CHAPTER 12

Data Acquisition with LabVIEW

Chapter Outline

12.1 Introduction 347
12.2 Computer-Based DAQ 348
12.3 Acquisition of Data 348
12.4 National Instruments LabVIEW 349
12.4.1 Virtual Instruments 350
12.5 Introduction to Graphical Programming in LabVIEW 351
12.6 Elements of the Tools Palette 353
12.7 Logic Operations in LabVIEW 355
12.8 Loops in LabVIEW 356
12.9 Case Structure in LabVIEW 357
12.10 DAQ Using LabVIEW 359
12.10.1 LabVIEW Function Generation 361
12.11 LabVIEW Implementation of Digital Filters 364
12.11.1 Higher Order Digital Filters in LabVIEW 365
12.12 Summary 368
12.13 Exercises 368

12.1 Introduction

This chapter is designed to introduce the reader to the concept of computer-based data acquisition (DAQ) and to LabVIEW, a software package developed by National Instruments that is used extensively in laboratory settings. The main reason for focusing on LabVIEW is its prevalence in both academic and industry settings. We should also point out that Matlab and other software tools used to model and simulate dynamic systems and to perform extensive numerical computations are at times used in laboratory setting although their use is often limited to specialized applications such as real-time control as opposed to DAQ. For this reason, we do not discuss these tools in the present chapter.

LabVIEW itself is an extensive programming platform. It includes a multitude of functionalities ranging from basic algebraic operators to advanced signal-processing components that can be integrated into rather sophisticated and complex programs for use

Measurement and Instrumentation. http://dx.doi.org/10.1016/B978-0-12-800884-3.00012-5

in laboratory (and even industrial setting). For pedagogical reasons and given the context of this presentation, we will only introduce the main ideas from LabVIEW that are necessary for functioning in a typical undergraduate engineering laboratory environment. Advanced programming skills can be developed over time as the reader gains comfort with the basic functioning of LabVIEW and its external interfaces.

Specific topics discussed in this chapter and the associated learning objectives are as follows:

- Structure of personal computer (PC)-based DAQ systems, the purpose of DAQ cards, and the role of LabVIEW in this context
- Development of simple Virtual Instruments (VIs) using basic functionalities of LabVIEW, namely arithmetic and logic operations
- Construction of functionally enhanced VIs using LabVIEW program flow control operations such as the while loop and the case structure
- Development of VIs that allow for interaction with external hardware as for instance acquisition of external signals via DAQ card input channels and generation of functions using DAQ card output channels

These functionalities are essential to use LabVIEW in laboratory setting. Additional capabilities of LabVIEW are explored in the subsequent chapter on signal processing in LabVIEW and will be introduced accordingly.

12.2 Computer-Based DAQ

In studying mechanical systems, it is often necessary to use electronic sensors to measure certain variables such as temperature (using thermocouples or Resistance Temperature Detectors; i.e. RTDs), pressure (using piezoelectric transducers), strain (using strain gauges), and so forth. Although it is possible to use oscilloscopes (as for instance depicted in Figure 12.1) or voltmeters to monitor these variables, it is often preferable to use a PC to view and record the data through the use of a DAQ card. One particular advantage of using computers in this respect is that data can easily be stored and converted to a format that can be used by spreadsheets (such as Microsoft Excel) or other software packages such as Matlab for more extensive analysis. Another advantage is that significant digital processing of the data can be performed in real-time via the same platform that is used to acquire the data. This can significantly improve the process of performing an experiment by making the real-time data more useful for further processing.

12.3 Acquisition of Data

One important step in the DAQ process is the conversion of analog signals received from sensing instruments to digital representations that can be processed by the computer.

