

2. LabVIEW Programming Environment

LabVIEW constitutes a graphical programming environment that allows one to design and analyze a DSP system in a shorter time as compared to text-based programming environments. LabVIEW graphical programs are called Virtual Instruments (VIs). VIs run based on the concept of data flow programming. This means that execution of a block or a graphical component is dependent on the flow of data, or more specifically a block executes when data is made available at all of its inputs. Output data of the block are then sent to all other connected blocks. Data flow programming allows multiple operations to be performed in parallel since its execution is determined by the flow of data and not by sequential lines of code.

2.1 Virtual Instruments (VIs)

A VI consists of two major components which include a Front Panel (FP), and a Block Diagram (BD). A FP provides the user-interface of a program while a BD incorporates its graphical code. When a VI is located within the block diagram of another VI, it is called a subVI. LabVIEW VIs are modular meaning that any VI or subVI can be run by itself.

2.1.1 Front Panel and Block Diagram

A FP contains the user interfaces of a VI shown in a BD. Inputs to a VI are represented by so called controls. Knobs, pushbuttons and dials are a few examples of controls. Outputs from a VI are represented by so called indicators. Graphs, LEDs (light indicators) and meters are a few examples of indicators. As a VI runs, its FP provides a display or user-interface of controls (inputs) and indicators (outputs).

A BD contains terminal icons, nodes, wires, and structures. Terminal icons are interfaces through which data are exchanged between a FP and a BD. Terminal icons correspond to controls or indicators that appear on a FP. Whenever a control or indicator is placed on a FP, a terminal icon gets added to the corresponding BD. A node represents an object which has input and/or output connectors and performs a certain function. SubVIs and functions are examples of nodes. Wires establish the flow of data in a BD. Structures are used to control the flow of a program such as repetitions or conditional executions. Figure 2-1 shows what a FP and a BD window look like.

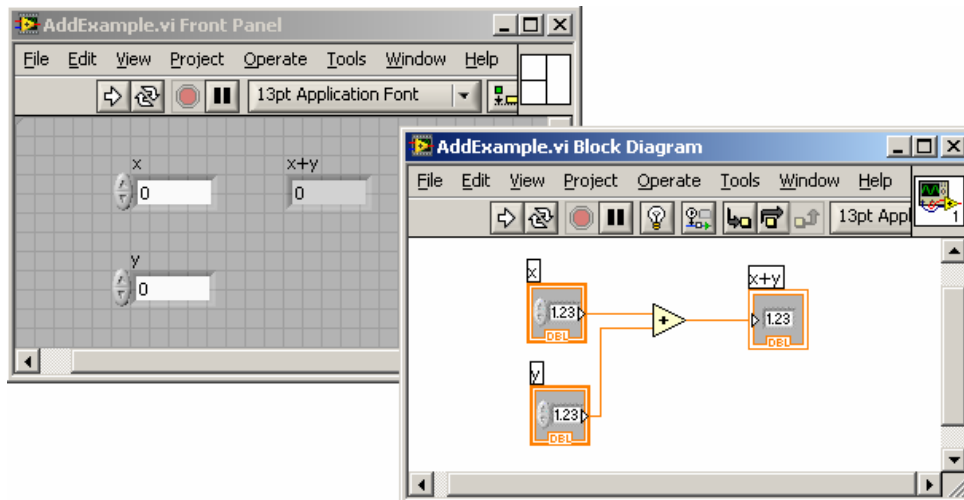


Figure 2-1: LabVIEW windows: Front Panel and Block Diagram.

2.1.2 Icon and Connector Pane

A VI icon is a graphical representation of a VI. It appears in the top right corner of a BD or a FP window. When a VI is inserted in a BD as a subVI, its icon gets displayed.

A connector pane defines inputs (controls) and outputs (indicators) of a VI. The number of inputs and outputs can be changed by using different connector pane

patterns. In Figure 2-1, a VI icon is shown at the top right corner of the BD and its corresponding connector pane having two inputs and one output is shown at the top right corner of the FP.

2.2 Graphical Environment

2.2.1 Functions Palette

The Functions palette, see Figure 2-2, provides various function VIs or blocks for building a system. This palette can be displayed by right clicking on an open area of a BD. Note that this palette can only be displayed in a BD.

