

# **MODULE – II**

## **INTRODUCTION TO LABVIEW**

### **REPETITION AND LOOPS**

#### **INTRODUCTION**

- LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical programming environment which has become prevalent throughout research labs, academia and industry.
- It is a powerful and versatile analysis and instrumentation software system for measurement and automation.
- Its graphical programming language called G programming is performed using a graphical block diagram that compiles into machine code and eliminates a lot of the syntactical details.
- LabVIEW offers more flexibility than standard laboratory instruments because it is software based.
- Using LabVIEW, the user can originate exactly the type of virtual instrument needed and programmers can easily view and modify data or control inputs.
- LabVIEW programs are called virtual instruments (VIs), because their appearance and operation imitate physical instruments like oscilloscopes. LabVIEW is designed to facilitate data collection and analysis, as well as offers numerous display options.
- LabVIEW contains a comprehensive set of VIs and functions for acquiring, analyzing, displaying, and storing data, as well as tools to help you troubleshoot your code
- All test, measurement and control applications can be divided into three main components and the key to virtual instrumentation is the ability to acquire, analyze and present data.
- LabVIEW can acquire data using the devices like GPIB, Serial, Ethernet, VXI, PXI Instruments, Data Acquisition (DAQ), PCI extensions for Instrumentation (PXI), Image Acquisition (IMAQ), Motion Control, Real-Time (RT) PXI, PLC (through OPC Server), PDA, and Modular Instruments.
- To help you analyze your data LabVIEW includes analysis functions for Differential Equations, Optimization, Curve Fitting, Calculus, Linear Algebra, Statistics and so on.
- Express VIs is specifically designed for measurement analysis, including filtering and spectral analysis. Signal Processing VIs for Filtering, Windowing, Transforms, Peak Detection, Harmonic Analysis, and Spectrum Analysis are provided.
- LabVIEW includes the following tools to help in presenting data on the computer; Graphs, Charts, Tables, Gauges, Meters, Tanks, 3D Controls, Picture Control, 3D Graphs and Report Generation.
- LabVIEW can communicate with hardware such as data acquisition, vision, and motion control devices, and GPIB, PXI, VXI, RS-232, and RS-485 devices.
- LabVIEW also has built-in features for connecting your application to the Web using the LabVIEW Web Server and software standards such as TCP/IP networking and ActiveX.
- The LabVIEW Family consists of NI LabVIEW Graphical Programming Software for Measurement and Automation, LabVIEW Real-Time Module, LabVIEW FPGA Module, LabVIEW PDA Module, LabVIEW Datalogging and Supervisory Control Module.

## ADVANTAGES OF LABVIEW

- **Graphical user interface:**

Design professionals use the drag-and-drop user interface library by interactively customizing the hundreds of built-in user objects on the control's palette.

- **Drag-and-drop built-in functions:**

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

- **Modular design and hierarchical design:**

Run modular LabVIEW VIs by themselves or as subVIs and easily scale and modularize programs depending on the application.

- **Multiple high level development tools:**

Develop faster with application specific development tools, including the LabVIEW State chart Module, LabVIEW Control Design and Simulation Module and LabVIEW FPGA Module.

- **Professional Development Tools:**

Manage large, professional applications and tightly integrated project management tools; integrated graphical debugging tools; and standardized source code control integration.

- **Multi-Platforms:**

The majority of computer systems use the Microsoft Windows operating system. LabVIEW works on other platforms like Mac OS, Sun Solaris and Linux. LabVIEW applications are portable across platforms.

- **Reduces Cost and Preserves Investment:**

A single computer equipped with LabVIEW is used for countless applications and purposes—it is a versatile product. Complete instrumentation libraries can be created for less than the cost of a single traditional, commercial instrument.

- **Flexibility and Scalability:**

Engineers and scientists have needs and requirements that can change rapidly. They also need to have maintainable, extensible solutions that can be used for a long time. By creating virtual instruments based on powerful development software such as LabVIEW, you inherently design an open framework that seamlessly integrates software and hardware. This ensures that your applications not

- **Connectivity and Instrument Control:**

LabVIEW has ready-to-use libraries for integrating stand-alone instruments, data acquisition devices, motion control and vision products, GPIB/ IEEE 488 and serial/RS-232 devices, and PLCs to build a complete measurement and automation solution. Plug-and Play instrument drivers access the industry's largest source of instrument drivers with several instruments from various vendors.

- **Open Environment:**

LabVIEW provides the tools required for most applications and is also an open development environment. This open language takes advantage of existing code; can easily integrate with legacy

systems and incorporate third party software with .NET, ActiveX, DLLs, objects, TCP, Web services and XML data formats.

- **Distributed Development:**

Can easily develop distributed applications with LabVIEW, even across different platforms. With powerful server technology you can offload processor intensive routines to other machines for faster execution, or create remote monitoring and control applications.

- **Visualization Capabilities:**

LabVIEW includes a wide array of built-in visualization tools to present data on the user interface of the virtual instrument as chart, graphs, 2D and 3D visualization. Reconfiguring attributes of the data presentation, such as colors, font size, graph types, and more can be easily performed.

- **Rapid Development with Express Technology:**

Use configuration-based Express Vis and I/O assistants to rapidly create common measurement applications without programming by using LabVIEW signal Express.

- **Compiled Language for Fast Execution:**

LabVIEW is a compiled language that generates optimized code with execution speeds comparable to compiled C and develops high-performance code.

- **Simple application distribution:**

Use the LabVIEW application builder to create executables(exes) and shared libraries (DLLs) for deployment.

- **Target management:**

Easily manage multiple targets, from real-time to embedded devices including FPGAs, microprocessors, microcontrollers, PDAs and touch panels.

- **Object–Oriented Design:**

Use object-oriented programming structures to take advantage of encapsulation and inheritance to create modular and extensible code.

- **Algorithm design:**

Develop algorithms using math-oriented textual programming and interactively debug .m file script syntax with LabVIEW Math Script.

## SOFTWARE ENVIRONMENT

- There are three steps to create our application in the software environment:

- Design a user interface
- Draw our graphical code
- Run our program

- A virtual instrument (VI) has three main components

- The front panel,
- The block diagrams
- The icon/connector pane.

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

### **FRONT PANEL WINDOWS**

- One of the most powerful features that LabVIEW offers engineers and scientists is its graphical programming environment to design custom virtual instruments by creating a graphical user interface on the computer screen to
  - Operate the instrumentation program
  - Control selected hardware
  - Analyze acquired data
  - Display results
- The front panel can include knobs, push buttons, graphs and various other controls (which are user inputs) and indicators (which are program outputs).
- Controls are inputs used to simulate instrument input devices and supply data to the block diagram of the VI, and indicators are outputs displays used to simulate instrument output devices and display data the block diagram acquires or generates.
- The front panel is customized to emulate control panels of traditional instruments, create custom test panels, or visually represent the control and operation of processes.

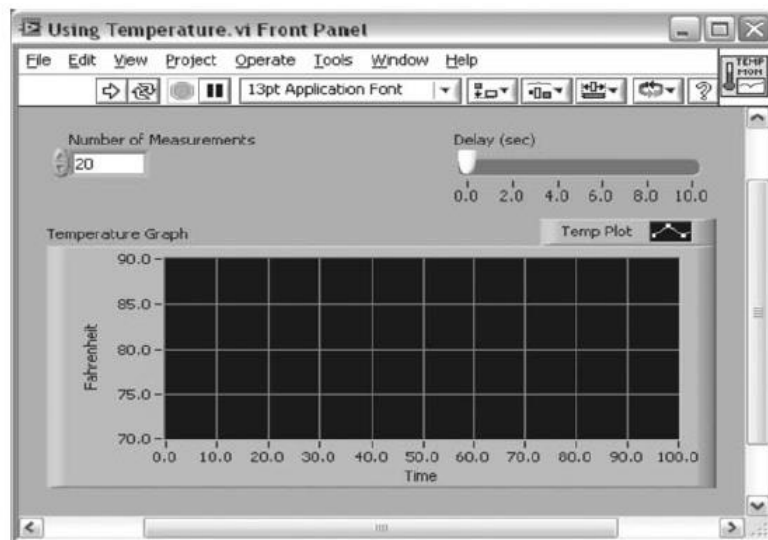


Figure 2.1 LabVIEW virtual instrument front panel.

## BLOCK DIAGRAM WINDOWS

- The block diagrams accompany the program for the front panel.
- Front panel objects appear as terminals on the block diagram and the components wired together.
- After the front panel is built, codes are added using graphical representations of functions in the block diagram to control the front panel objects.
- The block diagram contains the graphical source code composed of nodes, terminals, and wires.
- The components of a block diagram are lower-level VIs, built-in functions, constants and program execution control structures.
- Wires have to be drawn to connect the corresponding objects together to indicate the flow of data between each of them. Front panel objects have analogous terminals on the block diagram so that data can pass easily from the user to the program and back to the user.
- Use Express VIs, standard VIs and functions on the block diagram to create our measurement code. Block diagram objects include the terminals, sub- V is, functions, constants, structures and wires.
- LabVIEW is the easiest, most powerful tool for acquiring, analyzing and presenting real-world data. Terminals are entry and exit ports that exchange information between the panel and the diagram.

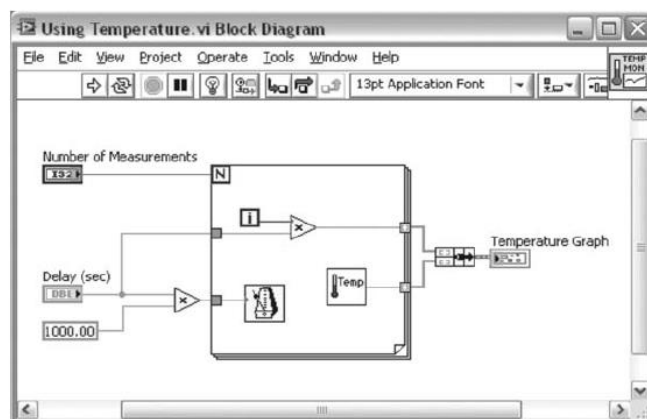
















Figure 2.2 LabVIEW virtual instrument block diagram.