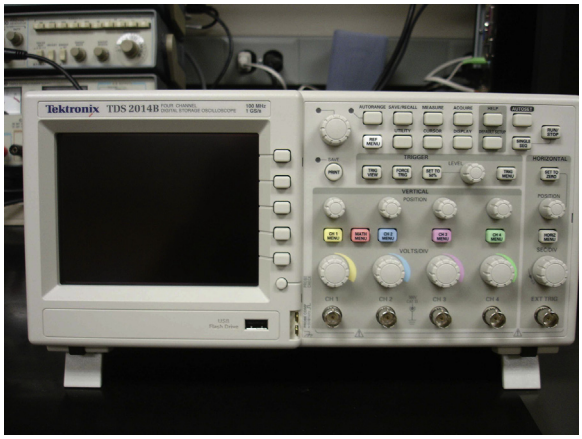

Figure 12.1
A typical laboratory oscilloscope.

Since data must be stored in the computer's memory in the form of individual data points represented by binary numbers, incoming analog data must be *sampled* at discrete time intervals and *quantized* (rounded off) to one of a set of predefined values. In most cases, this is accomplished using a digital-to-analog conversion component on the DAQ card inside the PC or interconnected to it via a Universal Serial Bus (USB) port. Note that both options are used commonly. However, laptop computers and/or low profile PCs generally require the use of USB-based DAQ devices (such as MyDAQ produced by National Instruments).

12.4 National Instruments LabVIEW

LabVIEW is a software package that provides control and a user interface for the DAQ process. Figure 12.2 depicts a schematic of data flow in the DAQ process. Note that the physical system may be a mechanical system such as a beam subjected to stress, a chemical process such as a distillation column, a DC motor with both mechanical and electrical components, and so forth. The key issue here is that certain measurements are taken from the given physical system and are acquired and processed by the PC-based DAQ system.

LabVIEW plays a pivotal role in the DAQ process. Through the use of VIs, LabVIEW directs the real-time sampling of sensor data through the DAQ card (also known as the I/O card) and is capable of storing, processing, and displaying the collected data. In most cases, one or more sensors transmit analog readings to the DAQ card in the computer. These analog data are then converted to individual digital values by the DAQ card and is made available to LabVIEW, at which point it can be displayed to the user. Although LabVIEW is capable of some data analysis functions, it is often preferable to export the data to a spreadsheet for detailed analysis and graphical presentation.

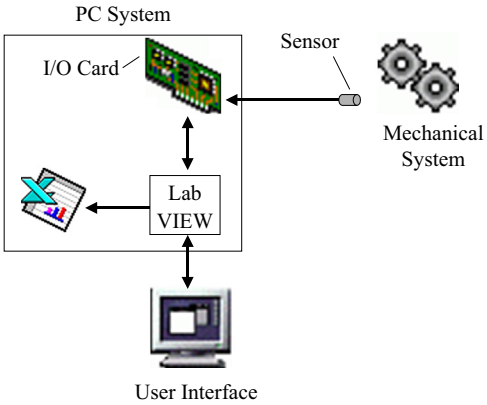

Figure 12.2
Schematic of data acquisition process.

12.4.1 Virtual Instruments

A VI is a program, created in the LabVIEW programming environment that simulates physical or hard instruments such as oscilloscopes and function generators. A simple VI used to produce a waveform is depicted in Figure 12.3. The Front Panel (shown in the figure) acts as the *user interface* while the DAQ (in this case the generation process) is performed by the combination of the PC and the DAQ card. Much like the Front Panel