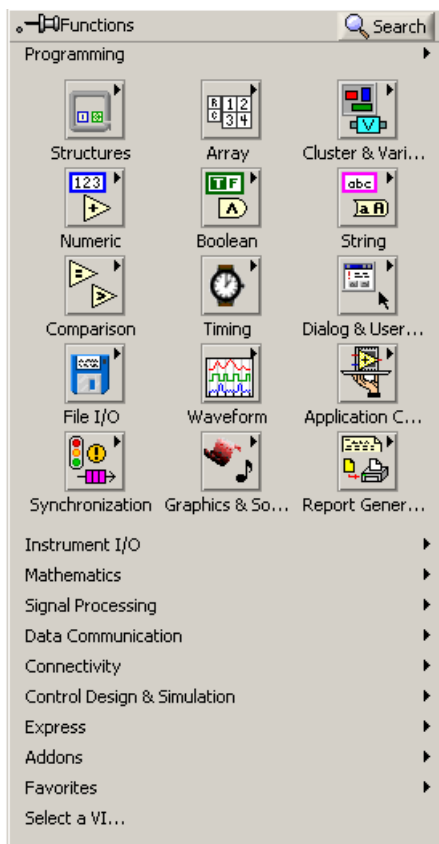


Figure 2-2: Functions palette.

2.2.2 Controls Palette

The Controls palette, see Figure 2-3, provides controls and indicators of a FP. This palette can be displayed by right clicking on an open area of a FP. Note that this palette can only be displayed in a FP.

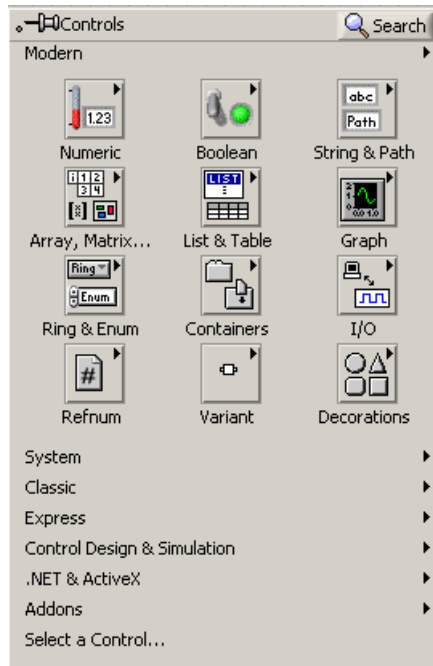


Figure 2-3: Controls palette.

2.2.3 Tools Palette

The Tools palette provides various operation modes of the mouse cursor for building or debugging a VI. The Tools palette and the frequently used tools are shown in Figure 2-4.

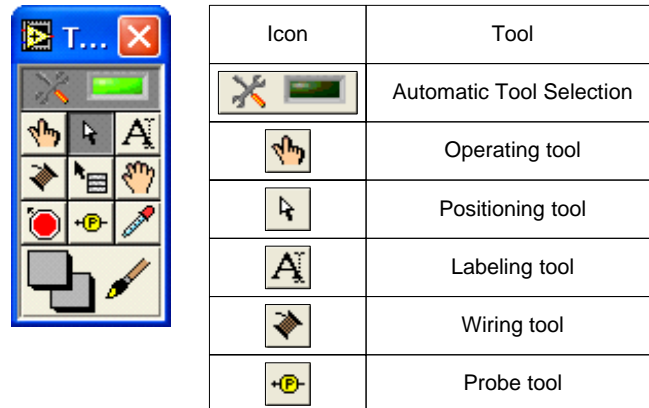


Figure 2-4: Tools palette.

Each tool is utilized for a specific task. For example, the Wiring tool is used to wire objects in a BD. If the automatic tool selection mode is enabled by clicking the **Automatic Tool Selection** button, LabVIEW selects the best matching tool based on a current cursor position.

2.3 Building a Front Panel

In general, a VI is put together by going back and forth between a FP and a BD, placing inputs/outputs on the FP and building blocks on the BD.

2.3.1 Controls

Controls make up the inputs to a VI. Controls grouped in the Numeric Controls palette (**Controls » Express » Num Ctrls**) are used for numerical inputs, grouped in the Buttons & Switches palette (**Controls » Express » Buttons**) for Boolean inputs, and grouped in the Text Controls palette (**Controls » Express » Text Ctrls**) for text and enumeration inputs. These control options are displayed in Figure 2-5.

