Control, we press it with the **right click** and **choose the method we want**, as shown in figure 2.108.

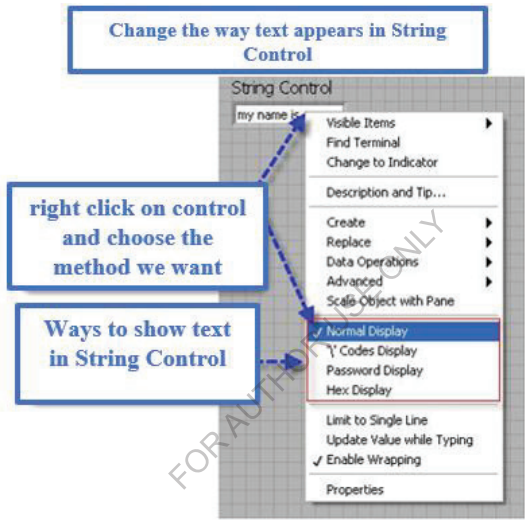



**Figure 2.108: Ways to Show Text in String Control**

These methods are:

### Normal Display

In it the text appears in the normal way we are used to, and this method is the basic method, as shown in figure 2.109.

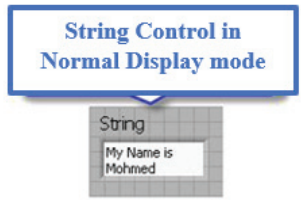



**Figure 2.109: Normal Display Mode**

In it, non-printable characters appear in the form of the **tag /** followed by the **hexadecimal ASCII Code** for each letter.

This table 2.1 shows how nonprintable characters appear in this mode.

**Table 2.1: Nonprintable Characters Appear in This Mode**

| **\00 to \FF** | **The tag \ and then the ASCII code for the character in hexadecimal The ASCII Code component characters include (\ 0C) or (\ FF).** |
|---|---|
| **\b** | (BS) Backspace is equivalent (\ 08) ASCII Code. |
| **\f** | (FF) Form feed is equivalent (\ 0C) ASCII Code. |
| **\n** | (LF) New Line equivalent (\ 0A) ASCII Code. |
| **\r** | (Cr) Carriage Return يكافىء (\0D) ASCII Code. |
| **\t** | (HT) Tab Equivalent (\ 09) ASCII Code. |
| **\s** | Space is equivalent (/ 20) ASCII Code. |
| \\ | (\) Backslash is equivalent (\ 5C) ASCII Code. |



**Notes:**

1- Special characters such as the letter s in **\ s** and the letter t in **\ t** is lowercase. If capital letters are capitalized, they will not be interpreted as special characters.

**Example:** \ b means Backspace, \ B does not mean Backspace, and \ B LabVIEW considers it \ 0 B ASCII Code.

2- The **hexadecimal system** characters are uppercase letters such as \ FF or \ 0A.

**Notes:**

1- Changing the display pattern does not affect the content of String.

2- This pattern is used to test the texts that are sent or received from measuring devices via Serial Port, for example, where non-printable characters such as **\ n New line** or **\ r Carriage Return** are used to end the orders and the data sent when addressing the measuring devices.

### Password Display

In this pattern, letters do not appear, but instead the symbol ***.** When you enter letters in String Control in this mode, the letters that you type do not appear, but instead the symbol * appears, but the value that is read through the String **Control Terminal** in the **Block Diagram** is the set of characters that you entered. This pattern is used when String Control is used for entering **passwords**, as shown in figure 2.110.



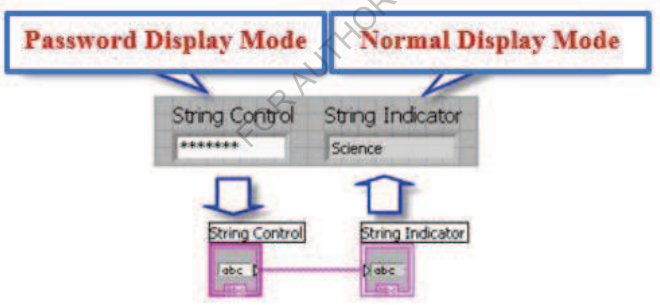



**Figure 2.110: Password Display Mode**

### Hex Display

In this mode the **ASCII Code** is displayed for all letters in the **hexadecimal text** instead of the letters, as shown in figure 2.111.

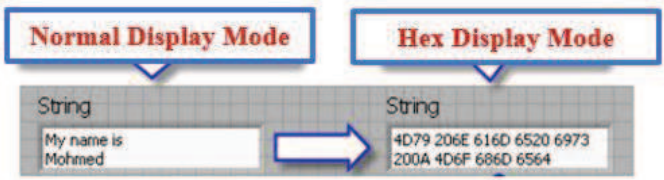



**Figure 2.111: Hex Display Mode**

## Limit Single Line Feature

This property determines the number of possible lines in String Control with one line and when entering letters and pressing Enter, this means that the line has been entered.

If this feature is not selected - this is the normal state - and press Enter while characters are entered into String Control, it will move to a new line in Control, as shown in figure 2.112.



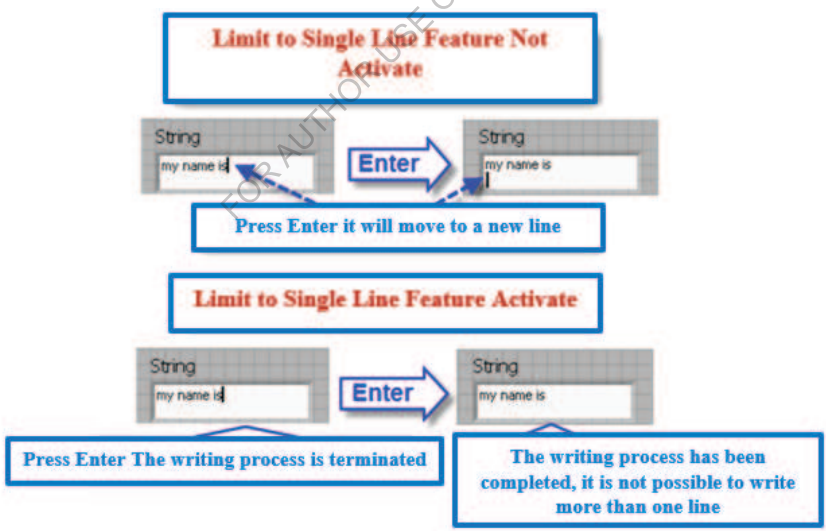



**Figure 2.112: Limit Single Line Feature**

To activate this feature, right-click on String Control and select **Limit to Single Line**, as shown in figure 2.113.

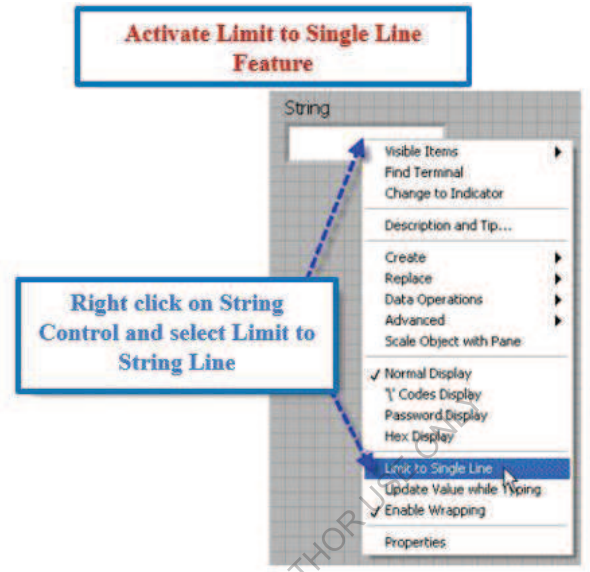



**Figure 2.113: Activate Limit to Line Feature**

## Update Value while Type Feature

When writing to String Control, the value of their Terminals in Block Diagram is not changed until after the writing process is complete, either by pressing Enter or by pressing the mouse anywhere outside of Control or by pressing the button ☑ on the toolbar. This is so that the text is not read completely incomplete before it is written. But if you want the Terminal value to be updated while writing, we enable the Update Value while Type feature.

The Update Value while typing feature is activated by right-clicking on String Control and choose **Update Value while Typing**.

# Creates Array LabVIEW

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW for Create Array.

## Apparatus

LabVIEW software.

## The Experiment Blocks


### For Loop

Executes its subdiagram n times, where *n* is the value wired to the count (**N**) terminal. The iteration (**i**) terminal provides the current loop iteration count, which ranges from 0 to *n*-1.

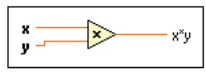


**Block Diagram➔R.C➔Structure➔For loop**


### Random Number

Produces a double-precision, floating-point number between 0 and 1. The number generated is greater than or equal to 0, but less than 1. The distribution is uniform.

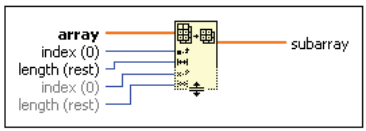
**Block Diagram➔R.C➔Numeric➔Random Number (0-1)**




### Multiply

Returns the product of the inputs.

**Block Diagram➔R.C➔Numeric➔Multiply**

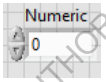



## Array Subset

Returns a portion of **array** starting at **index** and containing **length** elements.

**Block Diagram➔R.C➔Array➔Array Subset**


**Front Panel➔Numeric➔Numeric Indicator**



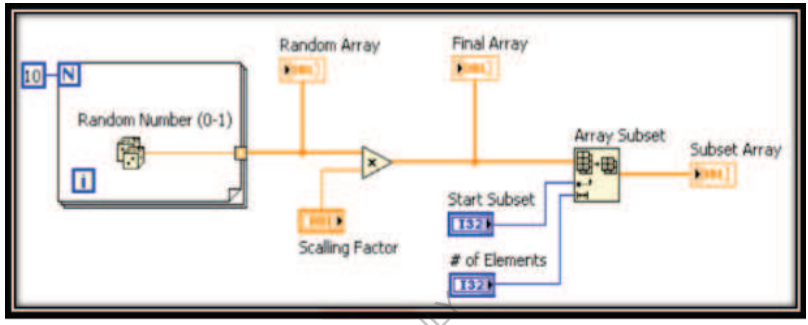

**Front Panel➔Numeric➔Numeric Control**



## Procedure

### 1- Front Panel

## 2- Block Diagram




## Discussion



1- Design 2D array by using LABVIEW.
2- Redesign Icon for Lab VIEW to create Two Dimension array? using random No. in implementation.
3- Design a diagram by using LABVIW to multiply two arrays.
4- Find max and min. value and index in the final array?
5- Find size of final array?

CHAPTER THREE
LABVIEW PROJECTS

# Convert a Temperature in Celsius to a Temperature in Fahrenheit

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW for solve equation converting temperature from Celsius to Fahrenheit.

## Apparatus

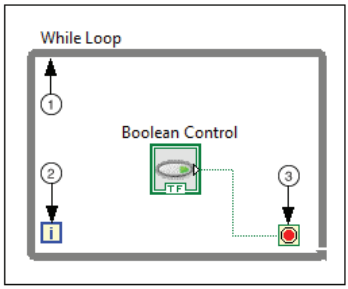
LabVIEW software.

## The Experiment Blocks

### While Loop

Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**

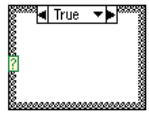



**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1,

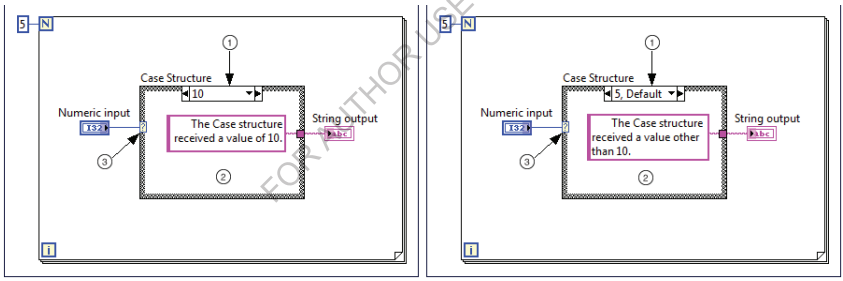
the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.



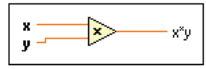

Contains one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the case selector determines which case to execute.

**Components of a Case Structure**



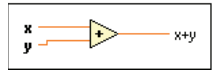



**Block Diagram➔R.C➔Structure➔Case Structure**




Returns the product of the inputs.

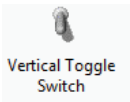
**Block Diagram➔R.C➔Numeric➔Multiply**




Computes the sum of the inputs.

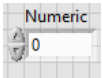
**Block Diagram➔R.C➔Numeric➔Add**

# CHAPTER THREE
# LABVIEW PROJECTS

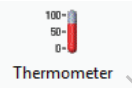



**Front Panel ➔R.C➔Boolean➔Vertical Toggle Switch**


**Front Panel➔Numeric➔Numeric Indicator**




**Front Panel➔Numeric➔Numeric Control**

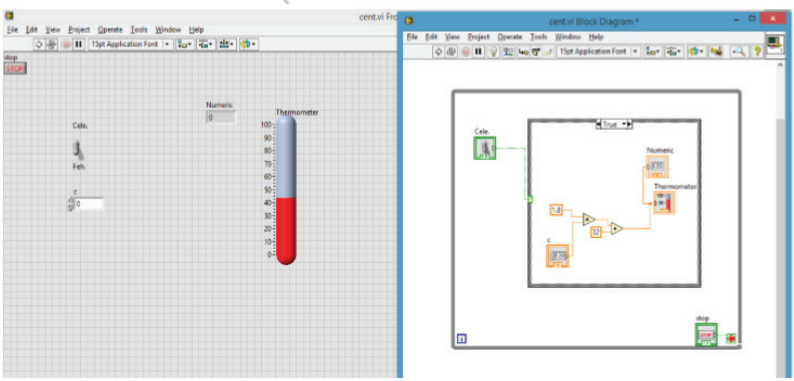



**Front Panel➔Numeric➔Thermometer**



## Procedure

### Front Panel& Block Diagram




***The complete design***

Complete the value in the table below:

| NO. | Temp.◦ C | Temp. ◦ F |
|---|---|---|
| 1 | 25 | |
| 2 | 30 | |
| 3 | 35 | |
| 4 | 40 | |
| 5 | 45 | |
| 6 | 50 | |
| 7 | 55 | |
| 8 | 60 | |

## Discussion

1- Redesign Icon for LabVIEW to converting temperature from Fahrenheit to Celsius?

2. Redesign program to converting temperature from Fahrenheit to Celsius using the equation (°F - 32) x 5/9 = °C?

3. Complete the value in the table below:



| NO. | Temp.◦ F | Temp. ◦ C |
|---|---|---|
| 1 | 10 | |
| 2 | 15 | |
| 3 | 20 | |
| 4 | 25 | |
| 5 | 30 | |
| 6 | 35 | |
| 7 | 40 | |
| 8 | 45 | |
| 9 | 50 | |

CHAPTER THREE
LABVIEW PROJECTS

# Build a VI to Generates a Signal Display

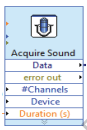
## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW for display and generate signal.

## Apparatus

LabVIEW software.

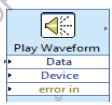
## The Experiment Blocks





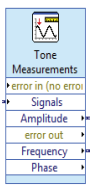

**(Acquire Sound Express VI):** Acquires data from a sound device.

**Block Diagram➔R.c ➔ Graphic&Sound ➔Sound ➔ Input**




**(Play Waveform Express VI):** Plays data from the sound output device using finite sampling.

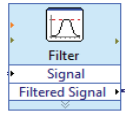
**Block Diagram➔R.c ➔ Graphic&Sound ➔ Sound ➔ Output**




**(Tone Measurements Express VI):** Finds the single tone with the highest amplitude or

searches a specified frequency range to find the single tone with the highest amplitude. You also can find the frequency and phase for a single tone.

**Block Diagram➔R.c ➔ Express ➔Signal Analysis ➔ Tone**

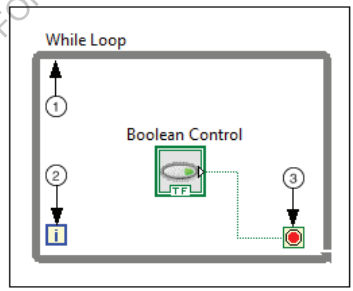



**(Filter):** Processes signals through filters and windows.

**Block Diagram➔R.c ➔Express ➔Signal Analysis ➔Filter**

**While Loop:** Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**



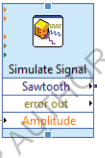

***Subdiagram***—Contains code that the While Loop executes once per iteration.

***Iteration Terminal (i)***—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or $2^{31}$-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to

keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range. ***Conditional Terminal***—Evaluates a Boolean input value to determine whether to continue executing the While Loop. To specify whether the loop stops for a TRUE or FALSE Boolean value, configure the continuation behavior of the loop. You also can determine when the loop stops by wiring an error cluster to the conditional terminal.