## ICON/CONNECTOR PANE








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

## FRONT PANEL TOOLBAR

	<i>Run button:</i> To run a VI and it appears as a solid white arrow.
	While the VI runs, the <i>Run</i> button appears as shown.
	If the VI that is running is a subVI, the <i>Run</i> button appears as shown.
	<i>Broken Run</i> arrow means a nonexecutable VI and the VI you are creating or editing contains errors. Click this button to display the <i>Error list</i> window, which lists all errors and warnings.
	<i>Run Continuously</i> button: To run the VI until you abort or pause execution. Click the button again to disable continuous running.
	<i>Abort Execution</i> button: To stop the VI immediately if there is no other way to stop the VI.
	<i>Pause</i> button: To pause a running VI.
	<i>Text Settings:</i> To change the font settings including size, style, colour.
	<i>Align Objects:</i> To align objects along axes, including vertical, top edge, left.
	<i>Distribute Objects:</i> To space objects evenly, including gaps, compression.
	<i>Resize Objects:</i> To resize multiple front panel objects to the same size.
	<i>Reorder:</i> When you have objects that overlap each other and you want to define which one is in front or back of another, select one of the objects with the positioning tool and then select from <i>Move Forward</i> , <i>Move Backward</i> , <i>Move To Front</i> , and <i>Move To Back</i> .
	<i>Show Context Help Window</i> button: To toggle the display of the Context Help window.
	<i>Type:</i> Remind you that a new value is available to replace an old value. The <i>Enter</i> button disappears when you click it, press the <Enter> key or click the front panel or block diagram workspace.



## BLOCK DIAGRAM TOOLBAR












	<i>Highlight Execution</i> button: To display an animation of the block diagram execution when you click the <i>Run</i> button. See the flow of data through the block diagram. Click the button again to disable execution highlighting.
	<i>Retain Wire Values</i> : To save the wire values at each point in the flow of execution so that when you place a probe on the wire, you can immediately retain the most recent value of the data that passed through the wire. You must successfully run the VI at least once before you are able to retain the wire values.
	<i>Step Into</i> : To open a node and pause. When you click the <i>Step Into</i> button again, it executes the first action and pauses at the next action of the sub VI or structure. You also can press <Ctrl> and down arrow keys. Single-stepping through a VI steps through the VI node by node. Each node blinks to denote when it is ready to execute. By stepping into the node, you are ready to single-step inside the node.
	<i>Step Over</i> : To open a node and pause. When you click the <i>Step Over</i> button again, it executes the first action and pauses at the next action of the sub VI or structure. You also can press <Ctrl> and down arrow keys. Single-stepping through a VI steps through the VI node by node. Each node blinks to denote when it is ready to execute. By stepping into the node, you are ready to single-step inside the node.
	<i>Step Over</i> : To execute a node and pause at the next node. You also can press <Ctrl> and right arrow keys. By stepping over the node, you execute the node without single-stepping through the node.
	<i>Step Out</i> : To finish executing the current node and pause. When the VI finishes executing, the <b>Step Out</b> button becomes dimmed. You also can press <Ctrl> and up arrow keys. By stepping out of a node, you can complete single-stepping through the node and navigate to the next node.
	<i>Warning</i> : Appears if a VI includes a warning and you placed a checkmark in the <i>Show Warnings</i> checkbox in the <i>Error List</i> window. A warning indicates there is a potential problem with the block diagram, but it does not stop the VI from running.

## PALETTES

- LabVIEW has graphical, floating palettes help to create and run VIs.
- The Three Palettes are
  - The Tools Palette
  - The Controls Palette
  - The Function's Palette

## THE TOOLS PALETTE




- The Tools palette shown in Figure 2.4 is available on both the front panel and the block diagram. You can create, modify, and debug VIs using the tools located on the floating Tools palette.
- A tool is a special operating mode of the mouse cursor. The cursor corresponds to the icon of the tool selected in the Tools palette.
- Use the tools to operate and modify the front panel and block diagram objects.
- You can manually choose the tool you need by selecting it on the Tools palette. Or select View» Tools Palette to display the Tools palette.

	<i>Automatic Tool Selection button:</i> To enable automatic tool selection. If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the <i>Tools</i> palette.
	<i>Operating tool:</i> To change the values of a control or select the text within a control. This tool changes to the icon shown at right when it moves over a text control, such as a numeric or string control.
	<i>Positioning tool:</i> To select, move, or resize objects and it changes to resizing handles when it moves over the edge of a resizable object.
	<i>Labeling tool:</i> To edit text and create free labels. This tool changes to the icon on the right when you create free labels.
	<i>Wiring tool:</i> To wire objects together on the block diagram.
	<i>Object Shortcut Menu tool:</i> To access an object shortcut menu with the left mouse button.
	<i>Scrolling tool:</i> To scroll through windows without using scrollbars.
	<i>Breakpoint tool:</i> To set breakpoints on VIs, functions, nodes, wires, and structures to pause execution at that location.
	<i>Probe tool:</i> To create probes on wires on the block diagram. Use the <i>Probe</i> tool to check intermediate values in a VI that produces questionable or unexpected results.
	<i>Color Copy tool:</i> To copy colors for pasting with the <i>Coloring tool</i> .
	<i>Coloring tool:</i> To color an object. It also displays the current foreground and background color settings.

## FRONT PANEL - CONTROLS PALETTE

- The Controls palette is available only on the front panel.
- The Controls palette contains the controls and indicators which you can use to create the front panel.
- The Controls palette can be accessed from the front panel by selecting View» Controls Palette or by right-clicking an open space on the front panel window to display the Controls palette.
- The Controls palettes contain subpalettes of objects which you can use to create a VI. When you click a subpalette icon, the entire palette changes to the subpalette you selected.



	Use the <i>search</i> button on the <i>Controls and Functions palettes</i> to search for controls, VIs, and functions. In search mode, you can perform text-based searches.
	Use the <i>Options</i> button on the <i>Controls or Functions palette</i> toolbar to change to another palette view or format.
	<i>Up to Owning palette</i> —Navigates up one level in the palette hierarchy.

## BLOCK DIAGRAM—FUNCTIONS PALETTE

- Use the Functions palette to create the block diagram.
- The Functions palette contains the Sub-VIs, functions, and constants that are available only on the block diagram.
- You can access the Functions palette from the block diagram by selecting View» Functions Palette.
- You can also right-click an open space on the block diagram to display the Functions palette.
- The VIs and functions located on the Functions palette depend on the palette view currently selected.
- The VIs and functions are located on subpalettes based on the types of VIs and functions.

## FRONT PANEL CONTROLS AND INDICATORS

- You can build the front panel with controls and indicators, which are the interactive input and output terminals of the VI, respectively.
- Controls are knobs, push buttons, dials and other input devices.
- Indicators are graphs, LEDs and other displays.
- Controls simulate instrument input devices and supply data to the block diagram of the VI.
- Indicators simulate instrument output devices and display data the block diagram acquires or generates.
- Every control or indicator has a data type associated with it. They are numeric data type, Boolean data type and string data type as shown in Figure 2.11.

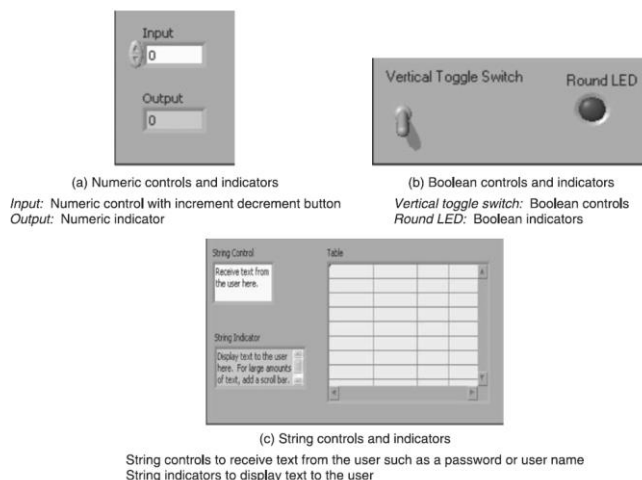


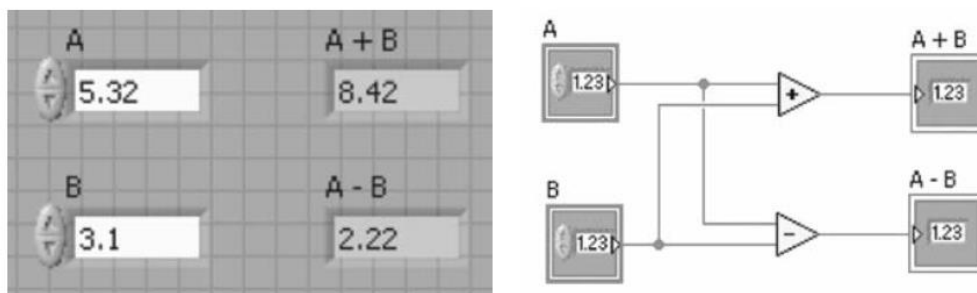
Figure 2.11 Numeric, Boolean and string data type.

- The numeric data type shown in Figure 2.11(a) can be of various types such as integer or real. The two most commonly used numeric objects are the numeric control and the numeric indicator.
- The Boolean data type represents data that only has two parts, such as TRUE and FALSE or ON and OFF as in Figure 2.11(b). Use Boolean controls and indicators to enter and display Boolean (True or False) values. Boolean objects simulate switches, push buttons, and LEDs.
- Figure 2.11(c) shows the string data type which is a sequence of ASCII characters. Use string controls to receive text from the user such as a password or user name and indicators to display text to the user.
- To keep an object proportional to its original size as you resize it, press the <Shift> key while you drag the resizing handles or circles.
- To resize an object as you place it on the front panel, press the <Ctrl> key while you click to place the object and drag the resizing handles or circles.