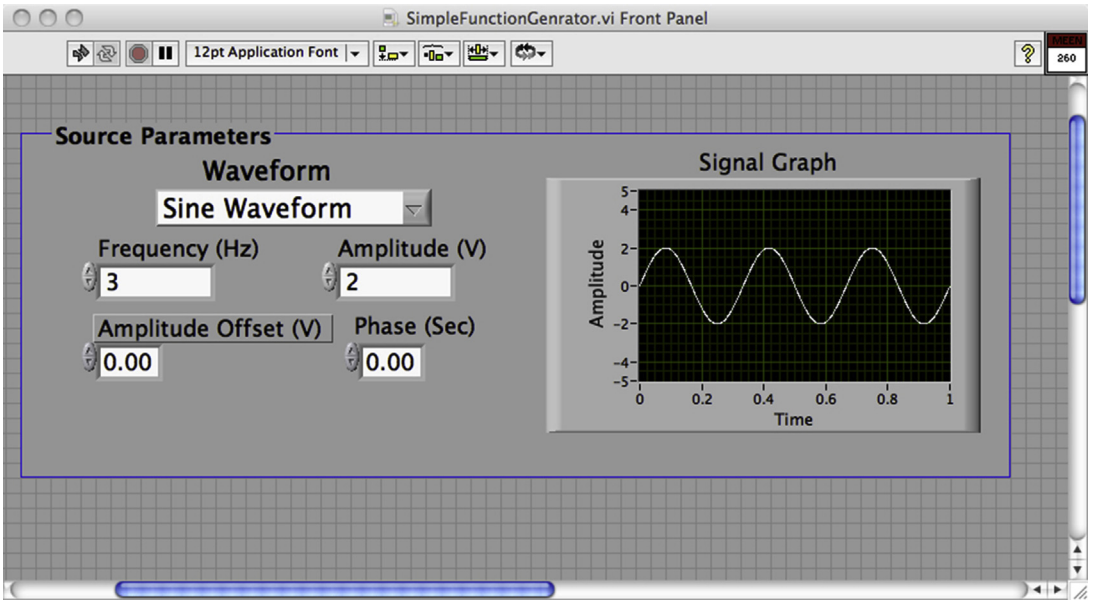

Figure 12.3
A simple function generator Virtual Instrument.

of a real instrument (such as the oscilloscope in Figure 12.1), the Front Panel window contains *controls* (i.e., knobs, switches, etc.) that allow the user to modify certain parameters during the experiment. These include a selector to choose the type of waveform, numerical controls to choose the frequency, and amplitude of the generated waveform as well as its phase and amplitude offset.

The Front Panel of a VI typically also contains *indicators* that display data or other important information relating to the experiment. In this case, a *graph* is used to depict the waveform. The Block Diagram (not shown but further discussed below) is analogous to the wiring and internal components of a real instrument. The configuration of the VI's Block Diagram determines how the Front Panel controls and indicators are related. It also incorporates functions such as communicating with the DAQ card and exporting data to disk files in spreadsheet format.

12.5 Introduction to Graphical Programming in LabVIEW

LabVIEW makes use of a graphical programming language that determines how each VI will work. In this section, we discuss the inner workings of a simple LabVIEW VI that is used to add and subtract two numbers. While this VI is not particularly useful in laboratory setting, it illustrates how basic LabVIEW components can be used to construct a VI and hence helps the reader move toward developing more sophisticated VIs. Figure 12.4 shows the Front Panel and Block Diagram of the VI, which accepts two numbers from the user (X and Y) via two simple *numeric controls*, and produces the sum and difference ($X + Y$ and $X - Y$) of the numbers that are displaced on the Front Panel via two simple *numeric indicators*.

The Block Diagram of the VI is a graphical, or more accurately a *data* flow, program that defines how the controls and indicators on the Front Panel are interrelated. The controls on the Front Panel of the VI show values that can be modified by the user while the VI is operating. The indicators display values that are output by the VI. Each control and indicator in the Front Panel window are associated with a *terminal* in the Block Diagram window. The wires in the Block Diagram window represent the flow of data within the VI. *Nodes* are additional programming elements that can perform operations on variables, perform input or output functions through the DAQ card, and serve a variety of other functions as well.

The two nodes in the VI shown in Figure 12.4 (Add and Subtract) have two inputs and one output, as depicted in Figure 12.5. Data can be passed to a node through its input terminals (usually on the left) and the results can be accessed through the node's output terminals, usually on the right.

Since LabVIEW diagrams are data flow driven, the sequence in which the various operations in the VI are executed is *not* determined by the *order* of a set of commands. Rather, a Block