**Block Diagram ➔ R.C ➔ Strucher ➔ While Loop**

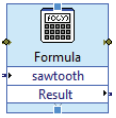

**(Knob):** **Front panel ➔ R.C ➔ Numeric➔Knob**





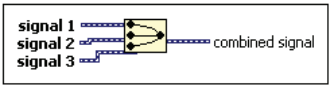

**(Simulate Signal):** Simulates a sine wave, square wave, triangle wave, sawtooth wave, or noise signal.

**Block Diagram ➔ R.C ➔ Express ➔ Signal Analysis ➔ Simulate Signal**



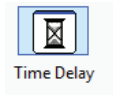

**(Formula):** Uses a calculator interface to create mathematical formulas.

**Block Diagram ➔ R.C ➔Express ➔Arithmetic&Comperision ➔Formula**

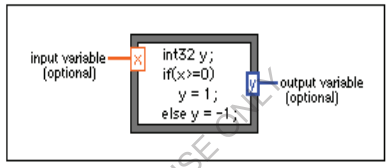


**(Merge Signals Function):** Merges two or more supported signals, such as scalar numerics, 1D or 2D arrays of numerics, scalar Booleans, 1D or 2D arrays of Booleans, waveforms, or 1D arrays of waveforms, into a single output.

**Block Diagram ➔ R.C ➔Express ➔Sig Manip➔Merge Signals**




Specifies how many seconds to delay running the calling VI. The default is 1.000.






## Formula Node

Evaluates mathematical formulas and expressions similar to C on the block diagram. The following built-in functions are allowed in formulas: abs, acos, acosh, asin, asinh, atan, atan2, atanh, ceil, cos, cosh, cot, csc, exp, expm1, floor, getexp, getman, int, intrz, ln, lnp1, log, log2, max, min, mod, pow, rand, rem, sec, sign, sin, sinc, sinh, sizeOfDim, sqrt, tan, tanh.

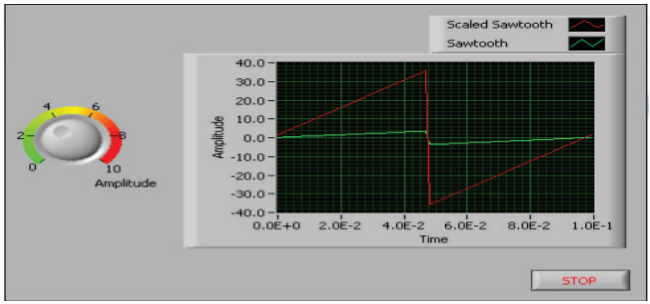
**Block Diagram >> R.C >> Structure >> Formula Node**

## Procedure

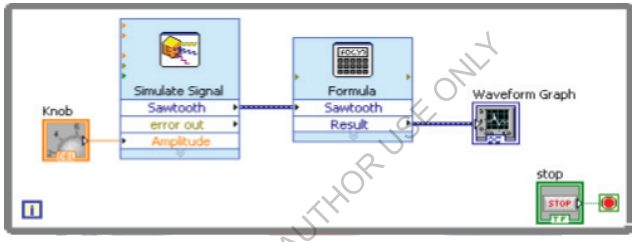
### Part1

### Front Panel

# CHAPTER THREE
# LABVIEW PROJECTS

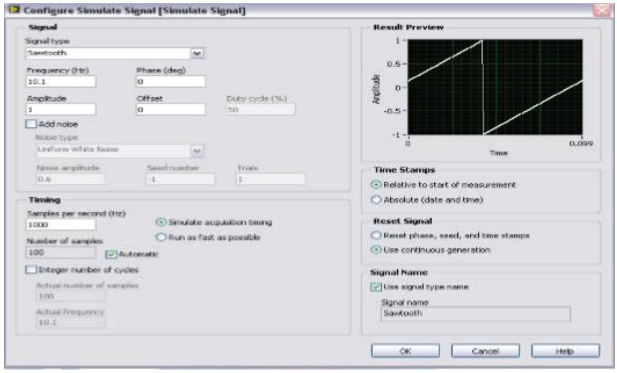



## Block Diagram






On the Block Diagram ➔Right click➔Express➔Input➔ Simulate signal➔select type of signal as “Saw tooth signal”➔Press “OK”. As shown below.



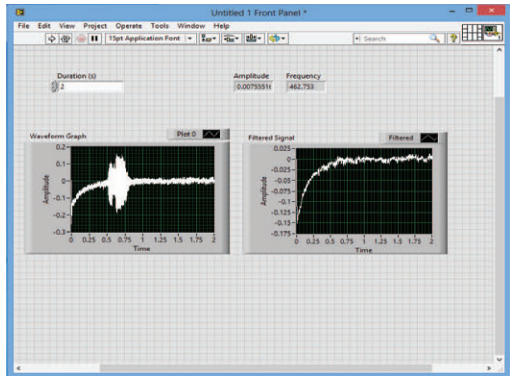

**signal simulate input**

Define the value of the scaling factor by entering *10 after Sawtooth in the Formula text box. You can use the Input buttons in the configuration dialog box or you can use the *, 1, and 0 keyboard buttons to enter the scaling factor. If you use the Input buttons in the configuration dialog box, LabVIEW places the formula input after the Sawtooth input in the Formula text box. If you use the keyboard, click in the Formula text box after Sawtooth and enter the formula you want to appear in the text box. The Configure Formula shown below.

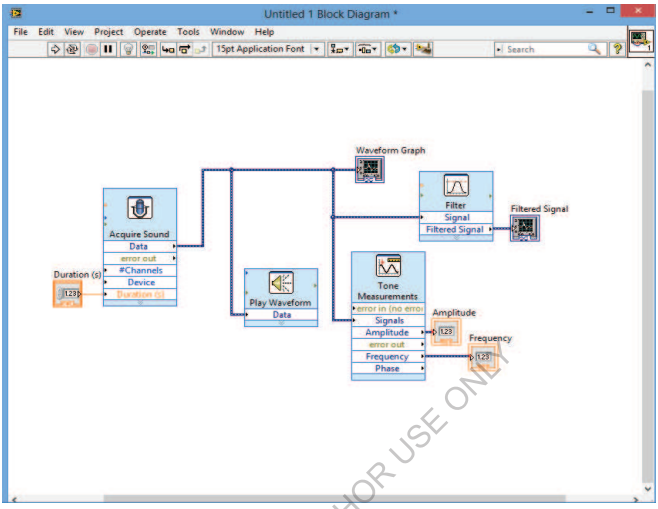



**formula configures**

## Part2 Front Panel



## Block Diagram

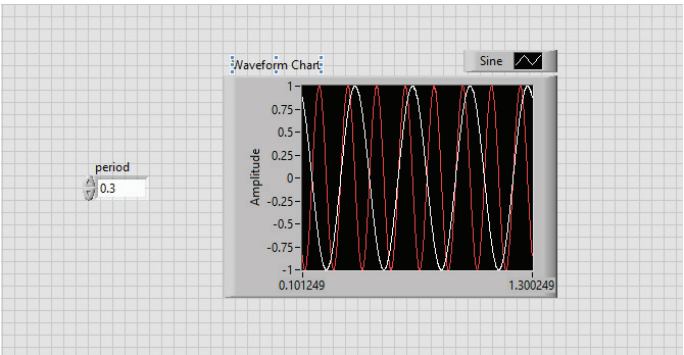




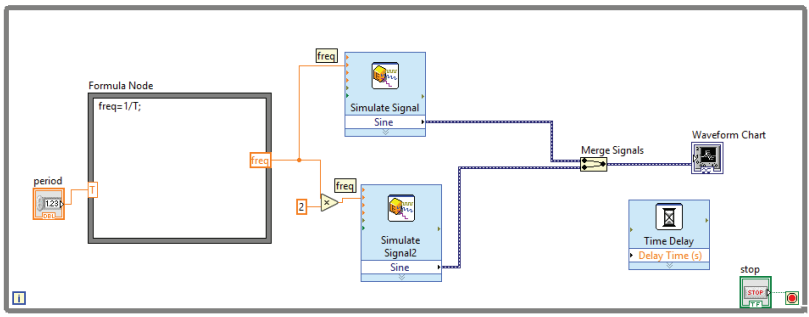

### Part3

### Generate two signals in waveform graph

## Front Panel




## Block Diagram




## Discussion

1- Redesign signal generated in steps over to display Sawtooth?

2- Redesign program using singe wave and also enlarge signal one hundred times and also write steps and graph front panel and block diagram?

3- what main difference when using waveform chats and waveform graph?

CHAPTER THREE
LABVIEW PROJECTS

# Using Formula Node

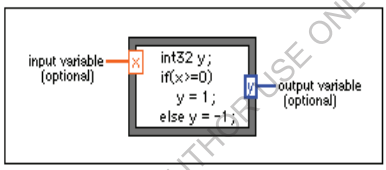
## Aim of Experiment

In this exercise, the objective is to use the Formula Node in a VI. Complete the following steps to build a VI that uses the Formula Node to perform a complex mathematical operation and simple program then show the graphs results.

## Apparatus

LabVIEW software.

## The Experiment Blocks






**Formula Node**

Evaluates mathematical formulas and expressions similar to C on the block diagram. The following built-in functions are allowed in formulas: abs, acos, acosh, asin, asinh, atan, atan2, atanh, ceil, cos, cosh, cot, csc, exp, expm1, floor, getexp, getman, int, intrz, ln, lnp1, log, log2, max, min, mod, pow, rand, rem, sec, sign, sin, sinc, sinh, sizeOfDim, sqrt, tan, tanh.

**Block Diagram >> R.C >> Structure >> Formula Node**


**For Loop**

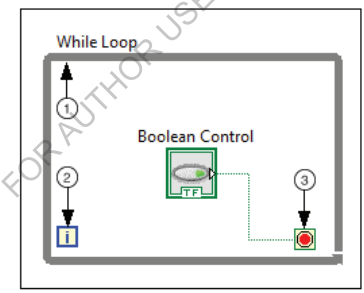
Executes its subdiagram n times, where *n* is the value wired to the count (**N**) terminal. The iteration (**i**) terminal provides the current loop iteration count, which ranges from 0 to *n*-1.

**Block Diagram➔R.C➔Structure➔For loop**

## While Loop

Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**



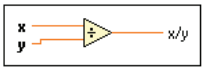



**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.

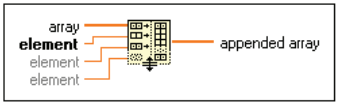
# CHAPTER THREE
# LABVIEW PROJECTS




## Divide Function

Computes the quotient of the inputs.

**Front Panel➔Numeric➔Divide**

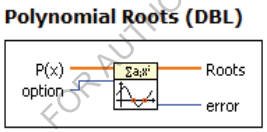



## Build Array Function



Concatenates multiple arrays or appends elements to an n-dimensional array.

**Block Diagram➔Array➔Build Array**




Finds the roots of polynomial **P(x)**. This VI removes leading coefficients of the polynomial that are equal to zero.

***P(x):*** contains the real polynomial coefficients in ascending order of power. **P(x)** cannot equal 0.

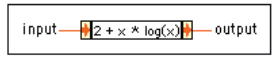
***option:*** specifies the option for root finding. The default is Simple Classification.

**Block Diagram ➔ R.C ➔ Mathmatics ➔ Polynomial ➔ Polynomial Roots**


**Front Panel➔Numeric➔Knob**

# CHAPTER THREE
# LABVIEW PROJECTS


**Block Diagram ➔ R.C ➔Array➔Array constant**




## Expression Node

Use the Expression Node to calculate expressions that contain a single variable. The following built-in functions are allowed in formulas: abs, acos, acosh, asin, asinh, atan, atanh, ceil, cos, cosh, cot, csc, exp, expm1, floor, getexp, getman, int, intrz, ln, lnp1, log, log2, max, min, mod, rand, rem, sec, sign, sin, sinc, sinh, sizeOfDim, sqrt, tan, tanh.

**input** is the value that the Expression Node uses as the variable.

**output** returns the value of the calculation.



## Procedure

### Formula node in mathematic operations:

### Part1

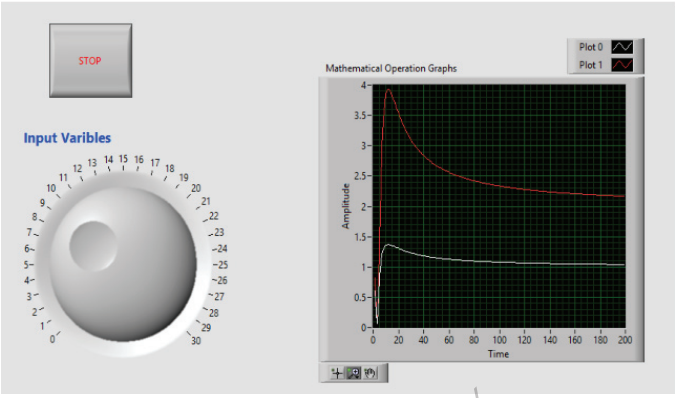
Type the following equations in the Formula Node :

$$a=Tanh(x) + Cos(x)$$

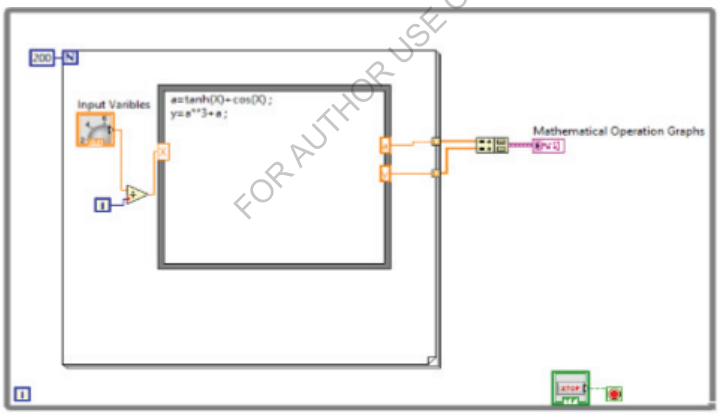
$$y= a^3 + a$$

## Front Panel

# CHAPTER THREE
# LABVIEW PROJECTS




## Block Diagram






## Part2

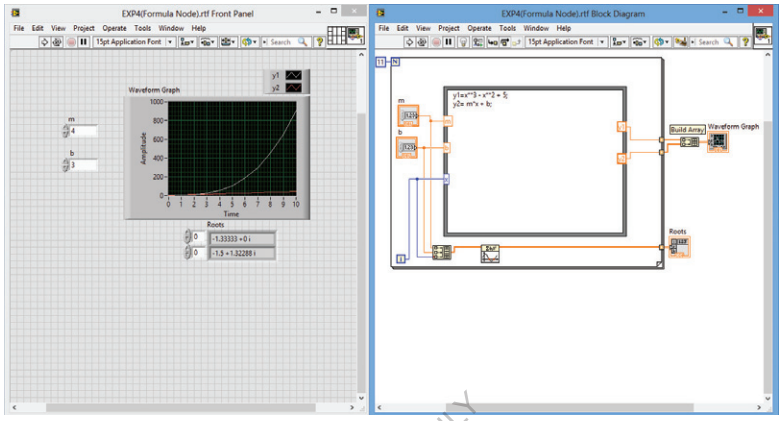
## Front Panel & Block Diagram

Write the following equations by using formula node:

$$y1=x^3+x^2+5$$

$$y2= m*x + b$$

# CHAPTER THREE
# LABVIEW PROJECTS




## Formula Node in Programming:

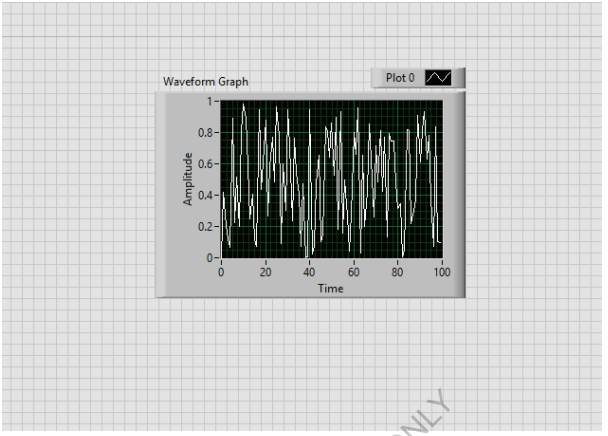
### Part1

The Formula Node, shown as follows, is a resizable box similar to the For Loop, While Loop, Case structure, Stacked Sequence structure, and Flat Sequence structure. However, instead of containing a sub diagram, the Formula Node contains one or more C-like statements delimited by semicolons, as in the following example. As with C, add comments by enclosing them inside a slash/asterisk pair (/*comment*/) or by preceding them with two slashes (//comment).