## BLOCK DIAGRAM

- Block diagram objects include terminals, Sub-VIs, functions, constants, structures and wires to transfer data among other block diagram objects.

### TERMINALS



**Figure 2.12** Front panel and its corresponding block diagram.

- Front panel objects appear as terminals on the block diagram. Figure 2.12 shows the front panel where two numeric values A and B need to be added and subtracted, and the block diagram for the corresponding front panel.
- Terminals are entry and exit ports that exchange information between the front panel and block diagram.
- Terminals are analogous to parameters and constants in text-based programming languages.
- Types of Terminals are
  - Control or Indicator Terminals
  - Node Terminals
- Control and indicator terminals belong to front panel controls and indicators. Data you enter into the front panel controls (A and B in the figure) enter the block diagram through the control terminals. The data then enter the Add and Subtract functions. When the Add and Subtract functions complete their calculations; they produce new data values. The data values flow to the indicator terminals, where they update the front panel indicators.
- The terminals represent the data type of the control or indicator.

- You can configure front panel controls or indicators to appear as icon or data type terminals on the block diagram. By default, front panel objects appear as icon terminals. For example, a numeric icon terminal, shown in Figure 2.13, represents a numeric control on the front panel.



Figure 2.13 Default icon terminals can be viewed as data type terminal.

- A DBL terminal represents a double-precision, floating-point numeric control.
- To display a terminal as a data type on the block diagram, right-click the terminal and select View As Icon from the shortcut menu.

## NODES

- Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when VI runs.
- They are analogous to statements, operators, functions, and subroutines in text-based programming languages.
- Nodes can be functions, Sub-VIs or structures.
- Structures are process control elements, such as Case structures, For loops, or While loops.

## FUNCTIONS

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

## SUB-VIs

- Sub-VIs are VIs that you build to use inside of another VI or that you access on the Functions palette.
- Any VI has the potential to be used as a Sub-VI. When you double-click a Sub-VI on the block diagram, its front panel and block diagram appear.
- The front panel includes controls and indicators. The block diagram includes wires, front panel icons, functions, possibly Sub-VIs and other LabVIEW objects.
- The upper-right corner of the front panel and block diagram displays the icon for the VI. This is the icon that appears when you place the VI on a block diagram as a Sub-VI.
- Sub-VIs also can be Express VIs.
- Express VIs are nodes that require minimal wiring because you configure them with dialog boxes. Use Express VIs for common measurement tasks. You can save the configuration of an Express VI as a Sub-VI.








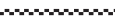







## EXPRESS VIs and VIs

- Express VIs are interactive VIs with configurable dialog page but Standard VIs are modularized VIs customized by wiring.
- LabVIEW uses colored icons to distinguish between Express VIs, VIs and functions on the block diagram.
- By default, icons for Express VIs appear on the block diagram as expandable nodes with icons surrounded by a blue field.
- Icons for VIs have white backgrounds, and icons for functions have pale yellow backgrounds.
- By default, most functions and VIs on the block diagram appear as icons that are not expandable, unlike Express VIs.

## WIRES

- You can transfer data among block diagram objects through wires.
- Each wire has a single data source, but you can wire it to many VIs and functions that read the data.
- Wires are different colors, styles and thicknesses, depending on their data types as shown in Table 2.5.

TABLE 2.5 Common wire types

Data type	Scalar	1D Array	2D Array
DBL Numeric (Floating Point)			
Integer Numeric			
Boolean			
String			
Dynamic			

- A broken wire appears as a dashed black line with a red X in the middle, as shown at left. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types.
- You must connect the wires to inputs and outputs that are compatible with the data that is transferred with the wire. You cannot wire an array output to a numeric input. In addition, the direction of the wires must be correct. You must connect the wires to only one input and at least one output. Also cannot wire two indicators together.
- The components that determine wiring compatibility include the data type of the control and/or the indicator and the data type of the terminal.
- LabVIEW automatically wires objects as you place them on the block diagram.
- LabVIEW connects the terminals that best match and leave terminals that do not match unconnected. As you move a selected object close to other objects on the block diagram, LabVIEW draws temporary wires to show you valid connections. When you release the mouse button to place the object on the block diagram, LabVIEW automatically connects the wires.
- Toggle automatic wiring by pressing the spacebar while you move an object using the Positioning tool. You can adjust the automatic wiring settings by selecting Tools» Options and selecting Block Diagram from the top pull-down menu

## DATA FLOW PROGRAM

- LabVIEW follows a dataflow model for running VIs.
- A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path.
- Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution.
- In control flow, the sequential order of program elements determines the execution order of a program.
- For a data flow programming, consider a block diagram shown in Figure 2.14 that adds two numbers and then subtracts 50.00 from the result of the addition.

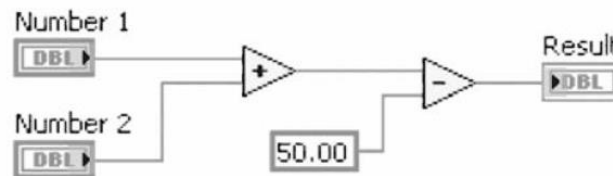


Figure 2.14 Data flow program.

- In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because the Subtract function cannot execute until the Add function finishes executing and passes the data to the Subtract function. A node executes only when data are available at all of its input terminals, and it supplies data to its output terminals only when it finishes execution.

## FOR LOOPS

- A For Loop executes a sub diagram, a set number of times.
- Figure 4.1(a) shows a For Loop in LabVIEW and Figure 4.1(b) shows the flow chart equivalent of the For Loop functionality.

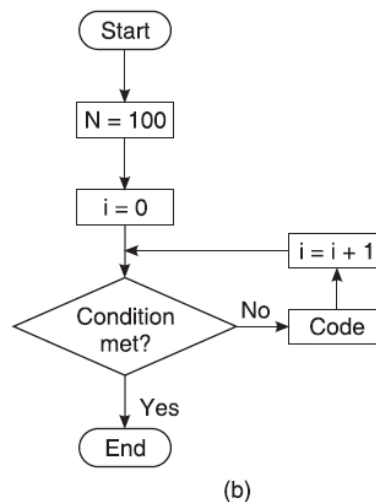
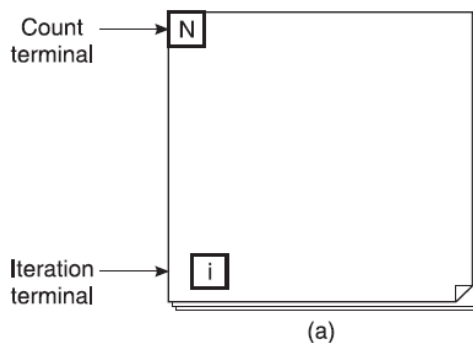
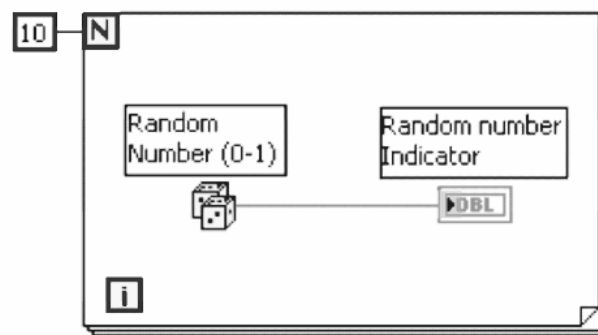


Figure 4.1 (a) For Loop in LabVIEW and (b) Flow chart equivalent to the For Loop.

- The For Loop is located on the Functions>>Programming>>Structures Palette.
- Select the For Loop from the palette and use the cursor to drag a selection rectangle to create a new For Loop or around the section of the block diagram you want to repeat. You also can place a While Loop on the block diagram, right-click the border of the While Loop, and select Replace with For Loop from the shortcut menu to change a While Loop to a For Loop.
- **N**: The value in the count terminal 'N' (an input terminal) indicates how many times to repeat the sub diagram. Set the count explicitly by wiring a value from outside the loop to the left or top side of the count terminal, or set the count implicitly with auto-indexing.
- **I**: The iteration terminal 'i' (an output terminal) contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0. Figure 4.2 shows a simple For Loop which generates 10 random numbers and displays in the Random Number Indicator.

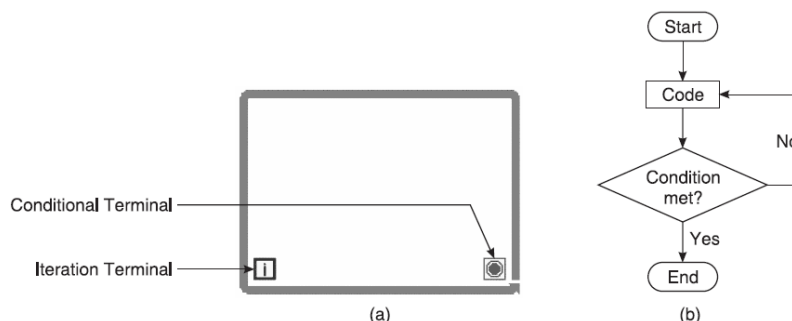


**Figure 4.2** Generating random numbers using For Loop.

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

## WHILE LOOPS

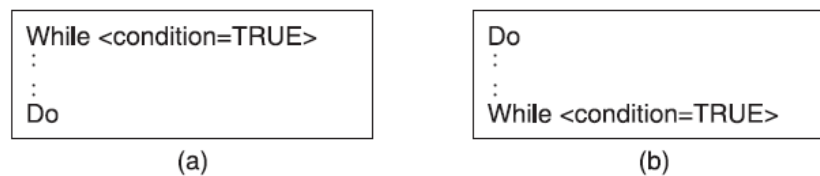
- A While Loop executes a sub diagram until a condition is met.
- The While Loop is similar to a Do Loop or a Repeat-Until Loop in text-based programming languages.
- Figure 4.4(a) shows a While Loop in LabVIEW and 4.4(b) is the flow chart equivalent of the While Loop.