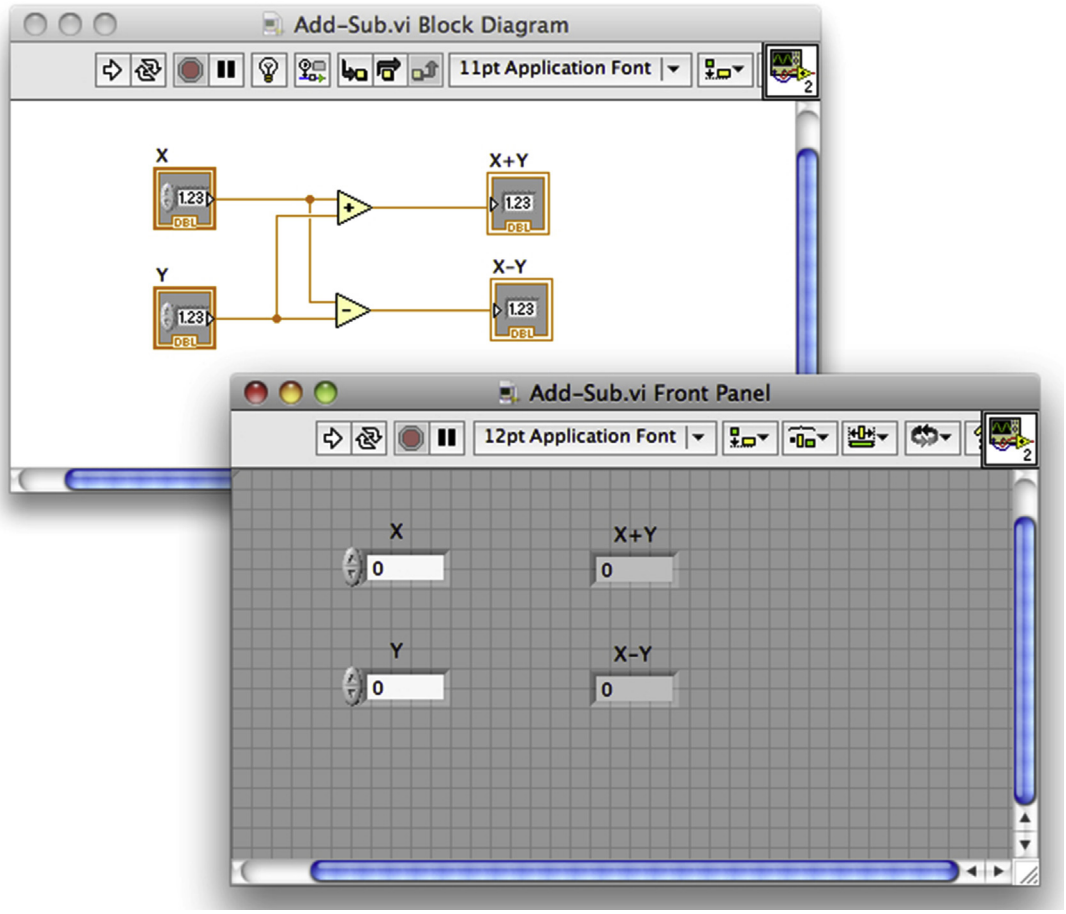

Figure 12.4
Addition and subtraction Virtual Instrument.

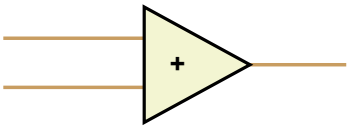

Figure 12.5
The Add node.

Diagram node executes when data are present at all of its input terminals. As a result, in the case of the Block Diagram in Figure 12.4, one does not know whether the Add node or the Subtract node will execute first. This issue has implications in more complex applications but is not particularly important in the present context. On the other hand, one cannot assume an order of execution merely on the basis of the position of the computational nodes (top to bottom or left to right). If a certain execution order is required or desired, one must explicitly

build program flow control mechanisms into the VI, which, in practice, is not always possible nor is it in the spirit in which LabVIEW was originally designed.

12.6 Elements of the Tools Palette

The mouse pointer can perform a number of different functions in the LabVIEW environment depending on which pointer tool is selected. One can change the mouse pointer tool by selecting the desired tool from the Tools palette shown in Figure 12.6. (If the Tools palette does not appear on the screen it can be displayed by selecting Tools on the View menu.)