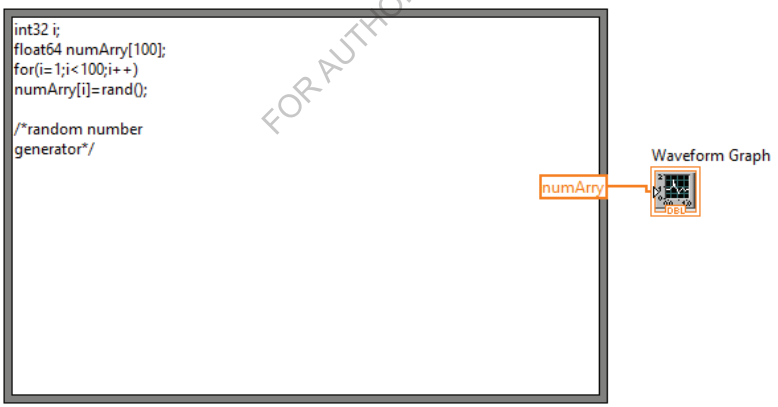
**Front Panel**

# CHAPTER THREE
# LABVIEW PROJECTS



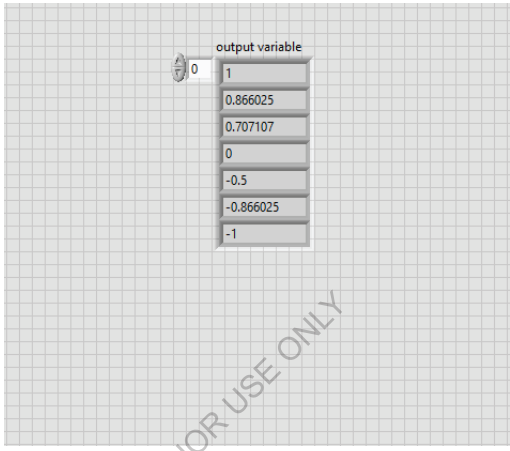



## Block Diagram



## Part2

### Front Panel






### Block Diagram

# CHAPTER THREE
# LABVIEW PROJECTS

## Discussion

1- excuite the following Equation using formula node:

answer1=(-B+$\sqrt{B^2 + 4AC)}$ )/(2$A$)

answer1=(-B+$\sqrt{B^2 - 4AC)}$ )/(2$A$)

2- y= q+p*x

w= $e^{\left(-0.5\left(\left(\frac{x-c}{s}\right)^2\right)\right.}$

a=a+wy

b=b+w

3- excuite the following Equation using formula node:

Xi=$\sqrt{(-2\ln(u_1)) * \cos(2\pi u_2)}$

Yi=$\sqrt{(-2\ln(u_1)) * \sin(2\pi u_2)}$



The inputs are random numbers and and the each output is array of size 1000.

4- excuite the following Equation using formula node:

F=$\frac{1}{\sqrt{5}}[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n]$

5- write the program by using formula node to check the multiply two numbers if the result is equal or less than 20 the LED is on else the led OFF.

CHAPTER THREE
LABVIEW PROJECTS

# Voltage Divider Rule

## Aim of experiment

To verify the voltage divider rule (VDR) by Using LabVIEW software.

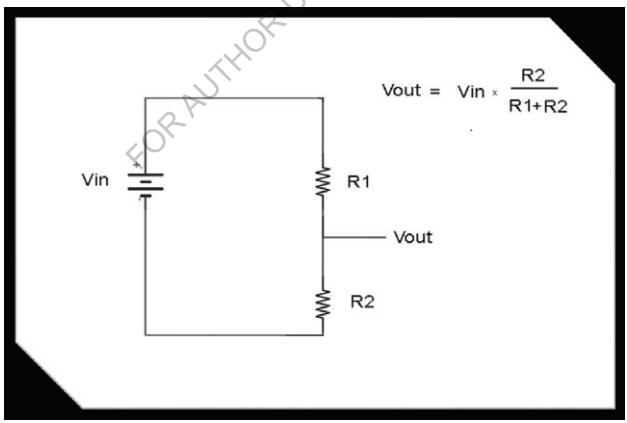
## Apparatus

LabVIEW software.

## Theory Voltage Divider Rule

The Voltage Divider Rule (VDR) states that the voltage across an element or across a **series** combination of elements in a **series** circuit is equal to the resistance of the element or **series** combination of elements divided by the total resistance of the series circuit and multiplied by the total impressed voltage.






In the general we can the voltage divider rule as

$$V_{out} = I\,R_n = \frac{E}{R1+R2+R3+\cdots+Rn} \times R_n \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(i)}$$

# CHAPTER THREE
# LABVIEW PROJECTS

Indicates that the voltage across any resistor $R_i$ ($R_i$, i= 1, 2... n) in a series circuits equal to the applied voltage (E) across the circuit multiplied by a factor $\frac{R_i}{\sum_{j=1}^{n} R_j}$

Actually, the VDR in figure (1) drive from equation bellow:

$$V_{in} = (R_1 + R_2)\text{I}\ldots\ldots\ldots\ldots\text{...By applying ohms Law.}$$

$$\text{I}=\frac{V_{in}}{R_1+R_2}\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ (1)$$

$$V_{out} = \text{I}R_2$$

$$V_{out} = \frac{V_{in}}{R_1+R_2}R_2\ldots\ldots\ldots\ldots\ldots\ldots\ (2)$$

It should be noted that this expression is only valid if the same current **I** flow through all the resistors.

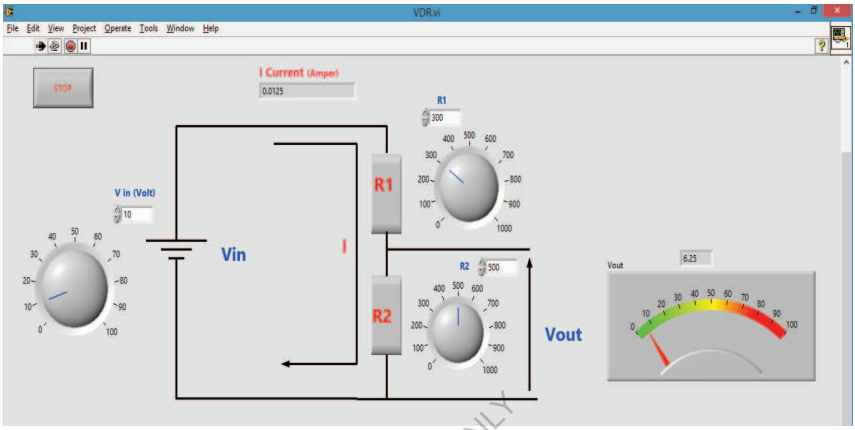


## Procedure

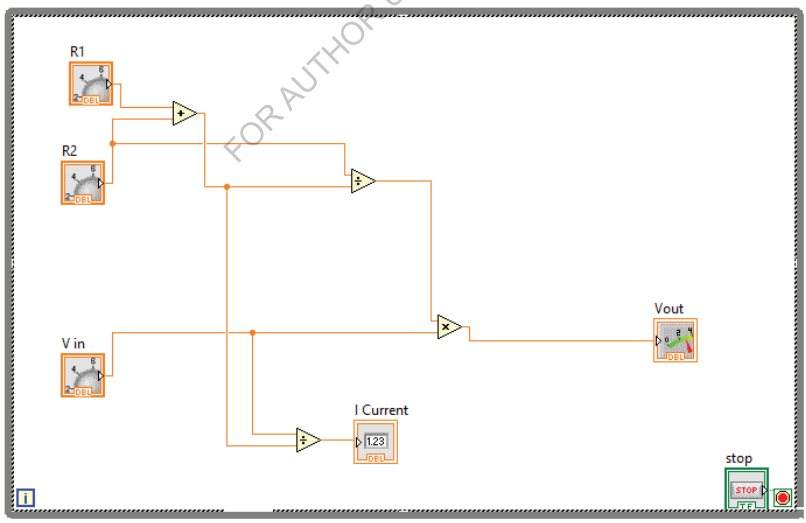
1. Design DC circuit in LabVIEW as shown in figure (2) and figure (3), connect the circuit shown in Fig. (1), take E =10V, $R_1$=82Ω, $R_2$= 100Ω
2. Measured the voltage $V_{out}$ and current of "$R_1$, $R_2$ ".
3. Exchange the value of resistors as following: R1= 10Ω, R2= 1000 Ω in figure (2).
4. Measured the voltage $V_{out}$ and current of "$R_1$, $R_2$ ".
5. Repeat step (2), change the value of resistors as following: R1=300Ω, R2= 500 Ω.
6. Measured the voltage $V_{out}$ and current of "$R_1$, $R_2$ ".

### Front Panel

# CHAPTER THREE
# LABVIEW PROJECTS




## Block Diagram



## Discussion

1. Determine I1, I2, Iout when apply: R1= 90 KΩ, R2= 95 KΩ and Ein=40 V.

2. Compare between the practical and theoretical results by applying formula in (1) and (2).

3. Determine Vout when apply: variable R1= (0-90) KΩ, R2= 95 KΩ and variable Ein= (0-60) V.

4. Repeat the step in (3) and plot the result for each period as shown in table bellow

| NO. | $V_{in}$volt | $R_1$KΩ | $R_2$KΩ | $V_{out}$ volt |
|---|---|---|---|---|
| 1 | 0 | 0 | 95 | |
| 2 | 10 | 10 | 95 | |
| 3 | 20 | 20 | 95 | |
| 4 | 30 | 30 | 95 | |
| 5 | 40 | 40 | 95 | |
| **6** | 50 | 50 | 95 | |
| **7** | 60 | 60 | 95 | |
| **8** | 60 | 70 | 95 | |
| **9** | 60 | 80 | 95 | |
| **10** | 60 | 90 | 95 | |



5. An experiment conducted to measure 10 values of voltages and the result is shown in the table in step (4). Calculate the accuracy of the 9th experiment

6. If analog voltmeter is used to measure voltage of 40V across a resistor R2. The reading value is 39 V. Find: A. absolute Error B. relative Error C. Accuracy D. Percent Accuracy.

7. Plot Vout in step (4) using Waveform Chart and register all variable in front panel.

CHAPTER THREE
LABVIEW PROJECTS

# Current Divider Rule

## Aim of experiment

To verify the current divider rule (CDR) by Using LabVIEW software.

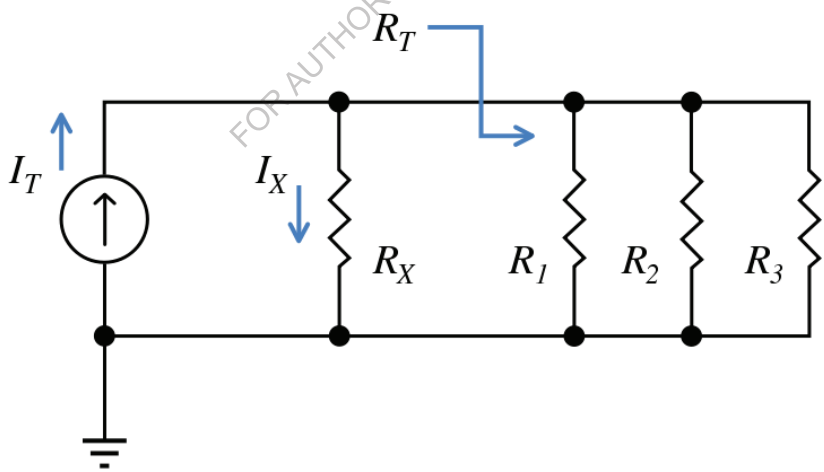
## Apparatus

LabVIEW software.

## Theory Voltage Divider Rule

The Current Divider Rule (CDR) states that the current through one of two parallel branches is equal to the resistance of the other branch divided by the sum of the resistances of the two parallel branches and multiplied by the total current entering the two parallel branches.





# CHAPTER THREE
# LABVIEW PROJECTS

A general formula for the current *Ix* in a resistor *Rx* that is in parallel with a combination of other resistors of total resistance $R_T$ as shown in Figure (1):

$$I_X = \frac{R_T}{R_X + R_T} \times I_T \quad \text{..... (i)}$$

where *IT* is the total current entering the combined network of *RX* in parallel with *RT*. **Notice** when *RT* is composed of a **parallel combination** of resistors, say *R1*, *R2*, ... *etc.*, then the reciprocal of each resistor must be added to find the total resistance *RT*:

$$\frac{1}{R_T} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_n} \quad \text{..... (1)}$$

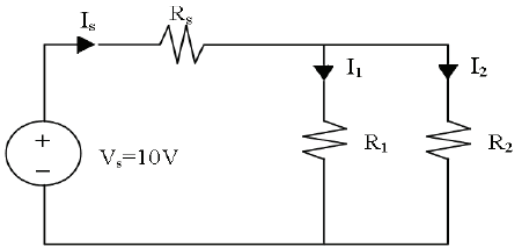
$$I_X = \frac{\frac{1}{R_X}}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}} I_T \quad \text{..... (2)}$$



It should be noted that this expression is only valid if the same Voltage **V** flows through all the resistors.

## Procedure

1. Design DC circuit in LabVIEW as shown in figure (3) and figure (4), connect the circuit shown in Fig. (2), take $V_S$ =10V, $R_s$= 10 Ω, $R_1$=82Ω, $R_2$ = 100Ω.

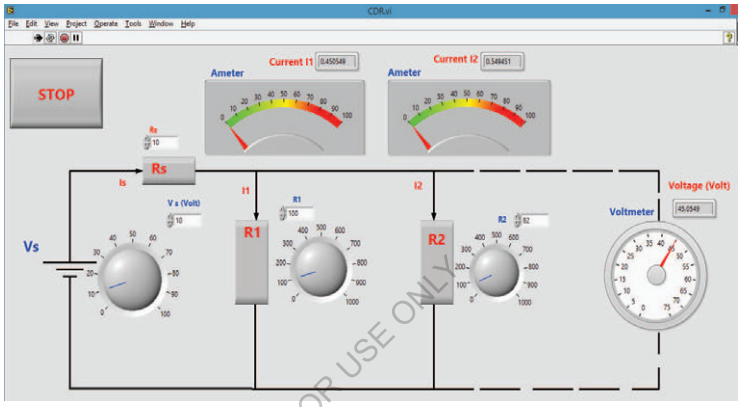



2. Measured the voltage V and current of "R1, R2 ".
3. Exchange the value of resistors as following: R1= 10Ω, R2= 1000 Ω in figure (3).
4. Measured the voltage V and current of "R1, R2 ".

5. Repeat step (2), change the value of resistors as following: R1=300Ω, R2= 500Ω.

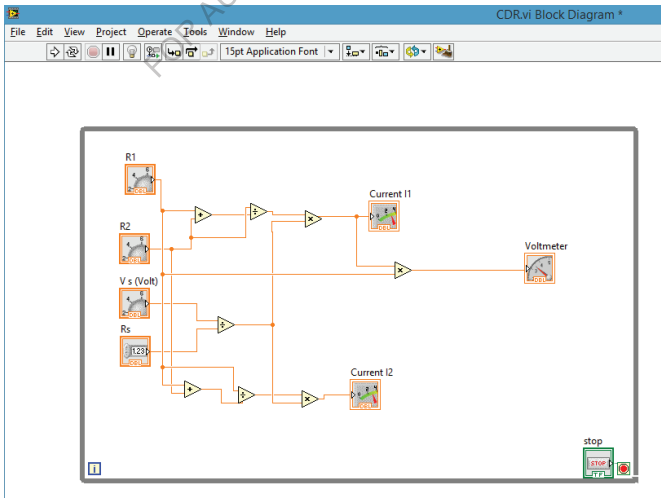
6. Measured the voltage V and current of "R1, R2 ".

## Front Panel






## Block Diagram



# CHAPTER THREE
# LABVIEW PROJECTS

## Discussion

1. Redesign DC circuit in LabVIEW as shown in figure (3) and figure (4) and determine V1, V2, I1, I2 when apply: R s=40 Ω R1= 90 KΩ, R2= 95 KΩ, VS=40 V.