**Figure 4.4** (a) While Loop in LabVIEW and (b) Flow chart equivalent to the While Loop.

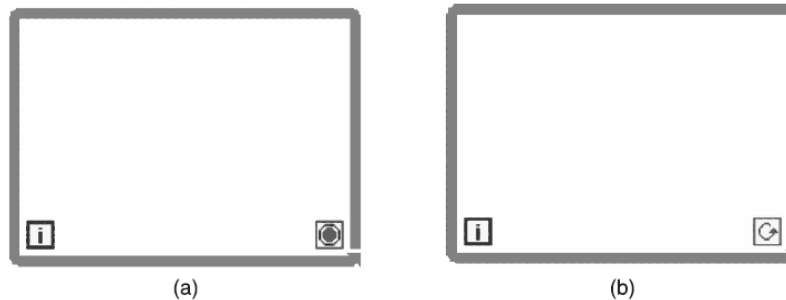


- The While Loop always executes at least once.
- The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing the sub diagram, only if the expected value at the conditional terminal exists.
- In LabVIEW, the WHILE Loop is located on the Functions>>Programming>>Structures palette. You also can place a For Loop on the block diagram, right-click the border of the For Loop, and select Replace with While Loop from the shortcut menu to change a For Loop to a While Loop.
- The While Loop contains two terminals,
  - Conditional Terminal
  - Iteration Terminal
- The Conditional Terminal is used to control the execution of the loop, whereas the Iteration Terminal is used to know the number of completed iterations.
- Conventional programming languages support two types of WHILE constructs as illustrated in Figure 4.5. These are
  - Pre- Post-Test Modes
  - Post-Test Modes.



**Figure 4.5** (a) Pre-test mode and (b) Post-test modes of a While loop.

- In the pre-test mode the condition is tested prior to the execution of every iteration and if the result is false, then the execution of the loop is aborted.
- In the post-test mode the test is carried out only at the end of the loop.
- Functionally, the major difference is that under the post-test mode even if the condition is false at the first execution or first iteration, the loop will be executed at least once, since the test is only performed at the end of the loop.
- LabVIEW supports only the post-test form of the While construct
- •-The While Loop executes the sub diagram until the conditional terminal, and receives a specific Boolean value. The default behavior and appearance of the conditional terminal is Stop if True as shown in Figure 4.6(a).

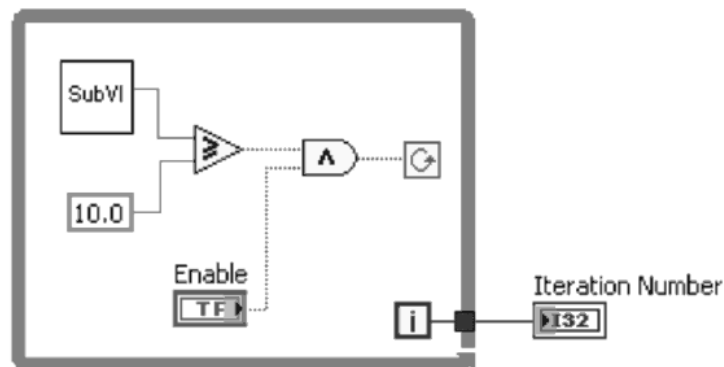


**Figure 4.6** (a) While Loop with conditional terminal as *Stop if True* and (b) While Loop with conditional terminal as *Continue if True*.

- When a conditional terminal is Stop if True, the While Loop executes its sub diagram until the conditional terminal receives a TRUE value. You can change the behavior and appearance of the conditional terminal by right-clicking the terminal or the border of the While Loop and selecting Continue if True from the shortcut menu as shown in Figure 4.6(b).
- When a conditional terminal is Continue if True, the While Loop executes its sub diagram until the conditional terminal receives a FALSE value. You also can use the Operating Tool to click the conditional terminal to change the condition. The VI shows error if the conditional terminal is unwired.
- You also can perform basic error handling using the conditional terminal of a While Loop. When you wire an error cluster to the conditional terminal, only the TRUE or FALSE value of the status parameter of the error cluster passes to the terminal. Also, the Stop if True and Continue if True shortcut menu items change to Stop if Error and Continue While Error.
- **I** -The iteration terminal 'i' (an output terminal), contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

## STRUCTURE TUNNELS

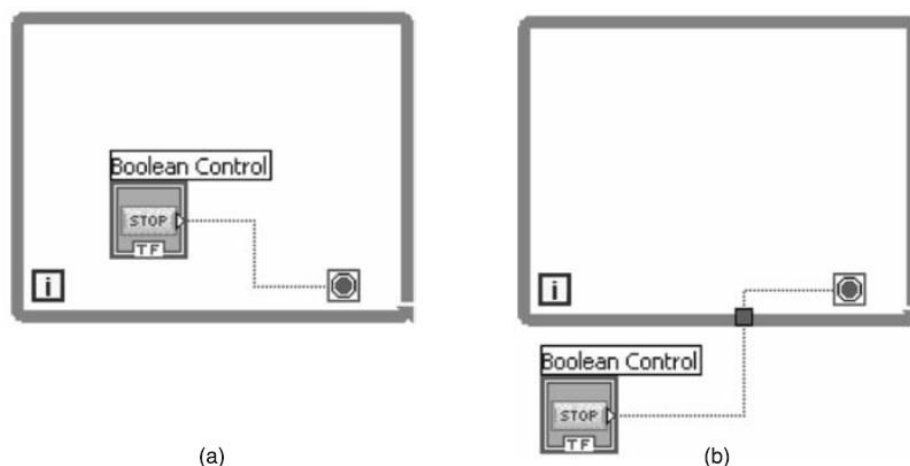
- Data can be passed out of or into a loop through a tunnel.
- Tunnels feed data into and out of structures.
- The tunnel appears as a solid block on the border of the loop.
- The block is the color of the data type wired to the tunnel.
- Data passes out of a loop after the loop terminates.
- When a tunnel passes data into a loop, the loop executes only after data arrives at the tunnel.
- In Figure 4.9, the iteration terminal is connected to a tunnel. The value in the tunnel does not pass to the Iteration Number indicator until the While Loop has finished execution. Only the last value of the iteration terminal displays in the Iteration Number indicator.



**Figure 4.9** Passing data outside the loop through a tunnel.

## TERMINALS INSIDE OR OUTSIDE LOOPS

- Inputs pass data into a loop at the start of loop execution.
- Outputs pass data out of a loop only after the loop completes all iterations.
- If you want the loop to check the value of a terminal for each iteration, place the terminal inside the loop.
- When you place the terminal of a front panel Boolean control inside a While Loop and wire the terminal to the conditional terminal of the loop, the loop checks the value of the terminal for every iteration to determine if it must iterate.
- You can stop the While Loop as shown in Figure 4.10(a) by changing the value of the front panel control to FALSE.



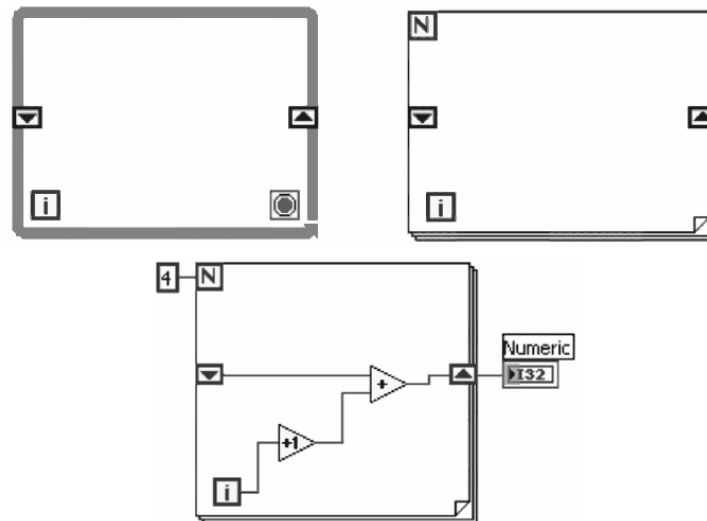
**Figure 4.10** (a) While Loop with Boolean control placed inside the loop and (b) While Loop with Boolean control placed outside the loop.

- If you place the terminal of the Boolean control outside the While Loop as shown in Figure 4.8(b), and the control is set to FALSE if the conditional terminal is Stop if True when the loop starts, you cause an infinite loop. You also cause an infinite loop if the control outside the loop is set to TRUE and the conditional terminal is Continue if True.
- Changing the value of the control does not stop the infinite loop because the value is only read once, before the loop starts. To stop an infinite loop, you must abort the VI by clicking the Abort Execution button on the toolbar.

## SHIFT REGISTERS

- When programming with loops, you often need to access data from previous iterations of the loop. For example, you may have a VI that reads the temperature and displays it on a graph. If you want to display a running average of the temperature as well, you need to use data generated in previous iterations. Two ways of accessing this data include the shift register and the feedback node.

- While Loops produce default data when the shift register is not initialized. For Loops produce default data if you wire 0 to the count terminal of the For Loop or if you wire an empty array to the For Loop as an input with auto-indexing enabled.
- The loop does not execute, and any output tunnel with auto-indexing disabled contains the default value for the tunnel data type. Use shift registers to transfer values through a loop regardless of whether the loop executes.
- Shift registers are used with For Loops and While Loops to transfer values from one loop iteration to the next.
- Shift registers are similar to static variables in text-based programming languages. A shift register appears as a pair of terminals, shown in Figure 4.11, directly opposite each other on the vertical sides of the loop border. The terminal on the right side of the loop contains an up arrow and stores data on the completion of an iteration. LabVIEW transfers the data connected to the right side of the register to the next iteration. After the loop executes, the terminal on the right side of the loop returns the last value stored in the shift register.



**Figure 4.11** Shift register operation.

- Create a shift register by right-clicking the left or right border of a loop and selecting the Add shift register from the shortcut menu.
- A shift register transfers any data type and automatically changes to the data type of the first object wired to the shift register. The data you wire to the terminals of each shift register must be the same type.
- You can add more than one shift register to a loop. If you have multiple operations that use previous iteration values within our loop, you can use multiple shift registers to store the data values from those different processes in the structure as shown in Figure 4.12.