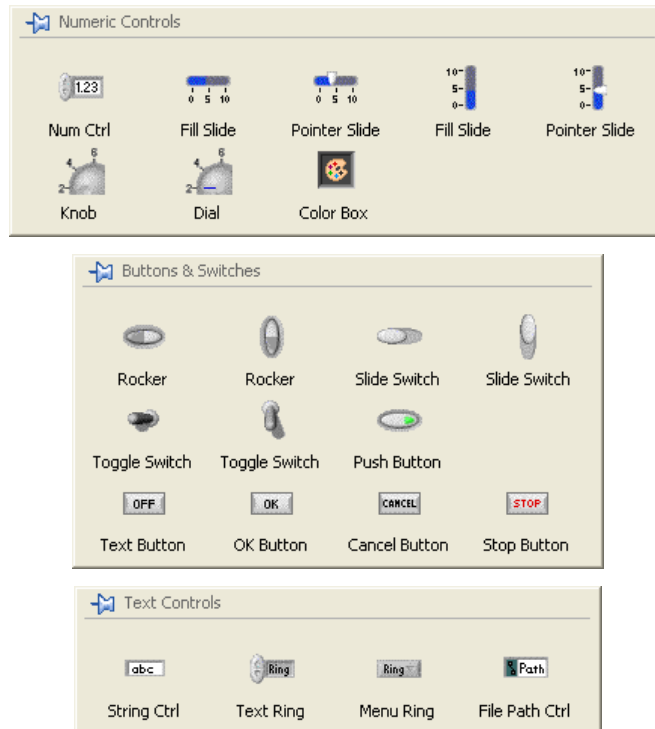


Figure 2-5: Control palettes.

2.3.2 Indicators

Indicators make up the outputs of a VI. Indicators grouped in the Numeric Indicators palette (**Controls » Express » Num Inds**) are used for numerical outputs, grouped in the LEDs palette (**Controls » Express » LEDs**) for Boolean outputs, grouped in the Text Indicators palette (**Controls » Express » Text Inds**) for text outputs, and grouped in the Graph Indicators palette (**Controls » Express » Graph Indicators**) for graphical outputs. These indicator options are displayed in Figure 2-6.

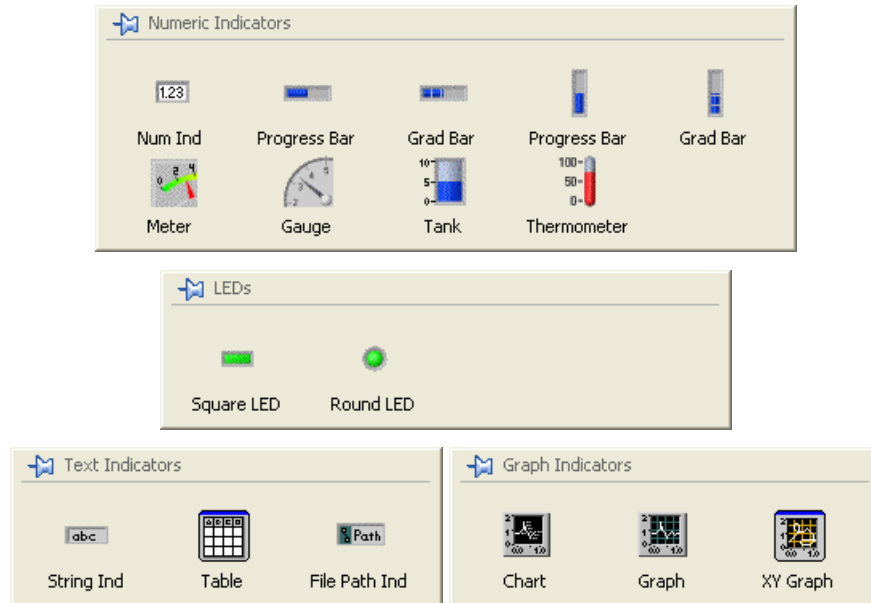


Figure 2-6: Indicator palettes.

2.3.3 Align, Distribute and Resize Objects

The menu items on the toolbar of a FP, see Figure 2-7, provide options to align and orderly distribute objects on the FP. Normally, after controls and indicators are placed on a FP, one uses these options to tidy up their appearance.





	Align Objects
	Distribute Objects
	Resize Objects
	Reorder

Figure 2-7: Menu for align, distribute, resize, and reorder objects.

2.4 Building a Block Diagram

2.4.1 Express VI and Function

Express VIs denote higher-level VIs that have been configured to incorporate lower-level VIs or functions. These VIs are displayed as expandable nodes with a blue background. Placing an Express VI in a BD brings up a configuration dialog window allowing adjustment of its parameters. As a result, Express VIs demand less wiring. A configuration window can be brought up by double clicking on its Express VI.