The choices available in the Tools palette are as follows:

- *Automatic tool selection*. Automatically selects the tool it assumes you need depending on context. For instance, it can be the *positioning tool*, or the *wiring tool*, or the *text tool* as further noted below.
- *The Operating tool*. This tool is used to operate the Front Panel controls before or while the VI is running.
- *The Positioning tool*. This tool is used to select, move, or resize objects in either the Front Panel or the Diagram windows. For example, to delete a node in a diagram, one would first select the node with the Positioning tool, and then press the Delete key.

Figure 12.6
The Tools palette.

- *The Text tool*. This tool is used to add or change a label. The Enter key is used to finalize the task.
- *The Wiring tool*. This tool is used to wire objects together in the Diagram window. When this tool is selected, you can move the mouse pointer over one of a node's input or output terminals to see the description of that terminal.

One can add controls and indicators to the Front Panel of a VI by dragging and dropping them from the Controls palette, depicted in Figure 12.7 and which is visible *only* when the Front Panel window is *active*.

If for some reason the Controls palette is not visible, one can access it by right-clicking anywhere in the Front Panel window. Once a control or indicator is added to the Front Panel of a VI, the corresponding terminal is automatically created in the Block Diagram window. Adding additional nodes to the Block Diagram requires the use of the Functions palette, which is accessible once the Block Diagram window is visible. One can add arithmetic and logic elements to the Block Diagram window by dragging and dropping

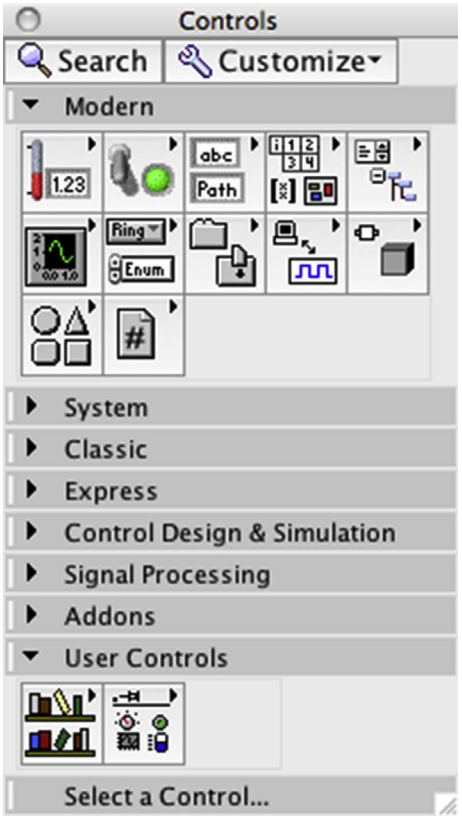

Figure 12.7
The Controls palette.

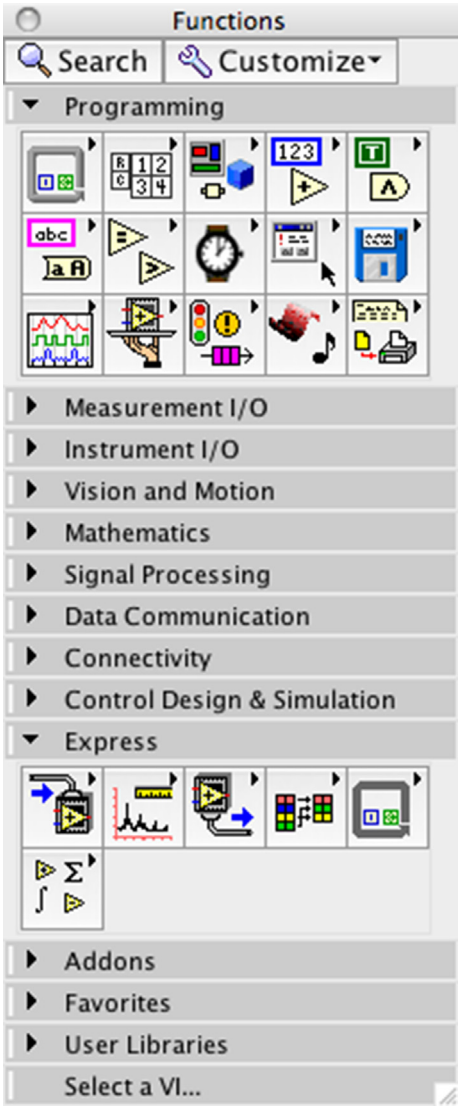

Figure 12.8
The Functions palette.

these elements from the Functions palette. The functions to add, subtract, etc. can be found in the Numeric subpalette of the Programming section of the Functions palette (Figure 12.8, the fourth icon from the left). One can also use constants from the Numeric subpalette into a Block Diagram, and wire those constants as inputs to various nodes.