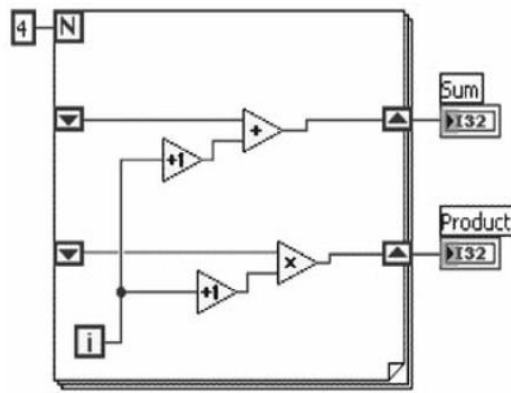


Figure 4.12 Multiple shift registers.

## INITIALIZING SHIFT REGISTERS

- Initializing the shift register resets the value the shift register passes to the first iteration of the loop when the VI runs.
- Initialize a shift register by wiring a control or constant to the shift register terminal on the left side of the loop.
- If you do not initialize the shift register, the loop uses the value written to the shift register when the loop last executed or the default value for the data type if the loop has never executed.
- Use a loop with an uninitialized shift register to run a VI repeatedly so that each time the VI runs, the initial output of the shift register is the last value from the previous execution.
- Use an uninitialized shift register to preserve state information between subsequent executions of a VI. After the loop executes, the last value stored in the shift register remains at the right terminal. If you wire the right terminal outside the loop, the wire transfers the last value stored in the shift register. You can add more than one shift register to a loop.
- If you have multiple operations that use previous iteration values within a loop, use multiple shift registers to store the data values from those different processes in the structure as shown in Figure 4.13.

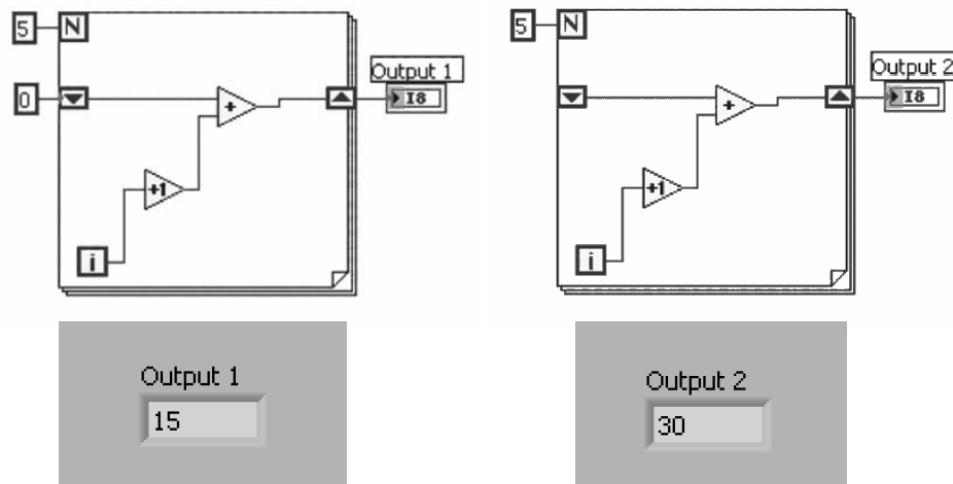


Figure 4.13 Initializing shift registers.

- The For Loop executes five times. The shift register value is added with the subsequent numbers starting from 1 to 5. After five iterations of the For Loop, the shift register passes the final value 15 to the indicator and the VI stops.
- Each time you run the VI, the shift register begins with a value of 0. The shift register is used without initialization. The first time you run the VI, the shift register begins with a value of 0, which is the default value for a 32-bit integer. After five iterations of the For Loop, the shift register passes the final value 15 to the indicator and ends. The next time you run the VI, the shift register begins with a value of 15, which was the last value from the previous execution. After five iterations of the For Loop, the shift register passes the final value 30 to the indicator. If you run the VI again, the shift register begins with a value of 30, and so on. Uninitialized shift registers retain the value of the previous iteration until you close the VI.

### STACKED SHIFT REGISTERS

- Stacked shift registers let you access data from previous loop iterations.
- Stacked shift registers remember values from multiple previous iterations and carry those values to the next iterations. This technique is useful for averaging data points.
- To create a stacked shift register, right-click the left terminal and select Add Element from the shortcut menu.
- Stacked shift registers, as shown in Figure 4.14, can only occur on the left side of the loop, because the right terminal only transfers the data generated from the current iteration to the next iteration.

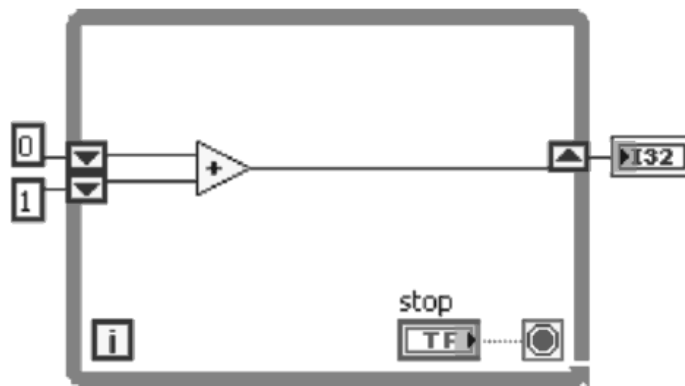


Figure 4.14 Stacked shift registers.

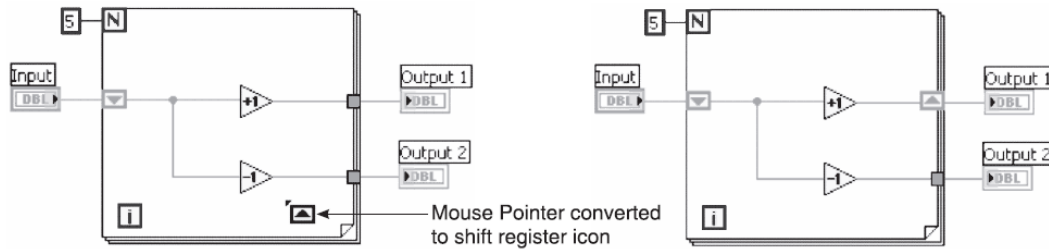
- If you add one more element to the left terminal, the value from the last two iterations carries over to the next iteration, with the most recent iteration value stored in the top shift register. The second terminal stores the data passed to it from the previous iteration. You can add more than two terminals in the shift register. If you add two more elements to the left terminal, the second terminal stores the data passed to it from the previous iteration and the bottom terminal stores the data passed to it from two iterations ago.

### REPLACING TUNNELS WITH SHIFT REGISTERS

- Tunnels can be replaced with shift registers wherever necessary.
- To replace a tunnel into a shift register, right-click the tunnel and select Replace with Shift Register.



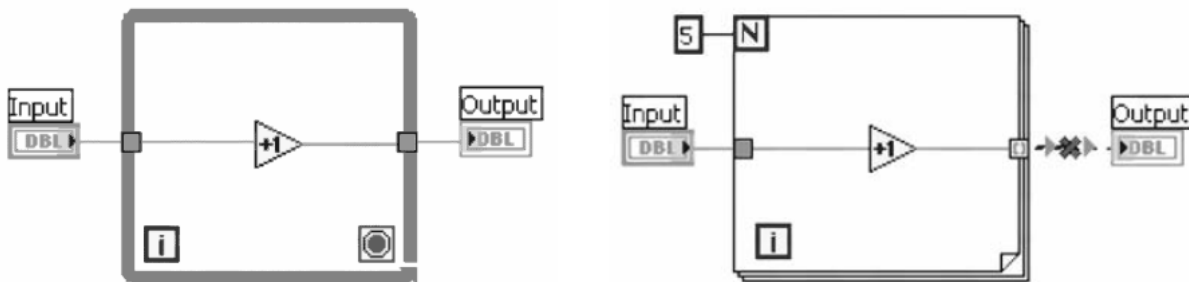
- If no tunnel exists on the loop border opposite of the tunnel you right-clicked, LabVIEW automatically creates a pair of shift register terminals. If one or more than one tunnel exists on the loop border opposite of the tunnel you right-clicked, the mouse pointer will turn to the symbol of a shift register. You can choose the particular tunnel which is to be converted as the shift register by clicking the mouse over that tunnel as shown in Figure 4.15.



**Figure 4.15** Replace a tunnel into a shift register.

## REPLACING SHIFT REGISTERS WITH TUNNELS

- Replace shift registers with tunnels when you no longer need to transfer values from one loop iteration to the next.
- To replace a shift register with a tunnel, right-click the shift register and select Replace with Tunnels.
- If you replace an output shift register terminal with a tunnel on a For Loop, the wire to any node outside the loop breaks because the For Loop enables auto-indexing by default as shown in Figure 4.16.

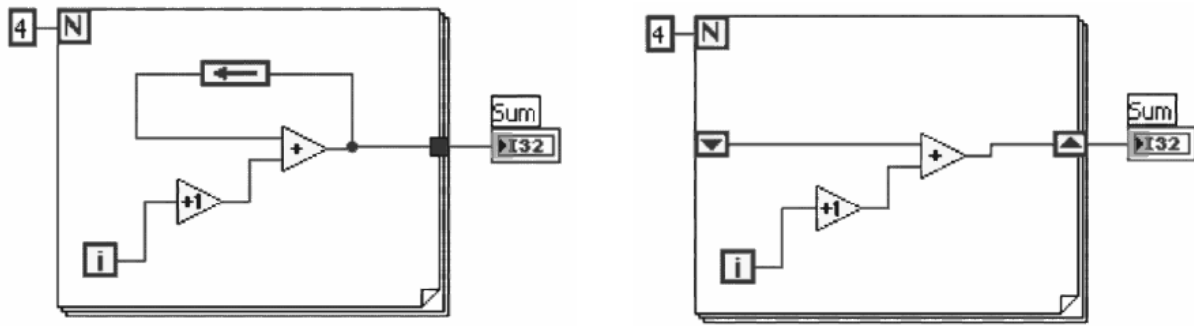


**Figure 4.16** Replace shift registers with tunnels.

- Right click the auto-indexed tunnel and select Disable Indexing on the tunnel to correct the broken wire. This problem does not occur in While Loops because auto-indexing is disabled by default in While Loops.