Basic operations such as addition or subtraction are represented by functions. Figure 2-8 shows three examples corresponding to three types of a BD object (VI, Express VI, and function).

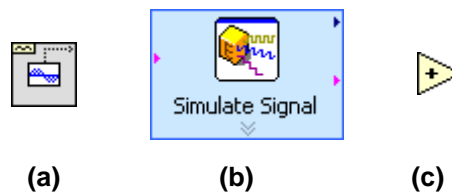


Figure 2-8: Block Diagram objects (a) VI, (b) Express VI, and (c) function.

A subVI or an Express VI can be displayed as icons or expandable nodes. If a subVI is displayed as an expandable node, the background appears yellow. Icons are used to save space in a BD, while expandable nodes are used to provide easier wiring or better readability. Expandable nodes can be resized to show their connection nodes more clearly. Three appearances of a VI/Express VI are shown in Figure 2-9.

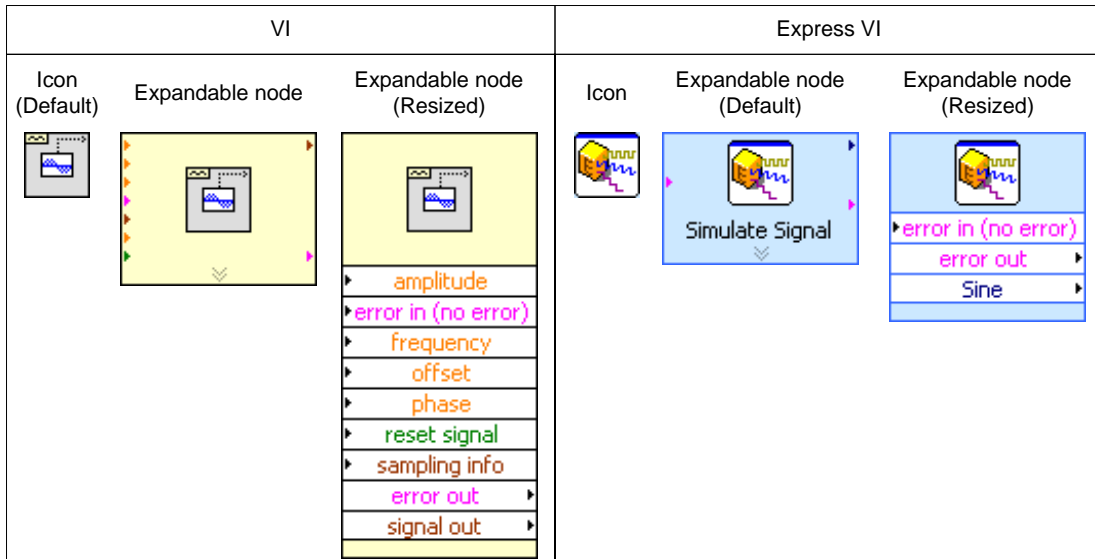


Figure 2-9: Icon vs. expandable node.

2.4.2 Terminal Icons

FP objects get displayed as terminal icons in a BD. A terminal icon exhibits an input or output as well as its data type. Figure 2-10 shows two terminal icon examples consisting of a double precision numerical control and indicator. As shown in this figure, terminal icons can be displayed as data type terminal icons to conserve space in a BD.

	Control	Indicator
Terminal Icons		
Data Type Terminal Icons		

Figure 2-10: Terminal icon examples displayed in a BD.

2.4.3 Wires

Wires transfer data from one node to another in a BD. Based on the data type of a data source, the color and thickness of its connecting wires change.

Wires for the basic data types used in LabVIEW are shown in Figure 2-11. Other than the data types shown in this figure, there are some other specific data types. For example, the dynamic data type is always used for Express VIs, and the waveform data type, which corresponds to the output from a waveform generation VI, is a special cluster of waveform components incorporating trigger time, time interval, and data value.





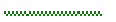





Wire Type	Scalar	1D Array	2D Array	Color
Numeric				Orange (Floating point) Blue (Integer)
Boolean				Green
String				Pink

Figure 2-11: Basic types of wires [2].

2.4.4 Structures

A structure is represented by a graphical enclosure. The graphical code enclosed by a structure gets repeated or executed conditionally. A loop structure is equivalent to a for loop or a while loop statement encountered in text-based programming languages, while a case structure is equivalent to an if-else statement.