2. Compare between the practical design in LabVIEW and theoretical results by applying formula in (i).

3. Determine I1, I2 when apply: variable R1= (0-90) KΩ, R2= 95 KΩ, R s=60 Ω and variable Vs.= (0-60) V.

4. Repeat the step in (3) and plot the result for each period as shown in table bellow

| NO. | $V_{in}$volt | $R_1$KΩ | $R_2$KΩ | I1 mAmpere | I2 mAmpere | $V_{out}$ volt |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 95 | | | |
| 2 | 10 | 10 | 95 | | | |
| 3 | 20 | 20 | 95 | | | |
| 4 | 30 | 30 | 95 | | | |
| 5 | 40 | 40 | 95 | | | |
| **6** | 50 | 50 | 95 | | | |
| **7** | 60 | 60 | 95 | | | |
| **8** | 60 | 70 | 95 | | | |
| **9** | 60 | 80 | 95 | | | |
| **10** | 60 | 90 | 95 | | | |



5. The accuracy of five digital ammeter are checked by using each of them to measure a standard 1.0000 Ampere from a calibration instrument. The ammeter readings are as follows: *I*1 =1.001 Ampere, *I*2 =1.002 Ampere, *I*3 = 0.999 Ampere, *I*4 = 0.998 Ampere,

and *I*5 =1.000 Ampere. Calculate the average measured ampere and the average deviation and also determine the standard deviation of the measurement

6. Plot I1, I2 in step (4) using Waveform Chart and register all variable in front panel.

CHAPTER THREE
LABVIEW PROJECTS

# Local and Global Variable

## Aim of Experiment

The objective of this exercise to learn how to use local and Global variables in LABVIEW.

## Apparatus

LabVIEW software.

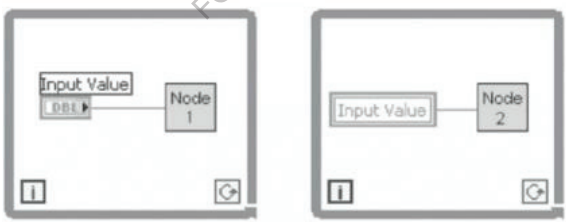
## Theory

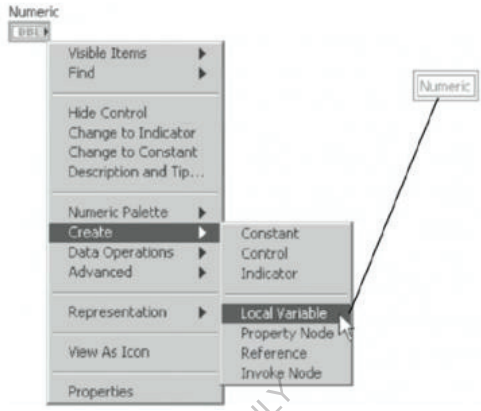
- **Local Variables in LabVIEW**

  Local variables transfer data within a single VI and allow data to be passed between parallel loops as shown in Figure 4.22. They also break the dataflow programming paradigm.

  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 as shown below.

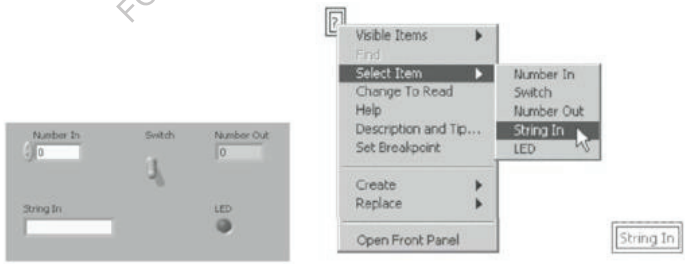







Another way is to select the Local Variable from the Structures palette. Create the front panel and select a local variable from the Functions palette and place it on the block diagram. The local variable node, shown as follows, is not yet associated with a control or indicator. To associate a local variable with a control or indicator, right-click the local variable node and select Item from the shortcut menu.

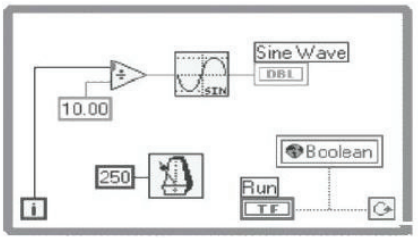


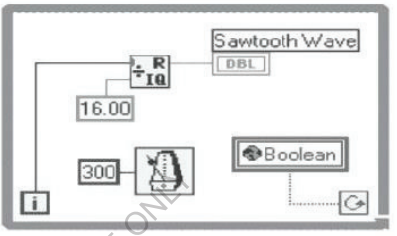


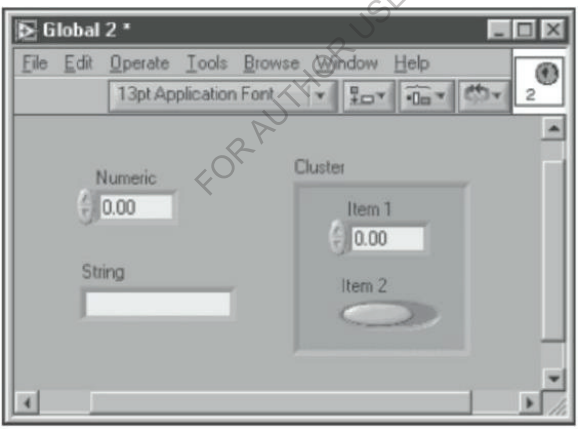

- **Global Variables in LabVIEW**

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. For example,

suppose you have two VIs running simultaneously. Each VI contains a While Loop and writes data points to a waveform chart. The first VI contains a Boolean control to terminate both VIs. You can use a global variable to terminate both loops with a single Boolean control as shown in Figure 4.25. If both loops were on a single block diagram within the same VI, you could use a local variable to terminate the loops.



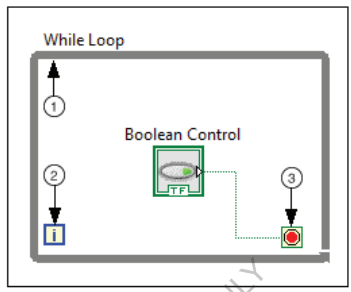



## The Experiment Blocks

While Loop

Repeats the code within its sub diagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**




**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.



**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.

**Block Diagram➔R.C➔Structure➔Case Structure**


## Local Variable

Use local variables to read or write to one of the controls or indicators on the front panel of a VI.

**Block Diagram➔R.C➔Structure➔Case Structure➔Local Variable**

# CHAPTER THREE
# LABVIEW PROJECTS


## Global Variable

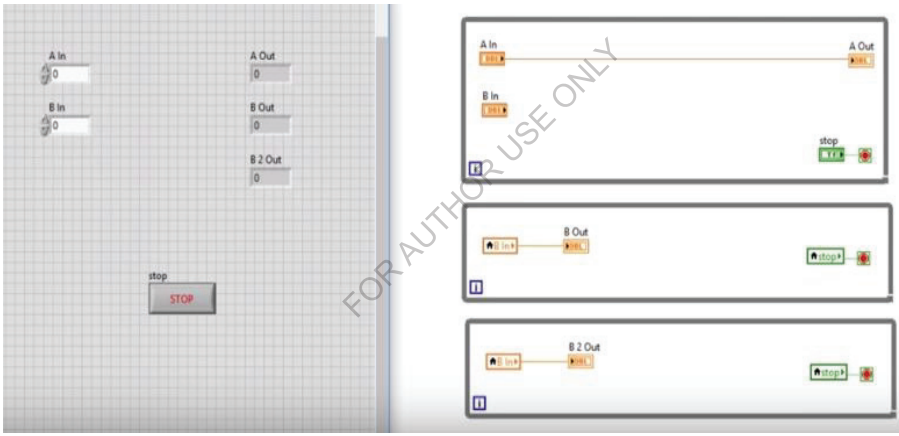
Use global variables to access and pass data among several VIs.

**Block Diagram➔R.C➔Structure➔Case Structure➔Global Variable**

## Procedure

### Local Variable

### Front Panel& Block Diagram

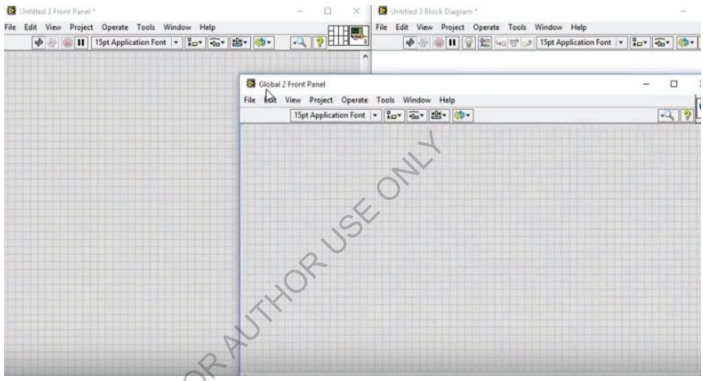



***The complete design***

### Global Variable

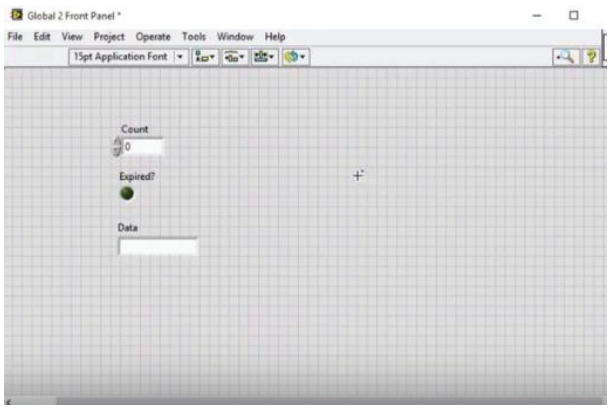
a. **Block Diagram➔R.C➔Structure➔Case Structure➔Global Variable**
b. **Click on the global variable function.**

**c. the new VI appear as shown below:**

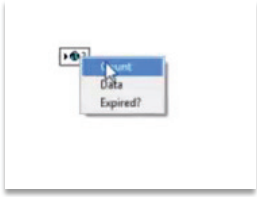


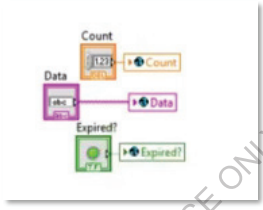


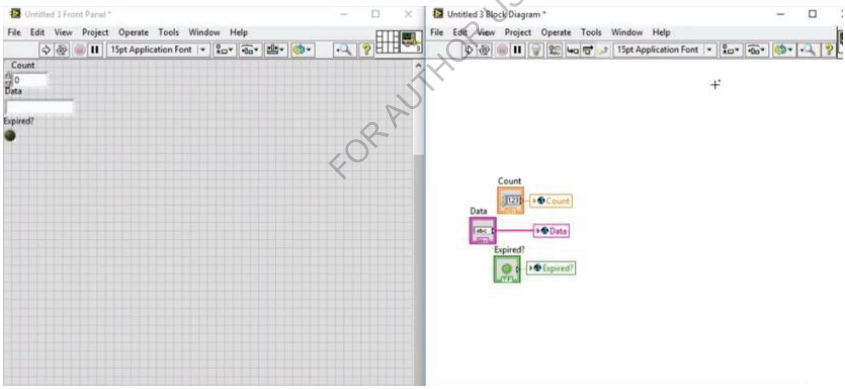

**d. Create the program inside the global function block.**



**e. Click on the global variable and select the variable.**






**f. Great the another VI.**




**g. Front Panel& Block Diagram**

# CHAPTER THREE
# LABVIEW PROJECTS


## Discussion

1- Make a design that contain local and global variable.

CHAPTER THREE
LABVIEW PROJECTS

# Traffic light

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW for execute many cases and design traffic light system.

## Apparatus

LabVIEW software.

## Procedure

### (Traffic light):

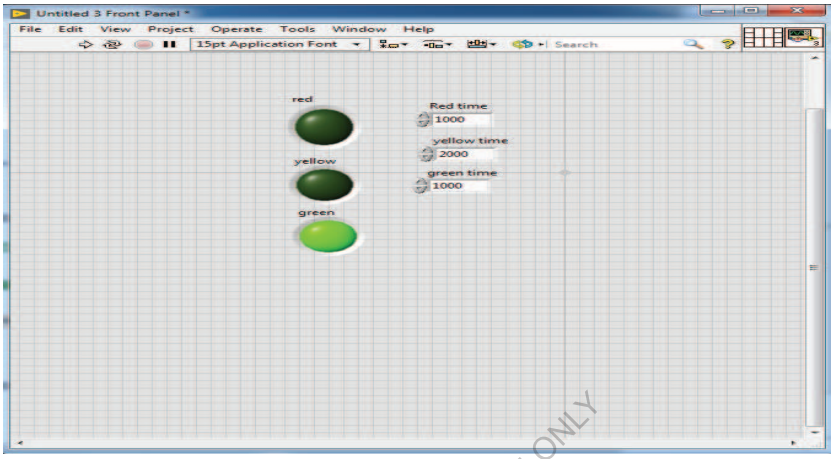
Create a Sub VI that to design the traffic light system.

1. Create the Sub VI.
2. Create the Front Panel and the Block Diagram.
3. Create necessary Connectors as shown below.
4. Run the program to see if it works.



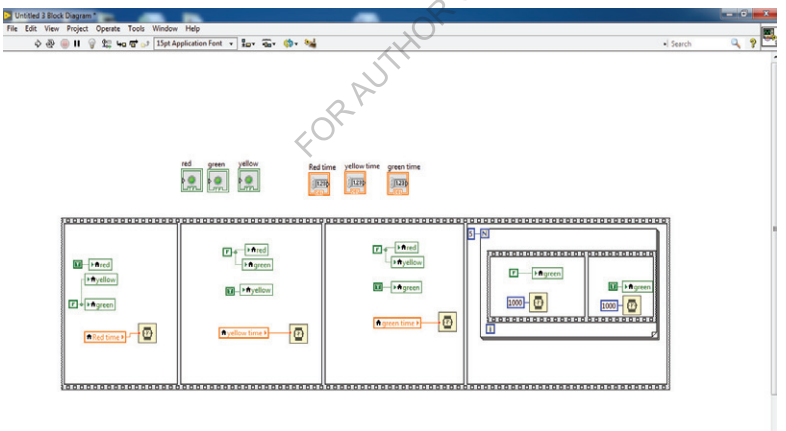
**Front Panel**

# CHAPTER THREE
# LABVIEW PROJECTS






## Block Diagram




## Solution:

1. On the Block diagram R.C ➔ Timing➔ Wait(ms).
2. On the Block diagram R.C ➔ structure➔ local variable.

3. On the Block diagram R.C➔structure➔flat sequence.
4. On the Block diagram R.C on the frame of flat sequence and select **add frame after**.
5. On the Block diagram R.C➔ Boolean➔ F
6. On the Block diagram R.C ➔Boolean➔ T
7. On the front panel R.C ➔Boolean➔ Round Led.

## Discussion

1- Using LabVIEW to Design the blinking led system every 1000 seconds.
2- Using LabVIEW to Design the traffic light system in another way.
3- Using LabVIEW to Design the Two traffic light system.

CHAPTER THREE
LABVIEW PROJECTS

# Tank Level Sensor

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW for design Tank Level Sensor system.

## Apparatus

LabVIEW software.

## Procedure

### (Tank Level Sensor): Part1

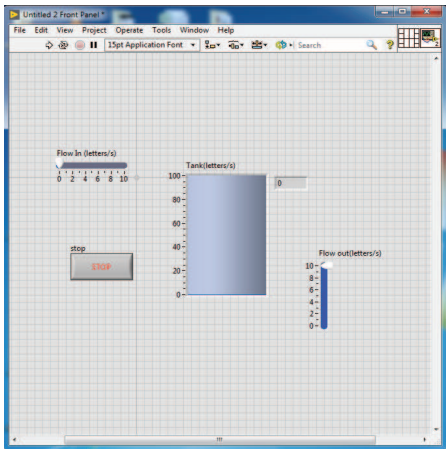
Create a Sub VI that to design the Tank Level Sensor system.

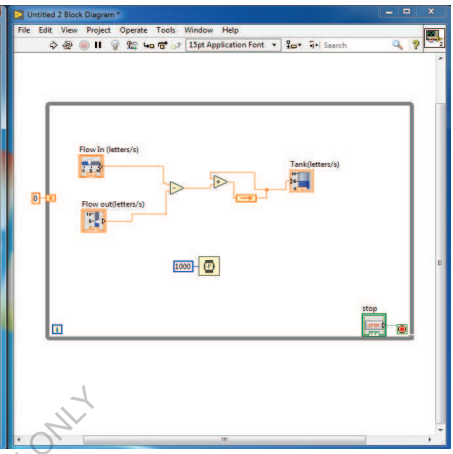
1. Create the Sub VI.
2. Create the Front Panel and the Block Diagram.
3. Create necessary Connectors as shown below.
4. Run the program to see if it works.