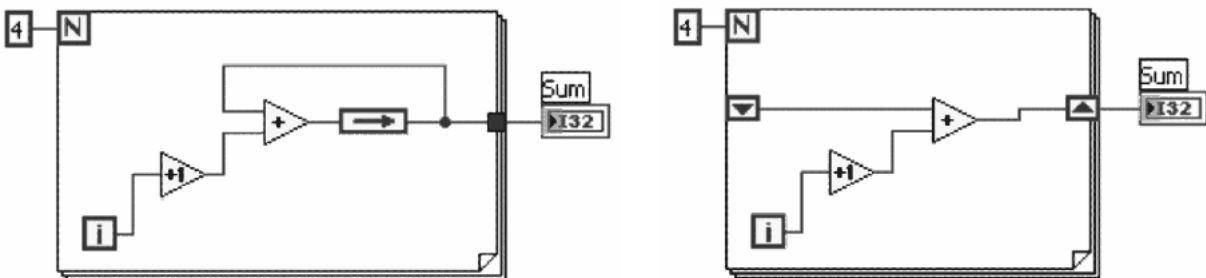
## FEEDBACK NODES

- When the output of a node is connected directly to the input, the feedback node is generated automatically.
- The feedback node shown in Figure 4.17 appears automatically in a For Loop or While Loop if we wire the output of a node or group of nodes to the input of that node or group of nodes.



**Figure 4.17** Feedback node.

- Like a shift register, the feedback node stores data when the loop completes an iteration, sends that value to the next iteration of the loop, and transfers any data type. Use the feedback node to avoid unnecessarily long wires in loops.
- The feedback node arrow indicates the direction in which the data flows along the wire. The arrow automatically changes direction if the direction of data flow changes.
- You also can select the Feedback Node on the Structures palette and place it inside a For Loop or While Loop.
- If you place the feedback node on the wire before you branch the wire that connects the data to the tunnel as in Figure 4.18, the feedback node passes each value to the tunnel.
- If you place the feedback node on the wire after you branch the wire that connects data to the tunnel, the feedback node passes each value back to the input of the VI or function and then passes the last value to the tunnel.



**Figure 4.18** Use of feedback node.

### INITIALIZING A FEEDBACK NODE

- Initializing a feedback node as shown in Figure 4.19 resets the initial value that the feedback node passes for the first time the loop executes when the VI runs.
- If you do not initialize the feedback node, the feedback node passes the last value written to the node or the default value for the data type if the loop has never executed.
- If you leave the input of the initializer terminal unwired, each time the VI runs, the initial input of the feedback node is the last value from the previous execution.
- To initialize a feedback node, right-click the Feedback Node and select Initializer Terminal from the shortcut menu and add wire a value from outside the loop to the initializer terminal. When you select the

Feedback Node on the Functions palette or if you convert an initialized shift register to a feedback node, the loop appears with an initializer terminal.

- To replace a feedback node with shift registers, right-click the Feedback Node and select Replace with Shift Register from the shortcut menu.
- To replace shift registers with a feedback node, right-click the Shift Registers and select Replace with Feedback Node from the shortcut menu.

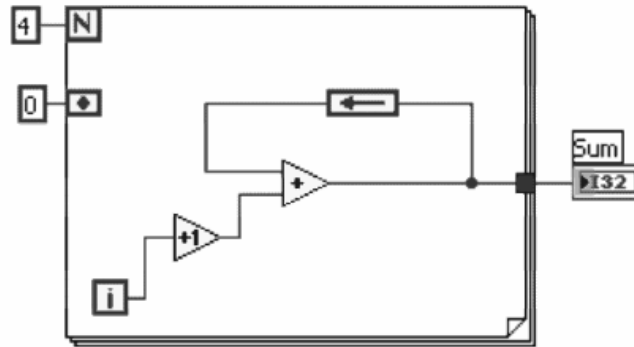


Figure 4.19 Initializing a feedback node.

## CONTROL TIMING

- When a loop finishes executing an iteration, it immediately begins executing the next iteration unless it reaches a stop condition.
- Most applications need precise control of the frequency or timing of the iteration to be maintained between successive operations of the loop.
- You might want to control the speed at which a process executes, such as the speed at which data values are plotted to a chart. You can use a wait function in the loop to wait an amount of time in milliseconds before the loop re-executes.
- LabVIEW consists of two wait functions. A wait function is placed inside a loop to allow a VI to sleep for a set amount of time. This allows your processor to address other tasks during the wait time.
- Wait functions use the operating system millisecond clock. They are Wait Until Next ms Multiple as shown in Figures 4.20(a) and Wait (ms) functions as shown in Figure 4.20(b).

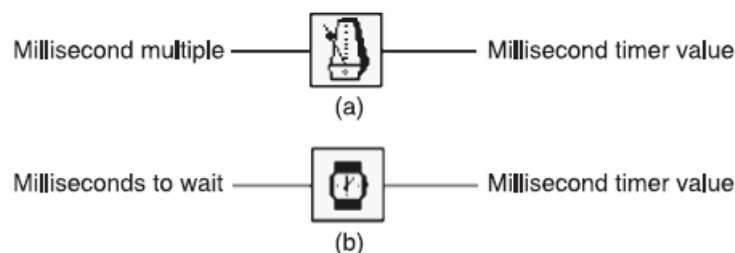
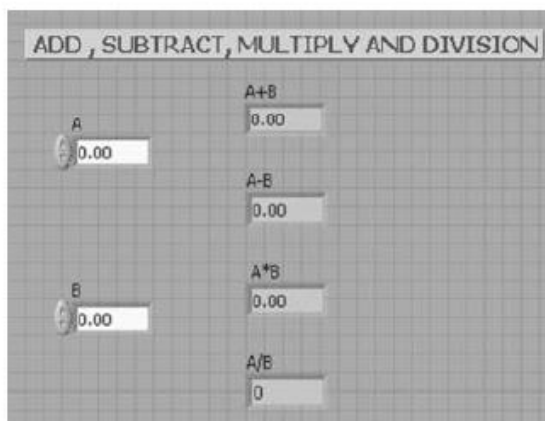


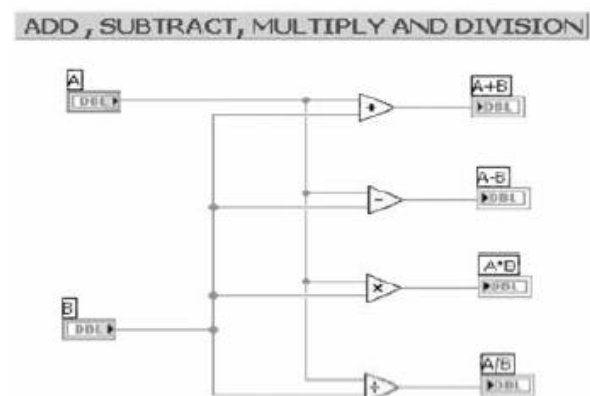
Figure 4.20 (a) Wait Until Next ms Multiple function and (b) Wait (ms) function.

- The Wait Until Next ms Multiple function monitors a millisecond counter and waits until the millisecond counter reaches a multiple of the time you specify. This function can be used for synchronization activities. You can place this function within a loop to control the loop execution rate. For this function to be effective, your code execution time must be less than the time specified for this function. The execution rate for the first iteration of the loop is indeterminate.
- The Wait (ms) function adds the wait time to the code execution time. This can cause a problem if code execution time is variable. The Wait (ms) function waits until the millisecond counter counts to an amount equal to the input you specify. This function guarantees that the loop execution rate is at least the amount of the input you specify
- The Time Delay Express VI, located on the Functions>>Execution Control palette, behaves similar to the Wait (ms) function with the addition of built-in error clusters.