2.4.4.1 For Loop

A For Loop structure is used to perform repetitions. As illustrated in Figure 2-12, the displayed border indicates a For Loop structure, where the count terminal  represents the number of times the loop is to be repeated. It is set by wiring a value

from outside of the loop to it. The iteration terminal **i** denotes the number of completed iterations, which always starts at zero.



Figure 2-12: For Loop.

2.4.4.2 While Loop




A While Loop structure allows repetitions depending on a condition, see Figure 2-13. The conditional terminal  initiates a stop if the condition is true. Similar to a For Loop, the iteration terminal **i** provides the number of completed iterations, always starting at zero.



Figure 2-13: While Loop.

2.4.4.3 Case Structure

A Case structure, see Figure 2-14, allows running different sets of operations depending on the value it receives through its selector terminal, which is indicated by . In addition to Boolean type, the input to a selector terminal can be of integer, string, or enumerated type. This input determines which case to execute. The case selector  shows the status being executed. Cases can be added or deleted as needed.

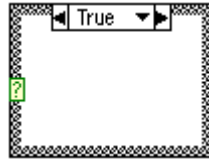


Figure 2-14: Case structure.

2.5 MathScript

MathScript is a new feature of the latest version of LabVIEW (LabVIEW 8.0) which allows one to perform textual programming in conjunction with graphical programming [6]. It includes more than 500 built-in functions and uses matrices and arrays as fundamental data types with built-in operators for data manipulation. User-defined functions can also be added to it. MathScript is compatible with the m-file script syntax that is encountered in MATLAB codes. MathScript possesses an interactive and a programming interface named MathScript Interactive Window and MathScript Node, respectively.

The LabVIEW MathScript Interactive Window is shown in Figure 2-15. The interfaces Command Window, Output Window, and MathScript Window are shown in this figure. The Command Window interface is used to enter commands and for script debugging or to view help statements for built-in functions. The Output Window interface is used for viewing output values. The MathScript Window interface is used to display variables, edit scripts, and display command history. Script editing allows the execution of a group of commands.

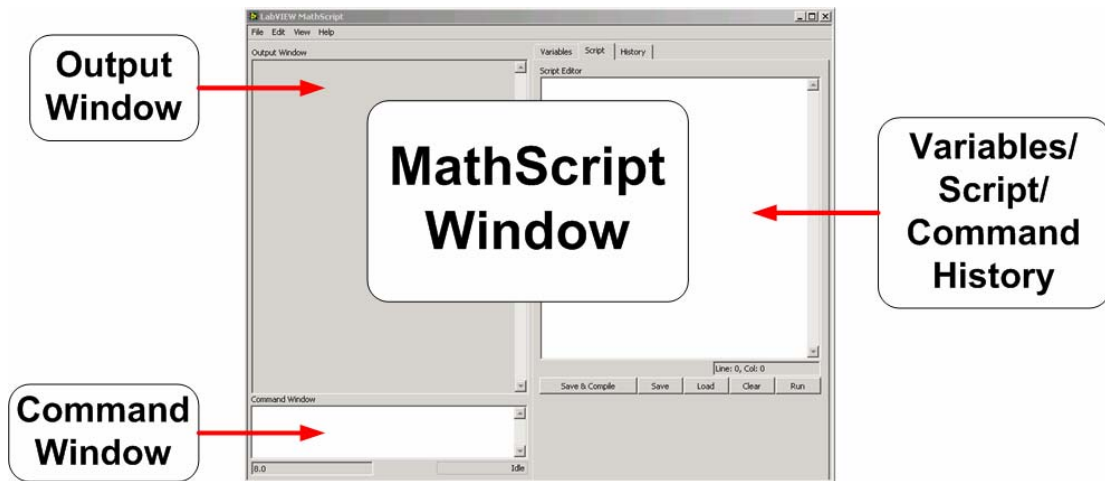


Figure 2-15: MathScript Interactive Window Interface.

A MathScript Node represents the textual code via a blue rectangle as shown in Figure 2-16. Its inputs and outputs are defined on the border of this rectangle for transferring data between the graphical environment and textual MathScript codes. For example, as indicated in Figure 2-16, the input variables on the left side, namely `lowcutoff`, `uppcutoff` and `order`, transfer values to the m-file script and the output variables on the right side, namely `F` and `sH`, transfer values to the graphical code. This process makes it easy to utilize m-file script variables within the graphical programming environment.