### Front Panel & Block Diagram

# CHAPTER THREE
# LABVIEW PROJECTS



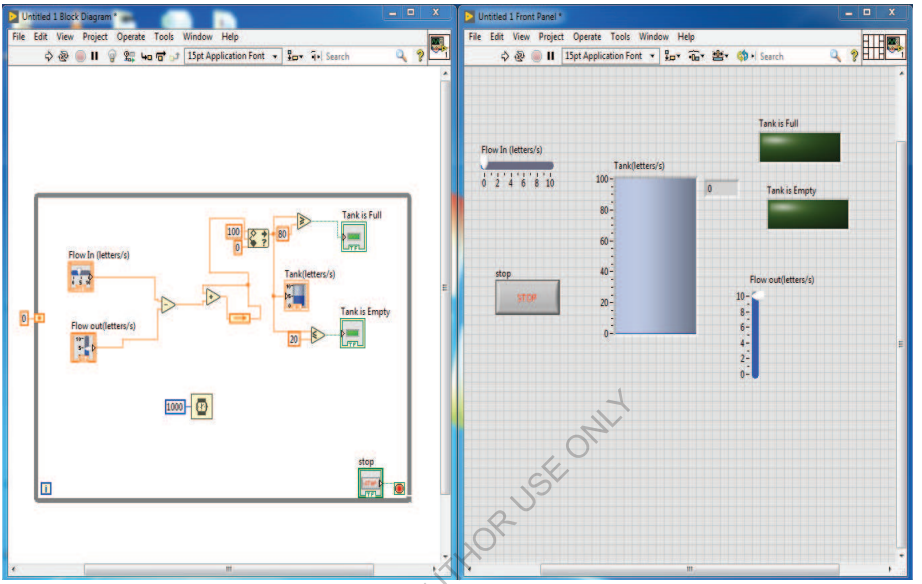



## Solution:

1. On the Block diagram R.C➔Timing➔ Wait(ms).
2. On the Block diagram R.C ➔Numeric➔ Subtract.
3. On the Block diagram R.C ➔Numeric➔ Add.
4. the Block diagram R.C ➔structure➔ while loop.
5. On the Block diagram R.C on the red button of while ➔loop➔Great control.
6. On the front panel R.C ➔Numerical ➔Vertical pointer slide.
7. On the front panel R.C ➔Numerical ➔Horizontal pointer slide.

## Part2

**Front Panel & Block Diagram**






Solution:

1. On the front panel R.C ➔Boolean ➔Square Led.
2. On the Block diagram R.C ➔Comparison ➔Greater or Equal to?.
3. On the Block diagram R.C ➔Comparison ➔Less or Equal to?.
4. On the Block diagram R.C ➔Comparison ➔In Range and Coerce.

## Discussion

1. The system is designed using the LabVIEW to control the water level of the tank. Compare two fixed and variable values. If the specified value is less than the variable value, the light is off but if not, the light is ON.

   Note: When the level of the tank reaches the critical value, the program stops.

2. Design the program by using LabVIEW to control the water level of the tank. Select the highest level and lowest level of water. If the water reaches the highest level, (max alarm) ON and if it reaches the lowest level (min alarm) ON.

CHAPTER THREE
LABVIEW PROJECTS

# check Username and Password

## Aim of Experiment

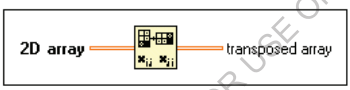
The objective of this exercise to learn how to use LabVIEW for check the username and password.

## Apparatus

LabVIEW software.

## Blocks of this Experiment:

1.

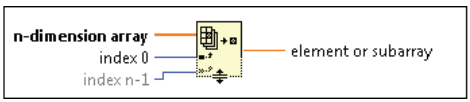



**Transpose 2D Array Function**



- Rearranges the elements of **2D array** such that **2D array**[i,j] becomes **transposed array**[j,i].
- The connector pane displays the default data types for this polymorphic function.

2.

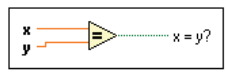



**Index Array Function**

- Returns the **element or subarray** of **n-dimension array** at **index**.

- When you wire an array to this function, the function resizes automatically to display **index** inputs for each dimension in the array you wire to **n-dimension array**.

3.

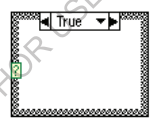



Equal? Function

- Returns TRUE if **x** is equal to **y**. Otherwise, this function returns FALSE.

4.






**Case Structure**

- Contains one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the case selector determines which case to execute.

5.


**For loop**

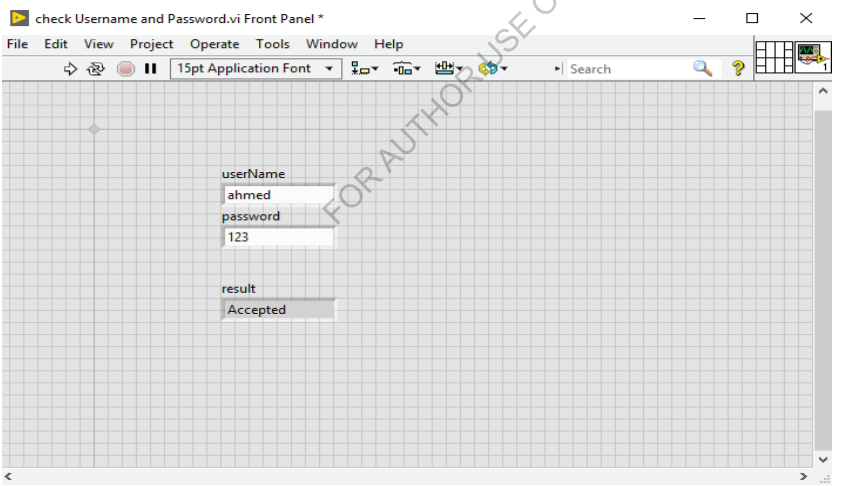
- Executes its subdiagram n times, where *n* is the value wired to the count (**N**) terminal. The iteration (**i**) terminal provides the current loop iteration count, which ranges from 0 to *n*-1.

## Procedure

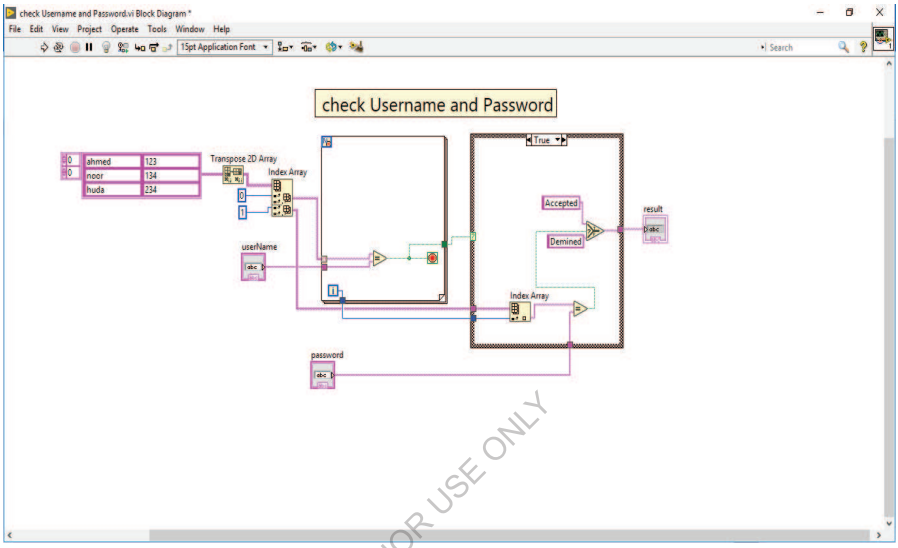
Create a Sub VI that to design the traffic light system.

1. Create the Sub VI.
2. Create the Front Panel and the Block Diagram.
3. Create necessary Connectors as shown below.
4. Run the program to see if it works.



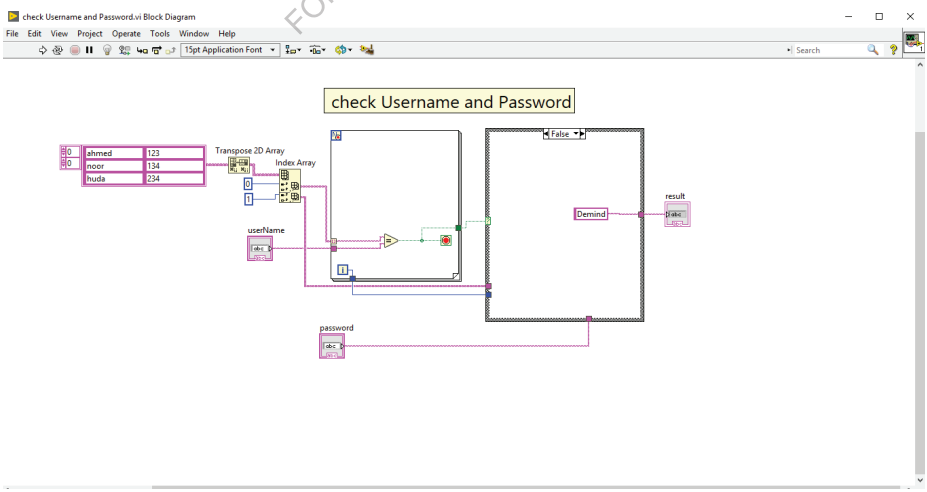



### True State






## False State



## Solution:

1. On the Block Diagram R.C➔ string ➔String constant.
2. On the Block Diagram R.C ➔Array➔Transpose 2D array.
3. On the Block Diagram R.C➔ Array➔ index array.
4. On the Block Diagram R.C➔structure➔ for loop.
5. On the Block diagram R.C on the for loop and select **conditional terminal**.
6. On the Block diagram R.C on the for-loop output ➔Tunnel mode
7. Last value.
8. On the Block Diagram R.C ➔ structure ➔ case structure.
9. On the Block Diagram R.C ➔ Comparison➔ select.

## Discussion



1. Use LABVIEW program to Design the program to find the length of the specific string. The string is (University).
2. Use LABVIEW program to Design the program to enter the two strings. The first string is(system) and the second string is (Department). Put these two strings in the same string and the output shall be(systemDeparment).
3. Use LABVIEW program to Design the program to find the subset from the string. The input string is your name (first&last name) and the output string is your last name.
4. Use LABVIEW program to Design the program to enter the string (Al-Nahrain University) and Replace the (University) word with a (system).
5. Use LABVIEW program to Design the program to convert the lower-case word to high case word and vice versa.

network ⟷ NETWORK

6. Use LABVIEW program to Design the program to convert the hexa **(ABCD)** to number and vice versa.

CHAPTER THREE
LABVIEW PROJECTS

# Marks Average

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW calculate the marks average.

## Apparatus

LabVIEW software.

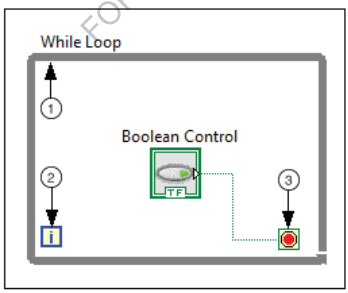
## The Experiment Blocks

### While Loop



Repeats the code within its sub diagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**

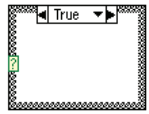



**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to

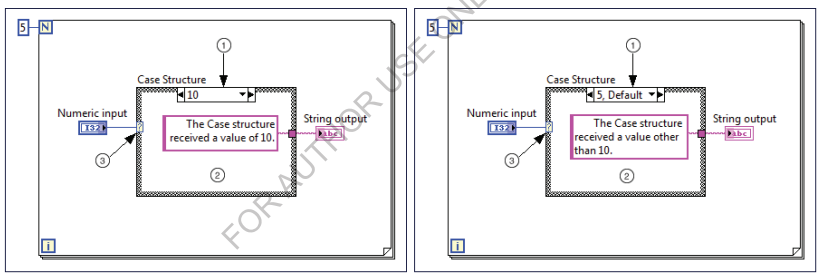
keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.



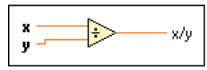

Contains one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the case selector determines which case to execute.

**Components of a Case Structure**





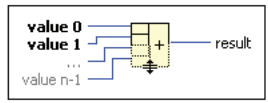

**Block Diagram➔R.C➔Structure➔Case Structure**




## Divide Function

Computes the quotient of the inputs

**Block Diagram➔R.C➔Numeric➔Divide**



# CHAPTER THREE
# LABVIEW PROJECTS

## Compound Arithmetic Function

Performs arithmetic on one or more numeric, array, cluster, or Boolean inputs. To select the operation (Add, Multiply, AND, OR, or XOR), right-click the function and select **Change Mode** from the shortcut menu. When you select this function from the Numeric palette, the default mode is Add. When you select this function from the Boolean palette, the default mode is OR.

**Block Diagram➔R.C➔Numeric➔Compound Arithmetic**


## Number to Decimal String Function

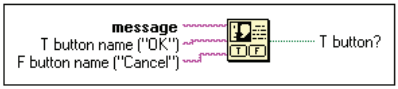


Converts **number** to a string of decimal digits at least **width** characters wide or wider if necessary. If **number** is floating-point or fixed-point, it is rounded to a 64-bit integer before conversion.

**number** can be a scalar number, array or cluster of numbers, array of clusters of numbers, and so on.

**width** must be numeric. If unwired, the function uses exactly as many digits as are needed to represent the number, with no extra padding.

**decimal integer string** is the resulting decimal string. The following table shows how the values of **number** and **width** affect **decimal integer string**. In this table, the underline character ( _ ) represents spaces in **decimal integer string**.




## Two Button Dialog Function

Displays a dialog box that contains a message and two buttons.

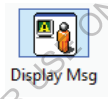
**message** is the text to display in the dialog box.

**T button name** is the name displayed on one of the dialog box buttons. The default is **OK**.

**F button name** is the name displayed on one of the dialog box buttons. The default is **Cancel**.

**T button?** returns a value of TRUE if you click the dialog box button named **T button name**. If you click the dialog box button named **F button name** or click the close window button, **T button?** returns a value of FALSE.

**Block Diagram➔R.C➔ Dialog & User Interface➔Two Btn Dialog**

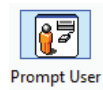





Display Message to User Express VI

Displays a standard dialog box that contains an alert or a message for users.

**Block Diagram➔R.C➔Dialog & User Interface➔Display Msg**



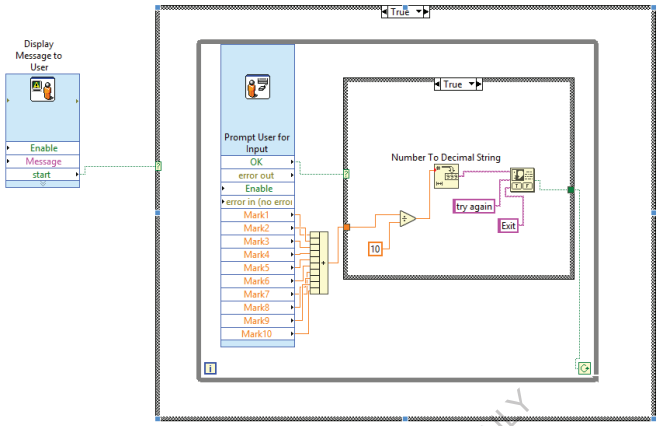

**Prompt User for Input Express VI**

Displays a standard dialog box that prompts users to enter information, such as a user name and password.

**Block Diagram➔R.C➔Dialog & User Interface➔Prompet User**

## Procedure

### Block Diagram




***The complete design***



## Discussion

1- By using LabVIEW program make a simple calculator.

CHAPTER THREE
LABVIEW PROJECTS

# Shift Register

## Aim of Experiment

The objective of this exercise to learn how to use Shift Register to shift data in LabVIEW.

## Apparatus

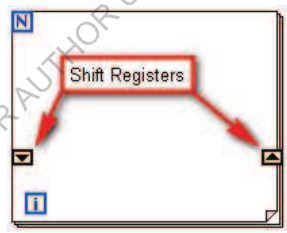
LabVIEW software.

## Theory

### Iterative Data Transfer

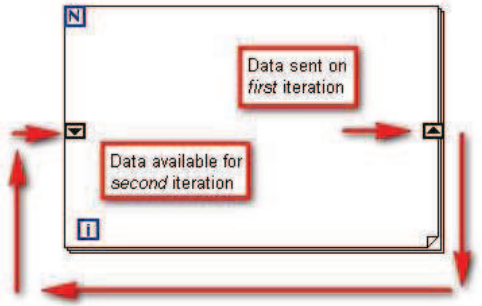
When programming with loops, sometimes you need to call data from previous iterations of the loop. In LabVIEW, you can use shift registers, which are similar to static variables in text-based programming languages, to pass values from one loop iteration to the next.






Data enters the right shift register and is passed to the left shift register on the next iteration of the loop.

# CHAPTER THREE
# LABVIEW PROJECTS




## Tutorial: Using Shift Registers

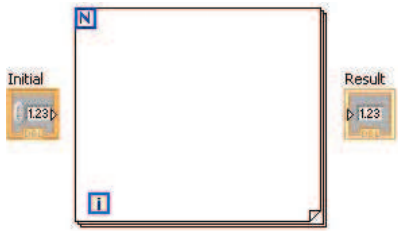
The next section of this tutorial guides you through creating and using shift registers in a simple LabVIEW VI.