### PROGRAM-1: Add, Multiply, Subtract and Divide Two Numeric Inputs

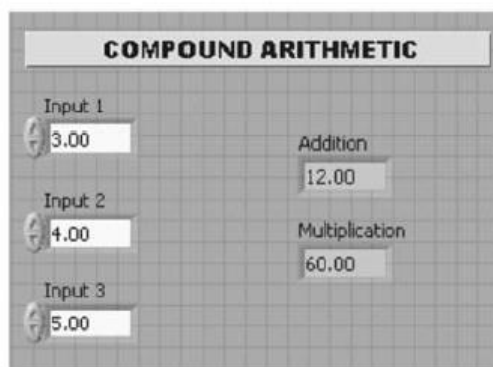


(a) Front panel

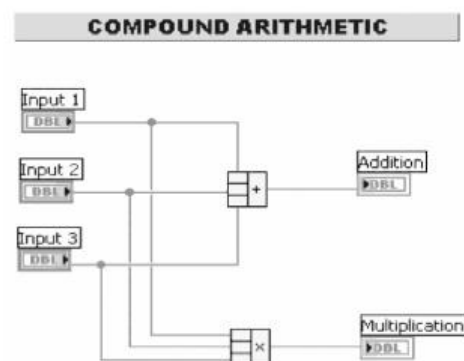


(b) Block diagram

### PROGRAM-2: Add and Multiply More Than Two Numeric Inputs.

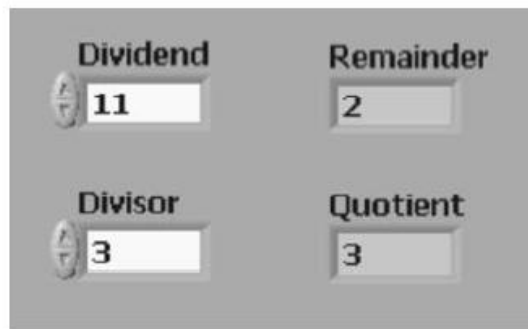


(a) Front panel

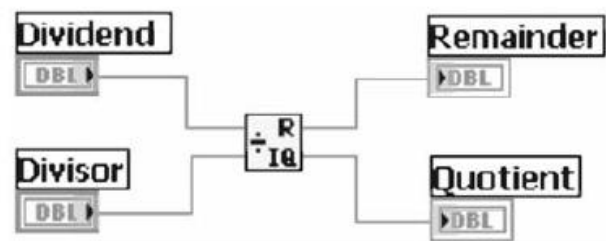


(b) Block diagram

**PROGRAM-3: Divide Two Numbers and Find the Remainder and Quotient.**

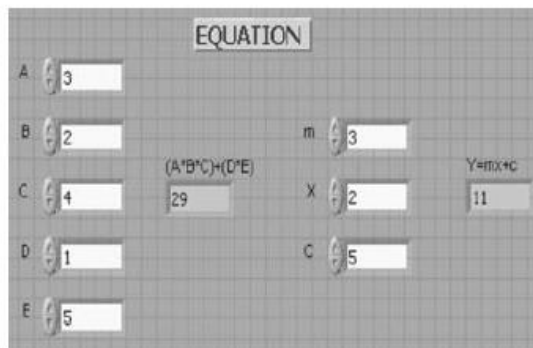


(a) Front panel

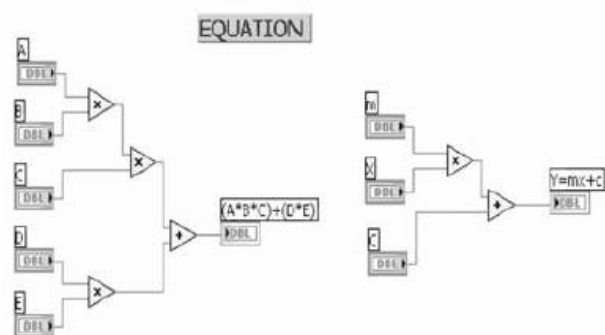


(b) Block diagram

**PROGRAM-4: Compute the Expressions  $Y = (A*B*C) + (D*E)$  and  $Y = mx + c$ .**

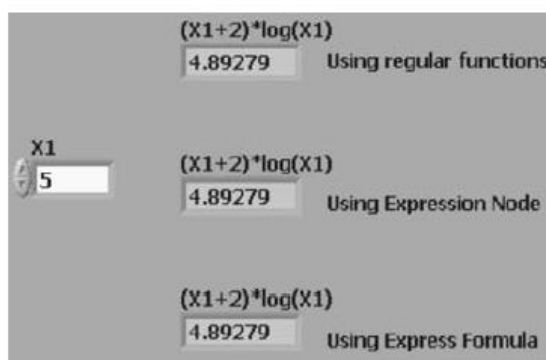


(a) Front panel

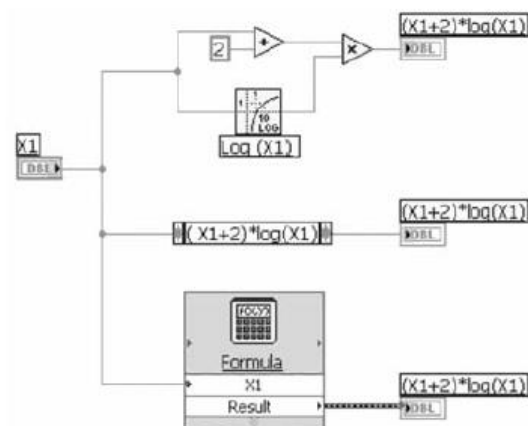


(b) Block diagram

**PROGRAM-5: Compute the equations  $(X1+2)*\log(X1)$  using functions, Expression node and Express Formula for the given inputs X1.**