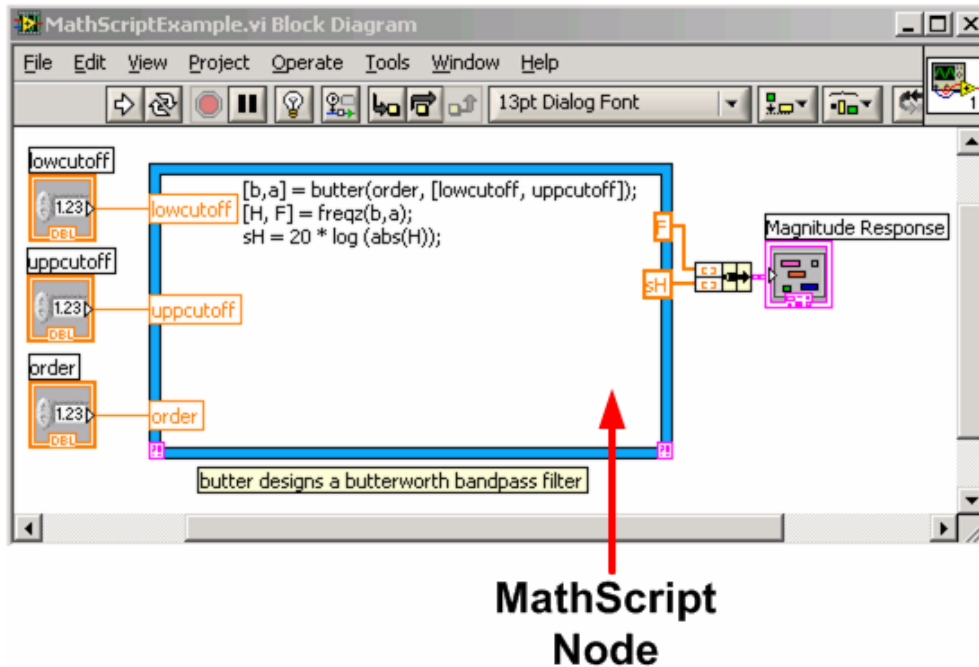


Figure 2-16: MathScript Node Interface.

2.6 Grouping Data: Array & Cluster

An array represents a group of elements having the same data type. An array consists of data elements having a dimension up to $2^{31} - 1$. For example, if a random number is generated in a loop, it makes sense to build the output as an array since the length of the data element is fixed at 1 and the data type is not changed during iterations.

A cluster consists of a collection of different data type elements, similar to the structure data type in text-based programming languages. Clusters allow one to reduce the number of wires on a BD by bundling different data type elements together and passing them to only one terminal. An individual element can be added to or extracted from a cluster by using the cluster functions such as `Bundle by Name` and `Unbundle by Name`.

2.7 Debugging and Profiling VIs

2.7.1 Probe Tool

The Probe tool is used for debugging VIs as they run. The value on a wire can be checked while a VI is running. Note that the Probe tool can only be accessed in a BD window.

The Probe tool can be used together with breakpoints and execution highlighting to identify the source of an incorrect or an unexpected outcome. A breakpoint is used to pause the execution of a VI at a specific location while execution highlighting helps one to visualize the flow of data during program execution.

2.7.2 Profile Tool

The Profile tool can be used to gather timing and memory usage information, i.e. how long a VI takes to run and how much memory it consumes. It is necessary to make sure a VI is stopped before setting up a Profile window.

An effective way to become familiar with LabVIEW programming is by going through examples. Thus, in the two labs that follow in this chapter, most of the key programming features of LabVIEW are presented via building some simple VIs. More detailed information on LabVIEW programming can be found in [1-6].

2.8 Bibliography

- [1] National Instruments, *LabVIEW Getting Started with LabVIEW*, Part Number 323427A-01, 2003
- [2] National Instruments, *LabVIEW User Manual*, Part Number 320999E-01, 2003

- [3] National Instruments, *LabVIEW Performance and Memory Management*, Part Number 342078A-01, 2003.
- [4] National Instruments, *Introduction to LabVIEW Six-Hour Course*, Part Number 323669B-01, 2003.
- [5] Robert H. Bishop, *Learning With Labview 7 Express*, Prentice Hall, 2003.
- [6] National Instruments, *Inside LabVIEW MathScript*, <http://zone.ni.com/devzone/conceptd.nsf/webmain>.