1. Create a new LabVIEW VI by navigating to File»New VI.
2. Place a numeric control on the front panel and change its value to 2.
3. Double-click on the control's name and change it to "Initial."

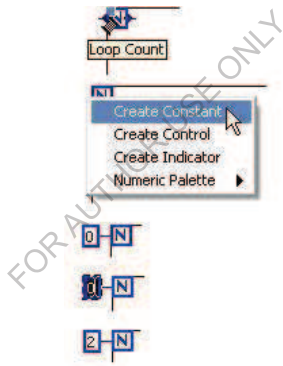



4. Place a numeric indicator on the front panel and name it "Result."


5. View the block diagram by selecting Window»Show Block Diagram or pressing <ctr-E>.
6. Place a for loop on the block diagram between the numeric control and indicator. The for loop is located atFunctions»Programming»Structures»For Loop.




7. Right-click on the input of the count terminal of the for loop and select Create Constant. Change the value of this constant to 2.



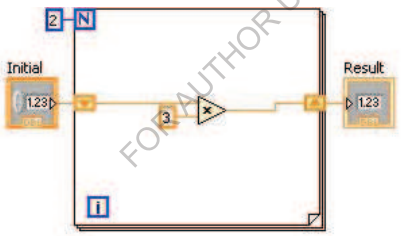



8. Wire the output of the Initial control to the right edge of the for loop to create a tunnel.

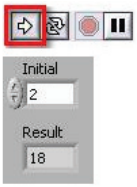
9. Right-click on the tunnel that you just created and select Replace with Shift Register.


10. Wire the output of the right shift register to the Result indicator.
11. Place a multiply function in the for loop.
12. Place a numeric constant in the for loop, assign it a value of 3, and connect it to one of the input terminals of the multiply function.
13. Wire the left shift register to the remaining input of the multiply function, and wire the output of the function to the right shift register.






14. View the block diagram by selecting Window»Show Front Panel or pressing <ctr-E>.
15. Run the VI. The VI changes the value of the Result indicator to 18.




**Explanation**

Shift registers are integral to this VI. To understand how the VI works, you can step through the code.

Because the for-loop's counter terminal is wired to a constant of 2, it runs twice. On the first iteration of the for loop, the value of Initial, 2, is multiplied by 3. The result is 6, and this value is passed to the right shift register. On the second iteration of the for loop, the left shift register receives the value that was sent to the right shift register on the previous iteration, 6. The value of 6 is multiplied by 3, for a result of 18. Because the for loop completed all of its iterations, it stops running and the value of 18 is sent to the Result indicator on the front panel.

The mathematical formula for this simple VI looks like this:

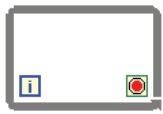
Result = ((Initial * 3) * 3)

If you changed the value of the for-loop's count terminal to 4, the mathematical formula looks like this:

Result = ((((Initial * 3 ) * 3 ) * 3 ) * 3 )



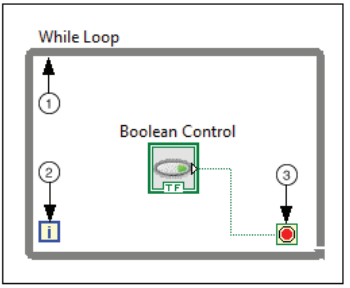
## The Experiment Blocks




### While Loop

Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**

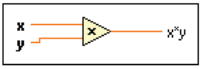



**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

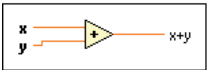


**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.



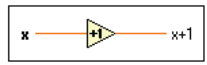

Returns the product of the inputs.

**Block Diagram➔R.C➔Numeric➔Multiply**



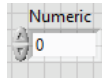

Computes the sum of the inputs.

**Block Diagram➔R.C➔Numeric➔Add**




**Block Diagram➔R.C➔Numeric➔Increment**


**Front Panel➔Numeric➔Numeric Indicator**



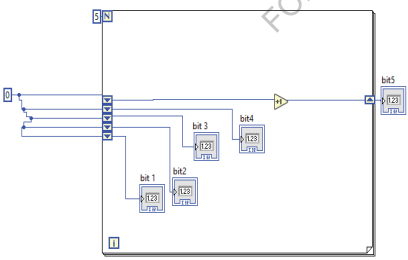

**Front Panel➔Numeric➔Numeric Control**

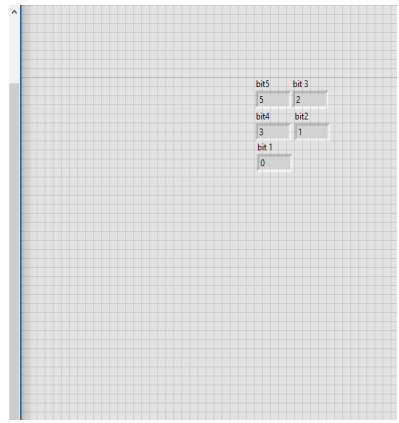
## Procedure

### Part1

### Front Panel& Block Diagram

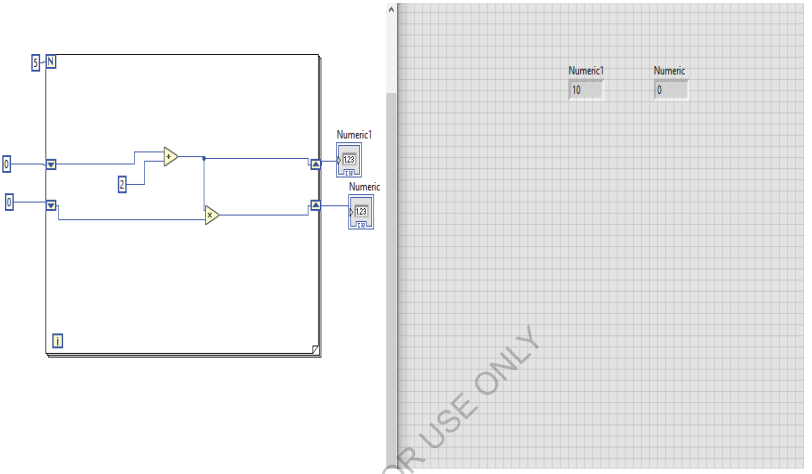





*The complete design*

## Part2






## Discussion

1- Design the simple project that used the shift register in it by using LABVIEW.

CHAPTER THREE
LABVIEW PROJECTS

# Home Automation

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW to design home automation.

## Apparatus

LabVIEW software.

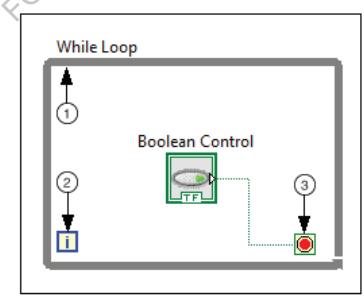
## The Experiment Blocks


**While Loop**



Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**

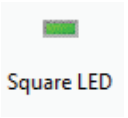



**Subdiagram**—Contains code that the While Loop executes once per iteration.

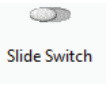
**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to

keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

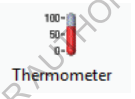
**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.



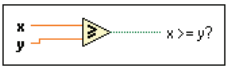

**Front Panel➔R.C➔Boolean➔Square Led**




**Front Panel➔R.C➔Boolean➔Slide Switch**




**Front Panel➔Numeric➔Thermometer**

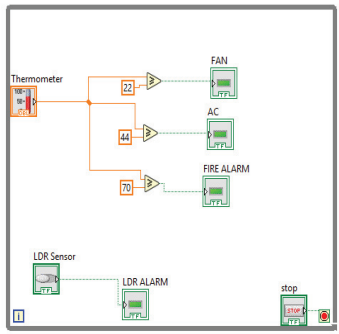



## Greater Or Equal? Function

Returns TRUE if **x** is greater than or equal to **y**. Otherwise, this function returns FALSE.

## Procedure

### Front Panel& Block Diagram




***The complete design***

## Discussion

1- Design another home automation by using LABVIEW.

CHAPTER THREE
LABVIEW PROJECTS

# Proximity and Distance Sensors

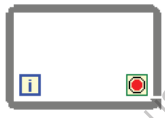
## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW to measure the distance by using distance sensors.

## Apparatus

LabVIEW software.

## The Experiment Blocks

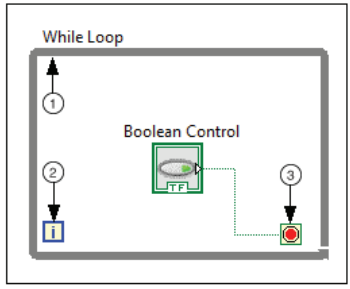


### While Loop

Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.



**Components of a While Loop**

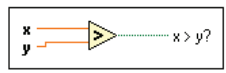



**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1,

the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

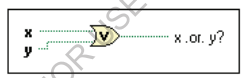
**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.




Greater? Function

Returns TRUE if **x** is greater than **y**. Otherwise, this function returns FALSE.

**Block Diagram➔R.C➔Comparison➔Greater**

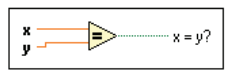



Or Function



Computes the logical OR of the inputs. Both inputs must be Boolean values, numeric values, or error clusters. If both inputs are FALSE, the function returns FALSE. Otherwise, it returns TRUE.

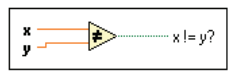
**Block Diagram➔R.C➔Boolean➔Or**




Equal? Function

Returns TRUE if **x** is equal to **y**. Otherwise, this function returns FALSE.

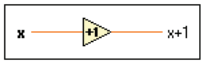
**Block Diagram➔R.C➔Comparison➔Equal?**




Not Equal? Function

Returns TRUE if **x** is not equal to **y**. Otherwise, this function returns FALSE.

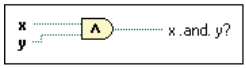
**Block Diagram➔R.C➔Comparison➔Not Equal?**




Increment Function

Adds 1 to the input value.

**Block Diagram➔R.C➔Numeric➔Increment**

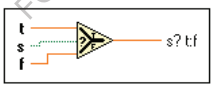



And Function



Computes the logical AND of the inputs. Both inputs must be Boolean values, numeric values, or error clusters. If both inputs are TRUE, the function returns TRUE. Otherwise, it returns FALSE.

**Block Diagram➔R.C➔Boolean➔And**



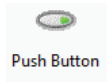

Select Function

Returns the value wired to the t input or f input, depending on the value of s. If s is TRUE, this function returns the value wired to t. If s is FALSE, this function returns the value wired to f.

**Block Diagram➔R.C➔Comparison➔Select**

# CHAPTER THREE
# LABVIEW PROJECTS

**Front Panel ➔R.C➔Numeric➔Vertical Pointer Slide**

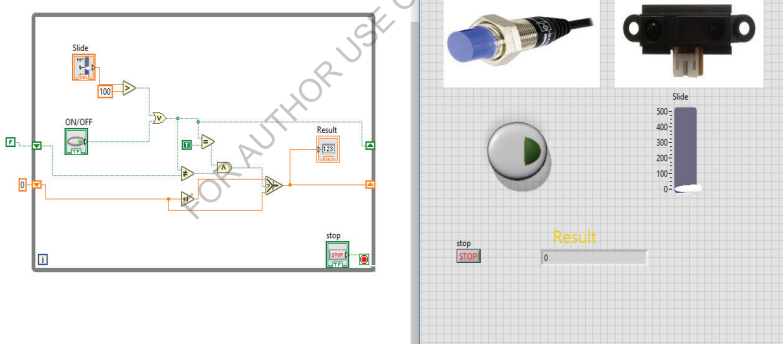



**Front Panel ➔R.C➔Boolean➔Push Button**


**Front Panel➔Numeric➔Numeric Indicator**

## Procedure

### Front Panel& Block Diagram






***The complete design***

## Discussion

1- Make the design by using other sensors and display the results by using LABVIEW program.

CHAPTER THREE
LABVIEW PROJECTS

# Binary Converter

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW to measure the distance by using distance sensors.

## Apparatus

LabVIEW software.

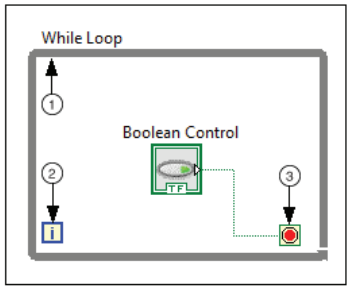
## The Experiment Blocks

### While Loop



Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**

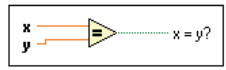



**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1,

the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.



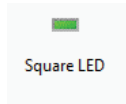

Equal? Function

Returns TRUE if **x** is equal to **y**. Otherwise, this function returns FALSE.

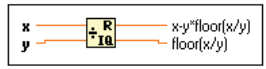
**Block Diagram➔R.C➔Comparison➔Equal?**




**Front Panel ➔R.C➔Numeric➔Vertical Pointer Slide**




**Front Panel ➔R.C➔Boolean➔Square Led**

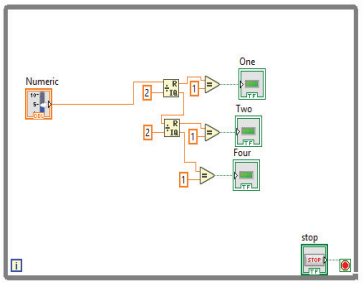


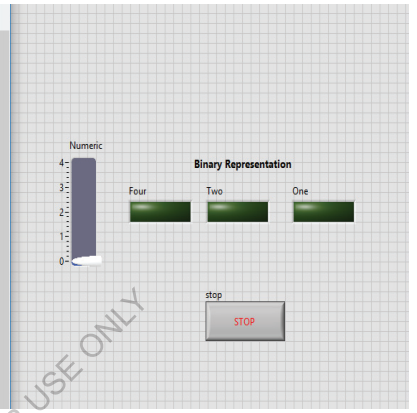

Quotient & Remainder Function

Computes the integer quotient and the remainder of the inputs. This function rounds floor(x/y) to the nearest integer towards -inf.

## Procedure

### Front Panel& Block Diagram




***The complete design***



## Discussion

1- Redesign the procedure and make it from 1-9.

CHAPTER THREE
LABVIEW PROJECTS

# Counter Up/Down

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW Timers and make the simple example to make a timer.

## Apparatus

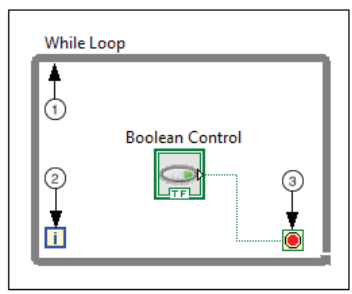
LabVIEW software.

## The Experiment Blocks

### While Loop

Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

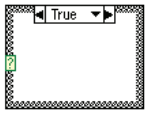
**Components of a While Loop**




**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

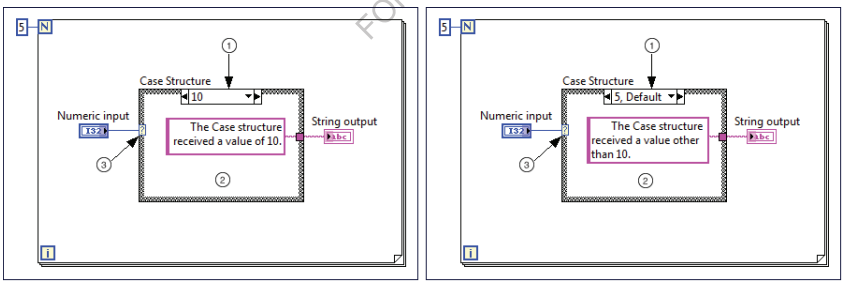
**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.



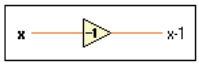



Contains one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the case selector determines which case to execute.

**Components of a Case Structure**



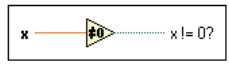

**Block Diagram➔R.C➔Structure➔Case Structure**




**Decrement Function**

Subtracts 1 from the input value.

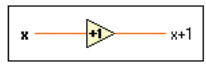
**Block Diagram➔R.C➔Numeric➔Decrement**




## Not Equal To 0? Function

Returns TRUE if x is not equal to 0. Otherwise, this function returns FALSE.

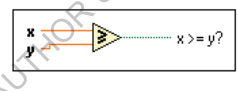
**Block Diagram➔R.C➔Comparation➔NO Equal To 0?**




## Increment Function

Adds 1 to the input value.

**Block Diagram➔R.C➔Numeric➔Increment**

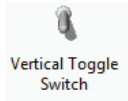



## Greater Or Equal? Function

Returns TRUE if **x** is greater than or equal to **y**. Otherwise, this function returns FALSE.

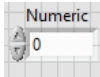
You can change the comparison mode of this function.