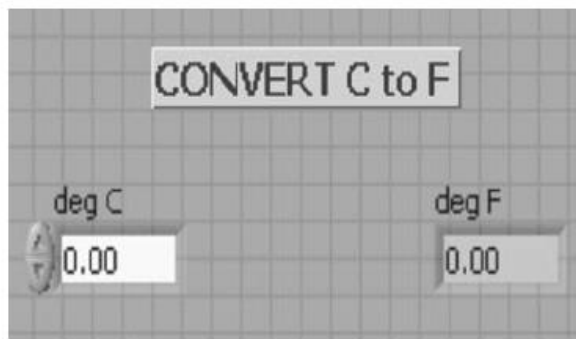


(a) Front panel

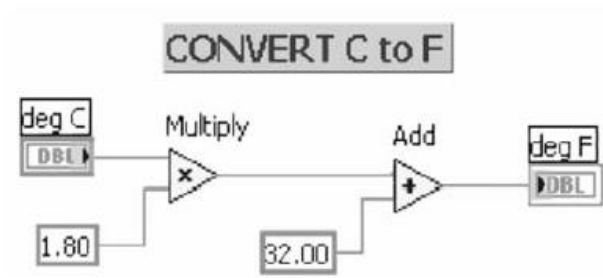


(b) Block diagram

**PROGRAM-6: Convert Celsius to Fahrenheit.**

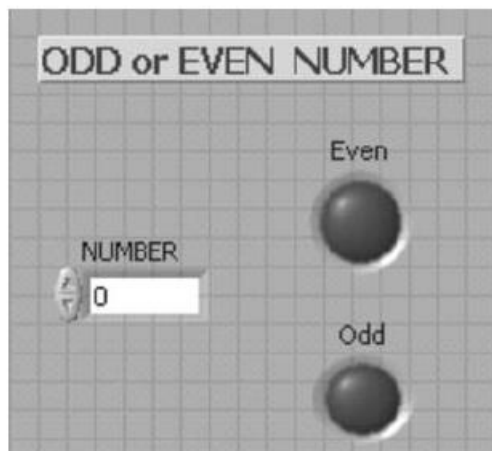


(a) Front panel

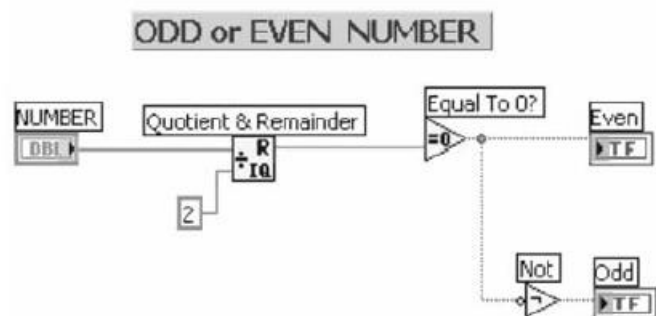


(b) Block diagram

**PROGRAM-7: Find whether the given number is odd or even.**

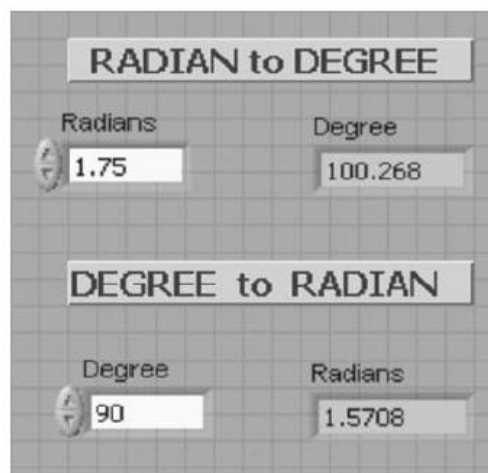


(a) Front panel

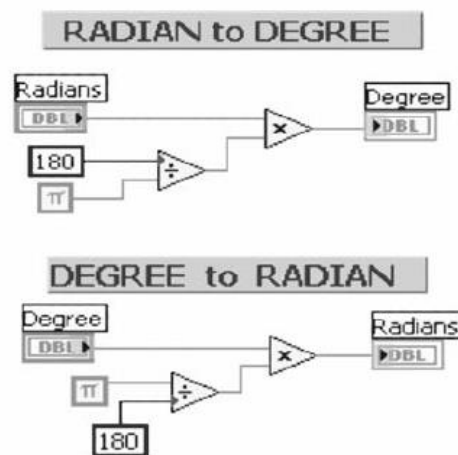


(b) Block diagram

**PROGRAM-8: Convert radians to degrees and degrees to radians.**



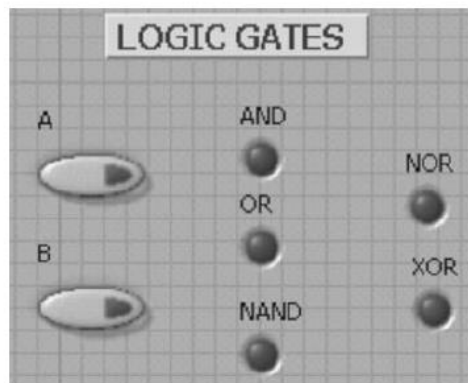
(a) Front panel



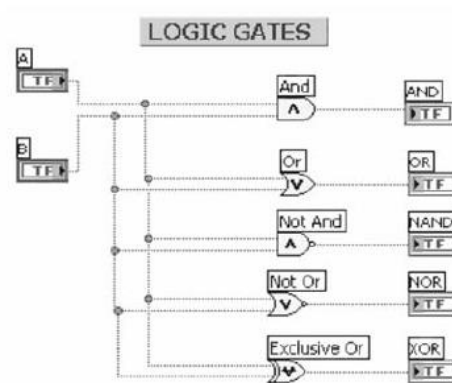
(b) Block diagram



**PROGRAM-9:** Perform various Boolean Operations (AND, OR, NAND, NOR, XOR).

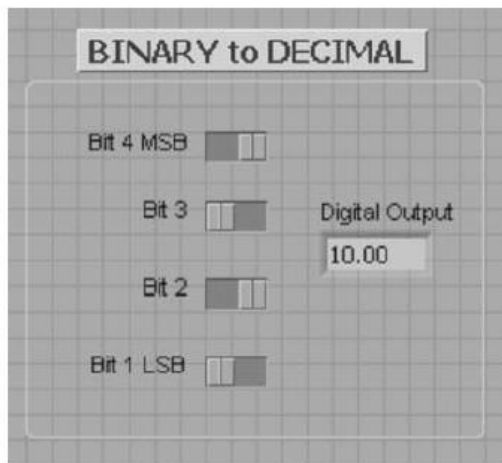


(a) Front panel

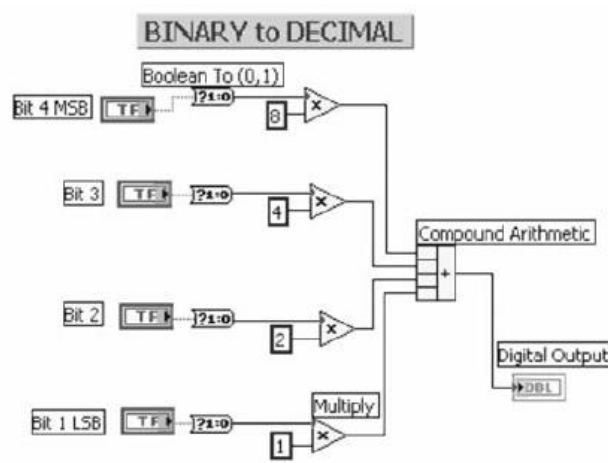


(b) Block diagram

**PROGRAM-10:** Convert a binary number to a decimal number.

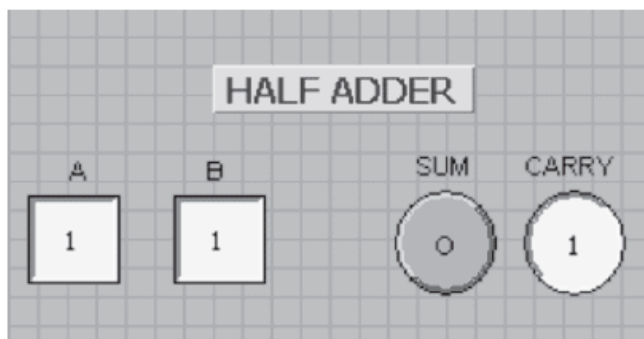


(a) Front panel

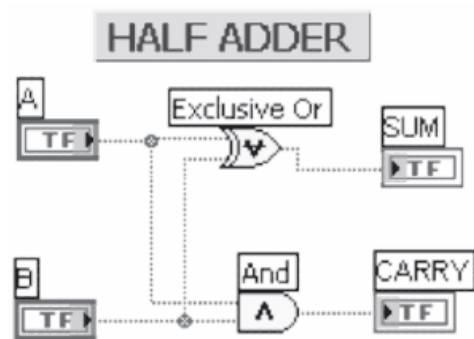


(b) Block diagram

**PROGRAM-11:** Add two binary bits and find the sum and carry (half adder).



(a) Front panel

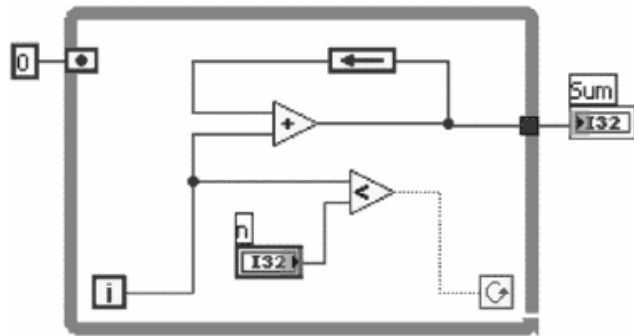


(b) Block diagram

**PROGRAM-12:** Create a VI to find the sum of first n natural numbers using a While Loop with a feedback node.



(a) Front panel

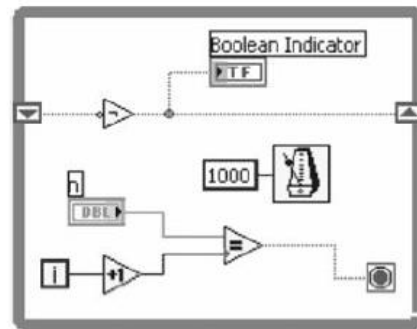


(b) Block diagram

**PROGRAM-13:** Create a VI to change the state of the Boolean indicator n times between TRUE and FALSE.



(a) Front panel

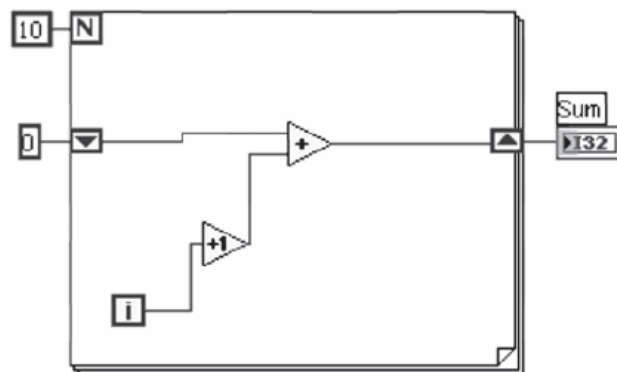


(b) Block diagram

**PROGRAM-14:** Create a VI to find the sum of first 10 natural numbers using a For Loop

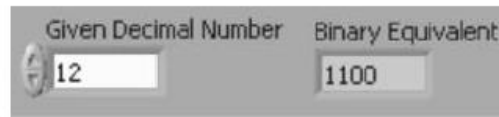


(a) Front panel

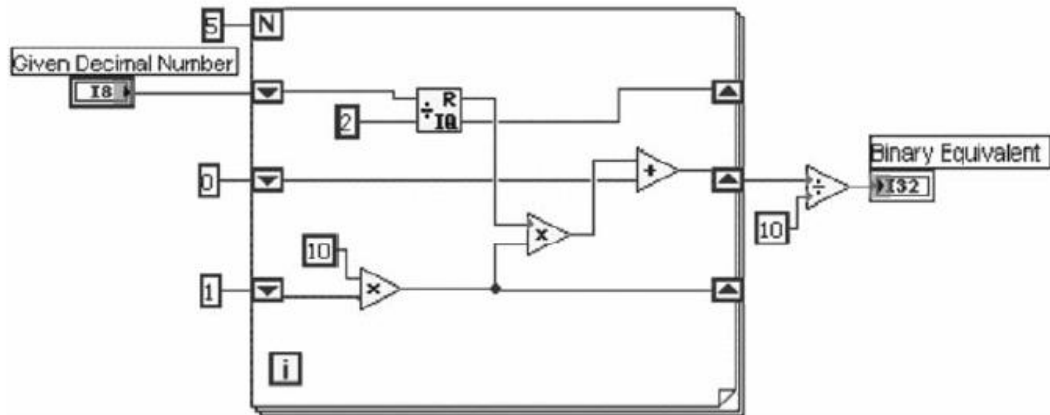


(b) Block diagram

**PROGRAM-15:** Create a VI which converts a decimal number to a binary number using For Loops



(a) Front panel

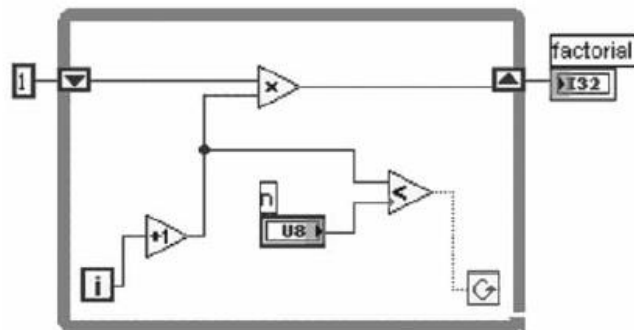


(b) Block diagram

**PROGRAM-16:** Create a VI to find the factorial of a given number using a While Loop.



(a) Front panel

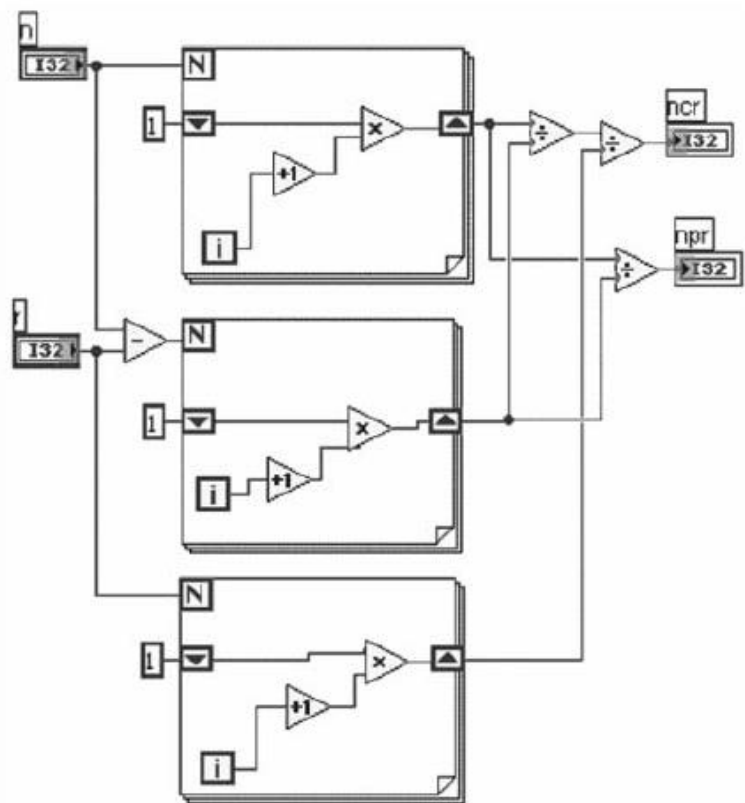


(b) Block diagram

**PROGRAM-16:** Create a VI to find  ${}^nC_r$  and  ${}^nP_r$  of a given number using a For Loop.



(a) Front panel



(b) Block diagram

JAIN COLLEGE OF ENG.