**Block Diagram➔R.C➔Comparation➔Greater or Equal?**




**Front Panel ➔R.C➔Boolean➔Vertical Toggle Switch**

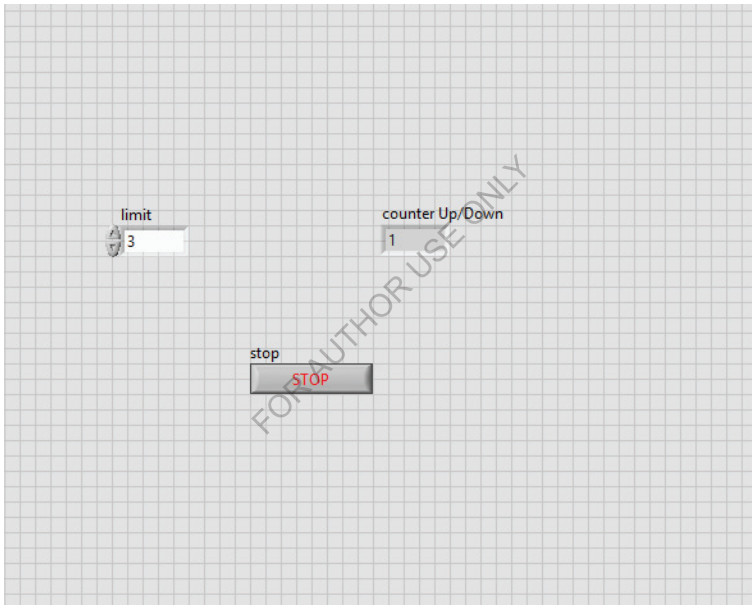

**Front Panel➔Numeric➔Numeric Indicator**

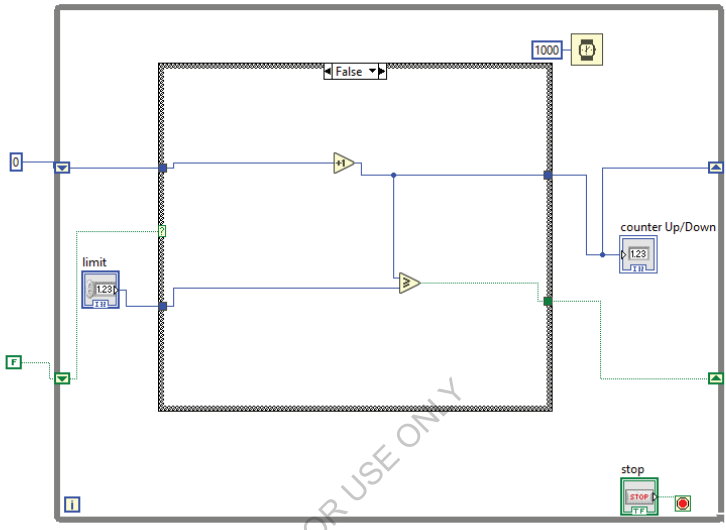



**Front Panel➔Numeric➔Numeric Control**

## Procedure

### Front Panel& Block Diagram






*The complete design*



## Discussion

1- Design the even counter by using LABVIEW grogram.

CHAPTER THREE
LABVIEW PROJECTS

# Timer

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW Timers and make the simple example to make a timer.

## Apparatus

LabVIEW software.

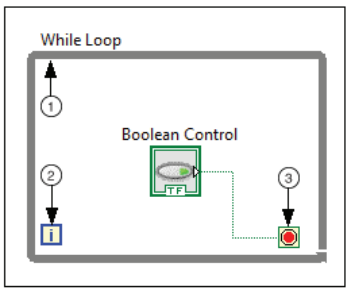
## The Experiment Blocks

**While Loop**



Repeats the code within its sub diagram until a specific condition occurs. A While Loop always executes at least one time.

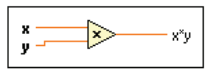
**Components of a While Loop**




**Sub diagram**—Contains code that the While Loop executes once per iteration.

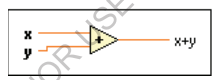
**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.




Returns the product of the inputs.

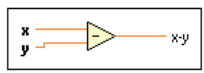
**Block Diagram➔R.C➔Numeric➔Multiply**




Computes the sum of the inputs.

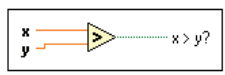
**Block Diagram➔R.C➔Numeric➔Add**






## Subtract Function

Computes the difference of the inputs.



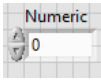

## Greater? Function

Returns TRUE if **x** is greater than **y**. Otherwise, this function returns FALSE. You can change the comparison mode of this function.

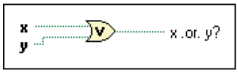
# CHAPTER THREE
# LABVIEW PROJECTS

**Front Panel ➔R.C➔Boolean➔Push Button**


**Front Panel➔Numeric➔Numeric Indicator**

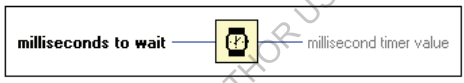



**Front Panel➔Numeric➔Numeric Control**




## Or Function

The connector pane displays the default data types for this polymorphic function.



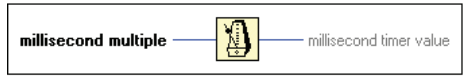

## Wait (ms) Function

Waits the specified number of milliseconds and returns the value of the millisecond timer. Wiring a value of 0 to the milliseconds to wait input forces the current thread to yield control of the CPU.

This function makes asynchronous system calls, but the nodes themselves function synchronously. Therefore, it does not complete execution until the specified time has elapsed.

**milliseconds to wait** specifies how many milliseconds to wait. This function does not wait for longer than 0x7fffff or 2,147,483,647 ms. To wait for a longer period, execute the function twice. Wiring a value of 0 to this parameter forces the current thread to yield control of the CPU.

**millisecond timer value** returns the value of the millisecond timer after the wait.




## Wait Until Next ms Multiple Function

Waits until the value of the millisecond timer becomes a multiple of the specified millisecond multiple. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate. However, it is possible that the first loop period might be short. Wiring a value of 0 to the milliseconds multiple input forces the current thread to yield control of the CPU.

This function makes asynchronous system calls, but the nodes themselves function synchronously. Therefore, it does not complete execution until the specified time has elapsed.

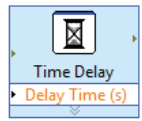
**millisecond multiple** is the input that specifies how many milliseconds lapse when the VI runs. Wiring a value of 0 to this parameter forces the current thread to yield control of the CPU.

**millisecond timer value** returns the value of the millisecond timer after the wait.



### Wait Until Next ms Multiple Details

When LabVIEW calls a VI for example, if **millisecond multiple** is 10 ms and **millisecond timer value** is 112 ms, the VI waits 8 more milliseconds until the millisecond timer value is 120 ms, a multiple of 10.




## Time Delay

Inserts a time delay into the calling VI.

### Dialog Box Options

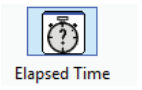
| Parameter | Description |
|---|---|
| **Time delay (seconds)** | Specifies how many seconds to delay running the calling VI. The default is 1.000. |

### Block Diagram Inputs

| Parameter | Description |
|---|---|
| **Delay Time (s)** | Specifies how many seconds to wait. The value you wire to this input overrides the value you set in the configuration dialog box. |
| **error in** | Describes error conditions that occur before this node runs. |

### Block Diagram Outputs

| Parameter | Description |
|---|---|
| **error out** | Contains error information. This output provides standard error out functionality. |




## Elapsed Time



Indicates the amount of time that has elapsed since the specified start time.

### Dialog Box Options

| Parameter | Description |
|---|---|
| **Elapsed time (seconds)** | Specifies how much time must elapse before the **Time has Elapsed** Boolean is set to TRUE. The default is 1.000. |
| **Automatically reset after time target** | Resets the elapsed time marker. |

### Block Diagram Inputs

| Parameter | Description |
|---|---|
| **Set Start Time (s)** | Uses the current time or a time offset from 12:00 a.m., Friday, January 1, 1904 [01-01-1904 00:00:00] as the start time instead of the time this VI first ran. If the value is 0, the Express VI uses the current time as the start time. The Express VI uses any other value as the number of seconds offset from 12:00 a.m., Friday, January 1, 1904 [01-01-1904 00:00:00]. For example if the value is 1, the Express VI measures the elapsed time since 12:00:01 a.m., Friday, January 1, 1904 [01-01-1904 00:00:01]. You can wire this input to the **Present (s)** output of another Elapsed Time Express VI to get this value. |
| **Time Target (s)** | Specifies how much time must elapse before the **Time has Elapsed** Boolean is set to TRUE. The default is 1. |
| **Reset** | Controls the initialization of the internal state of the VI. The default is FALSE. |
| **Auto Reset** | Resets the start time to the value in **Present (s)** when the Express VI reaches the **Time Target (s)**. |
| **error in** | Describes error conditions that occur before this node runs. |

### Block Diagram Outputs

| Parameter | Description |
|---|---|
| **Present (s)** | Displays the present time in seconds since 12:00 a.m., Friday, January 1, 1904, Universal Time [01-01-1904 00:00:00]. |
| **Present Text** | Displays the present date and time in text format. |
| **Elapsed Time (s)** | Displays the amount of time in seconds that has elapsed since the start time and the **Present (s)** time. |
| **Time has Elapsed** | Indicates if the **Elapsed Time (s)** is greater than the start time plus **Time Target (s)**. |
| **Elapsed Time Text** | Displays the amount of time in seconds that has elapsed since the start time. |
| **error out** | Contains error information. This output provides standard error out functionality. |
| **Start Time Text** | If **Automatically reset after time target** is FALSE, displays (in text format) the date and time the VI first ran or the time you wire to the **Set Start Time (s)** input. Otherwise displays the date and time of the last reset. |
| **Get Start Time (s)** | If **Automatically reset after time target** is FALSE, displays the date and time the VI first ran or the time you wire to the **Set Start Time (s)** input. Otherwise displays the date and time of the last reset. |

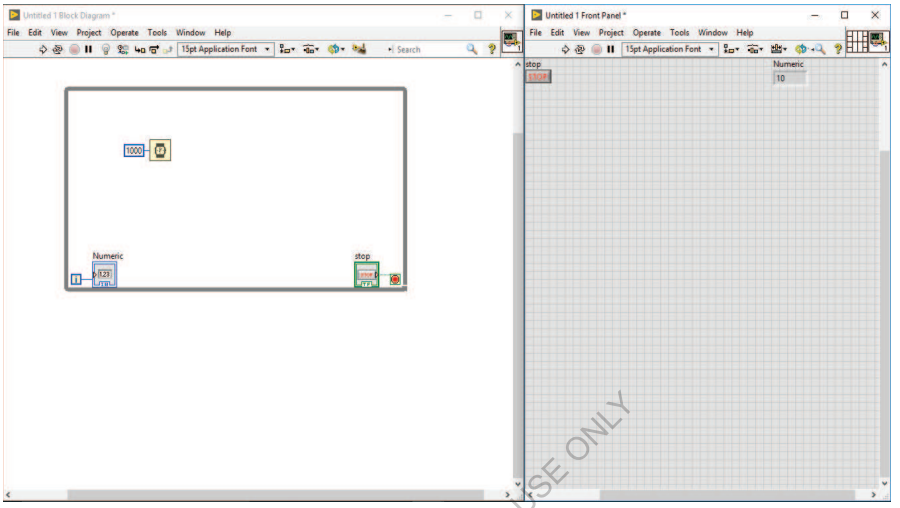

## Tick Count (ms) Function

Returns the value of the millisecond timer

## Procedure

### Wait (ms) Function

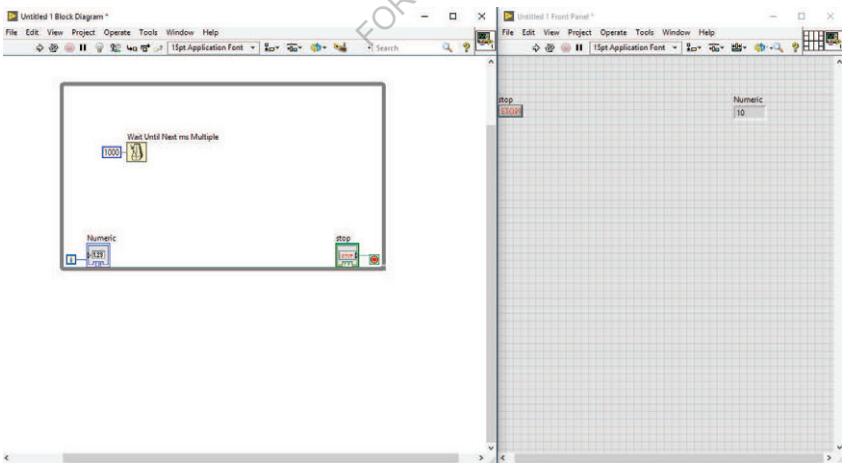
### Front Panel& Block Diagram



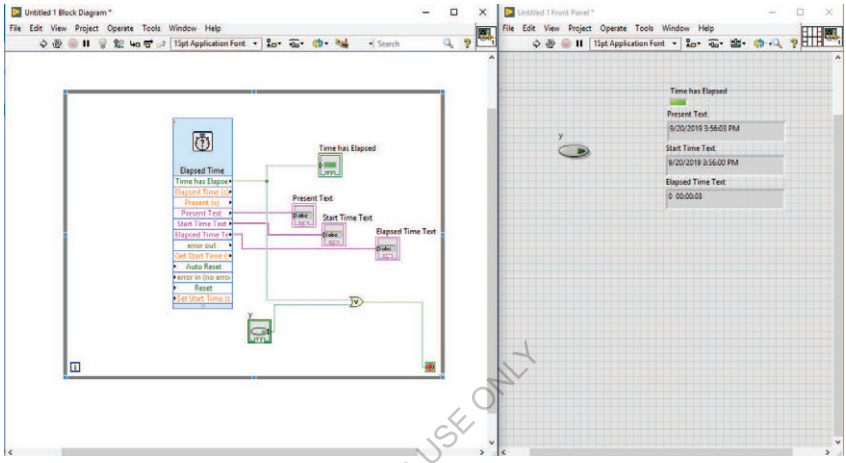

*The complete design*



## Wait Until Next ms Multiple Function



## Elapsed Time




## Timer Design

Design the timer that count from the start input number until down to zero.

## Discussion

1- Design the timer by using LABVIEW program to count from zero to infinity.

CHAPTER THREE
LABVIEW PROJECTS

# Animate Application

## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW to design animate application.

## Apparatus

LabVIEW software.

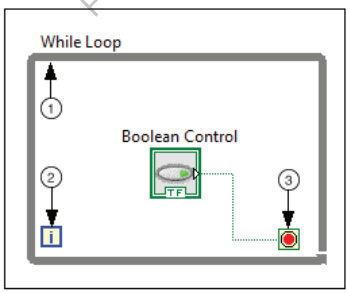
## The Experiment Blocks

### While Loop



Repeats the code within its subdiagram until a specific condition occurs. A While Loop always executes at least one time.

**Components of a While Loop**

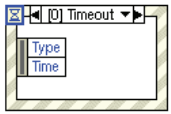



**Subdiagram**—Contains code that the While Loop executes once per iteration.

**Iteration Terminal (i)**—Provides the current loop iteration count. The loop count always starts at zero for the first iteration. If the iteration count exceeds 2,147,483,647, or 231-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to

keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

**Conditional Terminal**—Evaluates a Boolean input value to determine whether to continue executing the While Loop.

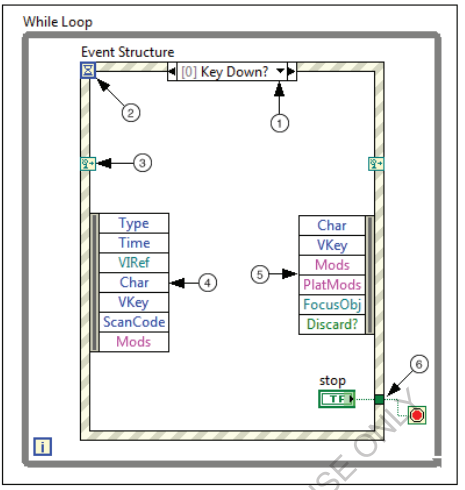



Waits until an event occurs, then executes the appropriate case to handle that event. The Event structure has one or more sub diagrams, or event cases, exactly one of which executes when the structure executes to handle an event. This structure can time out while waiting for notification of an event. Wire a value to the Timeout terminal at the top left of the Event structure to specify the number of milliseconds the Event structure waits for an event. The default is -1, which indicates never to time out.



### Event Structure Components

The following example shows an Event structure with the Key Down? event case.






1- The event selector label specifies which events cause the currently displayed case to execute. To view other event cases, click the down arrow next to the case name.

2- The Timeout terminal specifies the number of milliseconds to wait for an event before timing out. If you wire a value to the Timeout terminal, you must provide a Timeout event case to avoid an error.

3- The dynamic event terminals accept an event registration refnum or a cluster of event registration refnums for dynamic event registration. If you wire the inside right terminal, that terminal no longer carries the same data as the left terminal. You can wire the event registration refnum or cluster of event registration refnums to the inside right terminal through a Register For Events function and modify the event dynamically. Depending on the palette from which you select the Event structure, the dynamic event terminals might not appear by default. To display these terminals, right-click the Event structure and select **Show Dynamic Event Terminals** from the shortcut menu.

4- The Event Data Node identifies the data LabVIEW returns when an event occurs. Like the Unbundle By Name function, you can resize the node vertically and select the items you need. Use the Event Data Node to access event data elements, such as **Type** and **Time**, which are common to all events. Other event data elements, like **Char** and **VKey** for example, vary based on the event you configure.

5- The Event Filter Node identifies the event data you can modify before the user interface can process that data. This node appears in Event structure cases that handle filter events. If you want to change event data, you can wire and modify data items from the Event Data Node to the Event Filter Node. You also can change the event data by wiring new values to the node terminals. To completely discard an event, wire a TRUE value to the **Discard?** terminal. If you do not wire a value to a data item of the Event Filter Node, that data item remains unchanged.

6- Like a Case structure, the Event structure supports tunnels. However, by default you do not have to wire Event structure output tunnels in every case. All unwired tunnels use the default value for the tunnel data type. Right-click a tunnel and deselect **Use Default If Unwired** from the shortcut menu to revert to the default Case structure behavior where tunnels must be wired in all cases. You also can configure the tunnels to wire the input and output tunnels automatically in unwired cases.

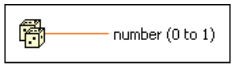


**Block Diagram➔R.C➔Structure➔Event Structure**


**For Loop**

Executes its subdiagram n times, where *n* is the value wired to the count (**N**) terminal. The iteration (**i**) terminal provides the current loop iteration count, which ranges from 0 to *n*-1.

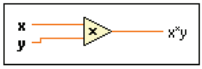
**Block Diagram➔R.C➔Structure➔For loop**




## Random Number

Produces a double-precision, floating-point number between 0 and 1. The number generated is greater than or equal to 0, but less than 1. The distribution is uniform.

**Block Diagram➔R.C➔Numeric➔Random Number (0-1)**




Returns the product of the inputs.

**Block Diagram➔R.C➔Numeric➔Multiply**



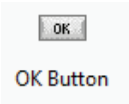

## Round To Nearest Function

Rounds the input to the nearest integer. If the value of the input is midway between two integers, the function returns the nearest even integer.

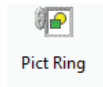
For example, if number is 1.5 or 2.5, **nearest integer** value is 2.


## To Unsigned Word Integer Function

Converts a number to a 16-bit unsigned integer in the range 0 to 65,535.



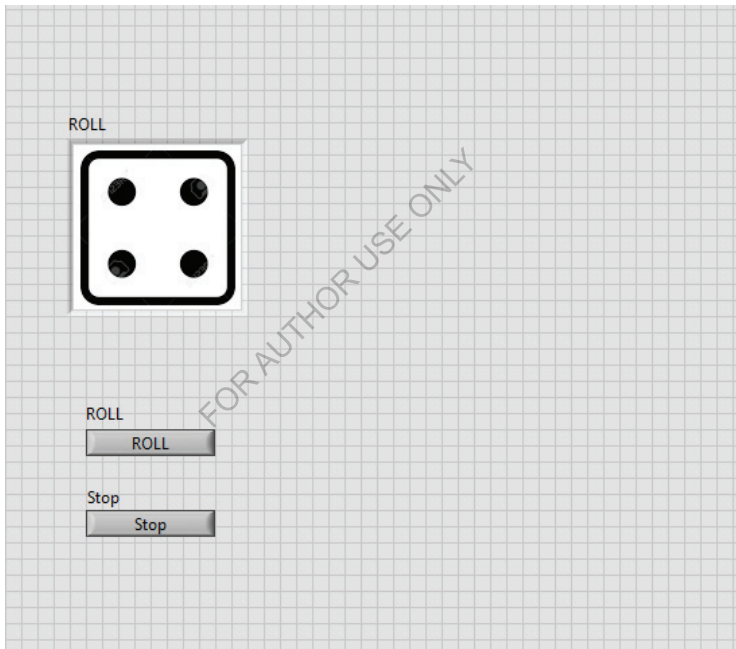

**Front Panel ➔R.C➔Boolean➔OK Button**

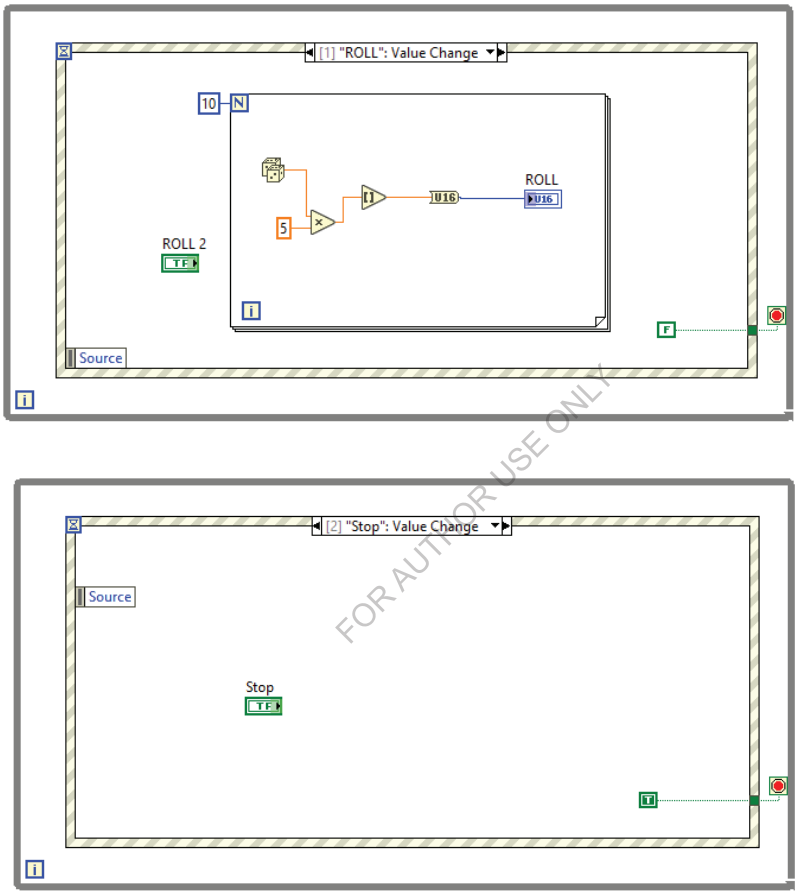



**Front Panel➔Ring & Enum➔Pict Ring**

## Procedure

### Front Panel& Block Diagram








***The complete design***

# CHAPTER THREE
# LABVIEW PROJECTS

## Discussion

1- Design the program by using LABVIEW to make the animate fan speed control.

CHAPTER THREE
LABVIEW PROJECTS

# Seven Segments

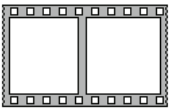
## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW design 7segments.

## Apparatus

LabVIEW software.

## The Experiment Blocks



**Flat Sequence Structure**

Consists of one or more sub diagrams, or frames, that execute sequentially. Use the Flat Sequence structure to ensure that a sub diagram executes before or after another sub diagram.

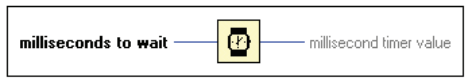
Data flow for the Flat Sequence structure differs from data flow for other structures. Frames in a Flat Sequence structure execute from left to right and when all data values wired to a frame are available. The data leaves each frame as the frame finishes executing. This means the input of one frame can depend on the output of another frame.

**Block Diagram➔R.C➔Structure➔Flat Sequence**


**Local Variable**

Use local variables to read or write to one of the controls or indicators on the front panel of a VI.

**Block Diagram➔R.C➔Structure➔Case Structure➔Local Variable**




## Wait (ms) Function

Waits the specified number of milliseconds and returns the value of the millisecond timer. Wiring a value of 0 to the milliseconds to wait input forces the current thread to yield control of the CPU.

This function makes asynchronous system calls, but the nodes themselves function synchronously. Therefore, it does not complete execution until the specified time has elapsed.

**milliseconds to wait** specifies how many milliseconds to wait. This function does not wait for longer than 0x7fffffff or 2,147,483,647 ms. To wait for a longer period, execute the function twice. Wiring a value of 0 to this parameter forces the current thread to yield control of the CPU.

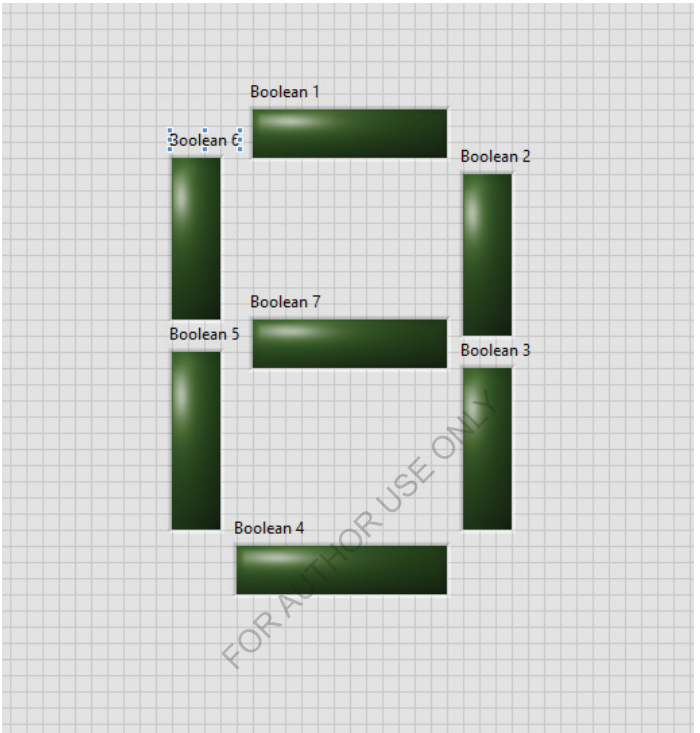
**millisecond timer value** returns the value of the millisecond timer after the wait.




**Front Panel➔R.C➔Boolean➔Square Led**

## Procedure

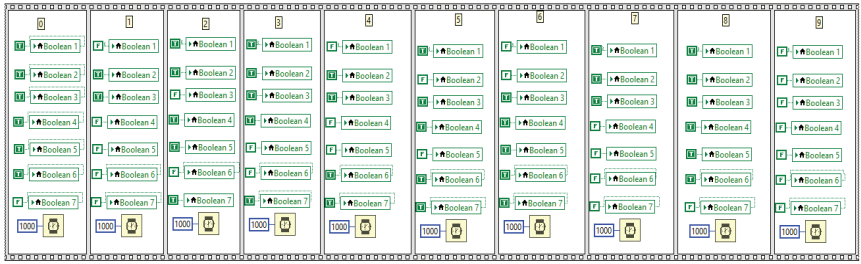
### Front Panel& Block Diagram

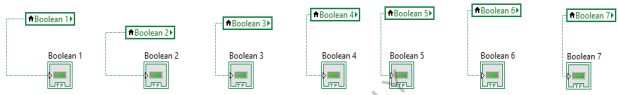





***The complete design***

# CHAPTER THREE
# LABVIEW PROJECTS




## Discussion

1- Design the 7segments that display the random numbers.

2- Design the counter and display the numbers by two 7segments.

CHAPTER THREE
LABVIEW PROJECTS

# Read and Write Spread Sheet

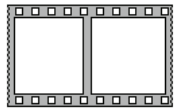
## Aim of Experiment

The objective of this exercise to learn how to use LabVIEW to read spread sheet.

## Apparatus

LabVIEW software.

## The Experiment Blocks



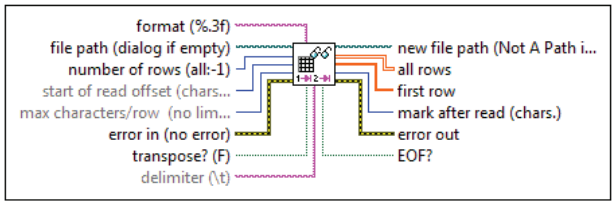
**Flat Sequence Structure**



Consists of one or more sub diagrams, or frames, that execute sequentially. Use the Flat Sequence structure to ensure that a sub diagram executes before or after another sub diagram.

Data flow for the Flat Sequence structure differs from data flow for other structures. Frames in a Flat Sequence structure execute from left to right and when all data values wired to a frame are available. The data leaves each frame as the frame finishes executing. This means the input of one frame can depend on the output of another frame.

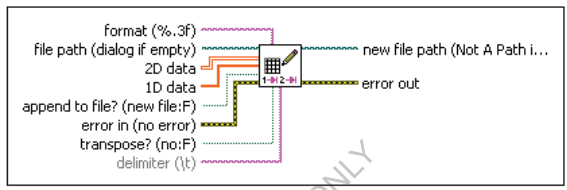
**Block Diagram➔R.C➔Structure➔Flat Sequence**



## Read Delimited Spreadsheet VI

Reads a specified number of lines or rows from a numeric text file beginning at a specified character offset and converts the data to a 2D, double-precision array of numbers, strings, or integers.

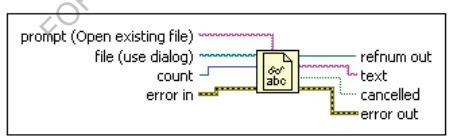
**Block Diagram➔R.C➔FileI/O➔ Read Delimited Spreadsheet VI**




## Write Delimited Spreadsheet VI

Converts a 2D or 1D array of strings, signed integers, or double-precision numbers to a text string and writes the string to a new byte stream file or appends the string to an existing file.



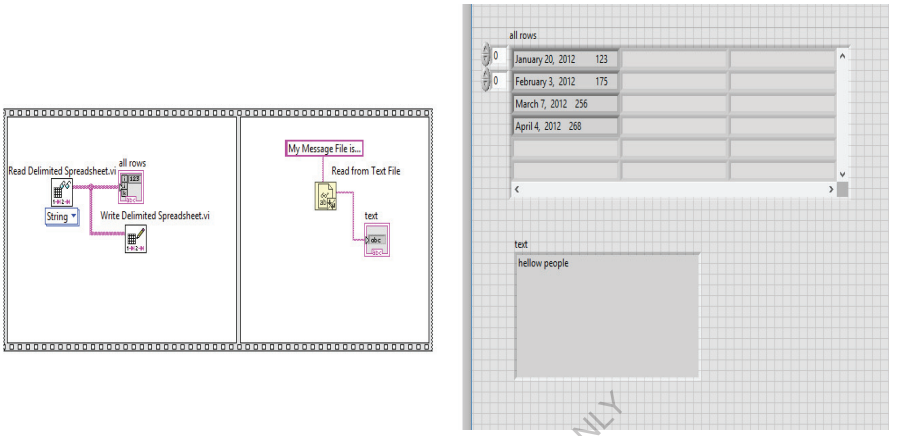



## Read from Text File Function

Reads a specified number of characters or lines from a byte stream file. This function does not work for files inside an LLB.

## Procedure

### Front Panel& Block Diagram




***The complete design***

## Discussion

1- Design the program by using LABVIEW to read values from excel sheet.

# CHAPTER THREE
# LABVIEW PROJECTS

## Reference


1- Ertugrul, Nesimi. *LabVIEW for electric circuits, machines, drives, and laboratories*. Prentice Hall PTR, 2002.

2- Conway, Jon, and Steve Watts. *Software Engineering with LabVIEW*. Pearson Education, 2003.

3- Jerome, Jovitha. *Virtual instrumentation using LabVIEW*. PHI Learning Pvt. Ltd., 2010.

4- Travis, Jeffrey. *LabVIEW for everyone*. Pearson Education India, 2002.

5- Kehtarnavaz, Nasser, and Namjin Kim. *Digital signal processing system-level design using LabVIEW*. Elsevier, 2011.

6- Clark, Cory L. *LabVIEW digital signal processing*. Tata McGraw-Hill Education, 2005.

7- Collamati, L., et al. "Induction machine stator fault on-line diagnosis based on LabVIEW environment." *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)*. Vol. 1. IEEE, 1996.

8- Bell, Ian, et al. "Integration of hardware into the LabVIEW environment for rapid prototyping and the development of control design applications." *Proc. of the UKACC Control 2004 Mini Symposia*. 2004.

9- Spanik, Pavol, et al. "Application of virtual instrumentation LabVIEW for power electronic system analysis." *2006 12th International Power Electronics and Motion Control Conference*. IEEE, 2006.

10- Odon, A., P. Otomański, and M. Płaczek. "LabView application to simulation of digital to analog converter based on binary weighted resistors ladder." *Poznan University of Technology Academic Journals. Electrical Engineering* 61 (2010): 139-145.