12.7 Logic Operations in LabVIEW

In more complex VIs, one may encounter situations where the VI must react differently depending on the condition at hand. For example, in a DAQ process, the VI may need to

Figure 12.9
Logic example.

turn on a warning light when an input voltage exceeds a certain threshold and therefore it may be necessary to compare the input voltage with the threshold value. In addition, the VI needs to make a decision based on what the result of the comparison (turn on a light on the screen or external to the DAQ system, produce an alarm sound, etc.). To allow for these types of applications, there are many logic operators available in LabVIEW. These can be found in the Comparison subpalette of the Programming section of the Functions palette. Note that as stated earlier, the Functions palette is available when the Diagram Window is the top (focus) window on the screen. If one needs to identify the Comparison subpalette one can move the mouse pointer over the Programming section of the palette to call out the different subpalettes.

Comparison nodes, as for instance depicted in Figure 12.9, are used to compare two numbers. Their output is either *True* or *False*. This value can be used to make decisions between two numbers as shown in the figure. Another important node in this respect is the *Select* node, which is also available in the same subpalette. This node makes a decision based on the outcome of a *previous* comparison such as depicted in the figure. If the Select node's middle input is True, it selects its top input as its output. If its middle input is False, it selects its bottom input as its output. In this way, one can pair the comparison nodes with a Select node to produce an appropriate action for subsequent processing.

12.8 Loops in LabVIEW

In building more sophisticated VIs, it will not be sufficient to simply perform an operation once. For example, if LabVIEW is being used to the factorial of an integer, n, the program will need to continue to multiply n by $n - 1$, $n - 2$, and so forth. More pertinently, in DAQ processes, one needs to acquire multiple samples of data and process the data. For this reason, LabVIEW includes several types of loop structures. These can be found in the Structures subpalette of the Programming section in the Functions palette. (Note that as with every one of LabVIEW's tools, one can use LabVIEW's help feature or its quick help feature to get more information on these constructs.) Here the user can find a *while* loop, a *for* loop, a *case statement*, and several other important loop structures. Figure 12.10 depicts a simple program that utilizes a loop. There are several important items to note about using loops. Everything contained inside the loop will be repeated until the *ending condition* of the loop is met. For a *while* loop, this will be some type of logical condition.

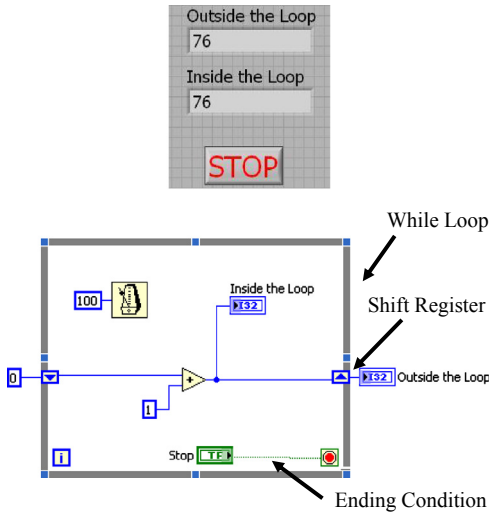

Figure 12.10
Example loop Virtual Instrument.

In our example above, we have used a button to *stop* the loop. The loop will stop when a value of *True* is passed to the stop button at the completion of the loop.

In addition, it is often important to be able to pass values from one iteration to the next. In the example above, we needed to take the number from the last iteration and add one to it. This is accomplished using a *shift register*. To add a shift register, one has to right-click (or option click on a Mac) on the right or left side of the loop and use the menu that opens up to add the shift register. To use the shift register, one must wire the value to be passed to the next iteration to the right side as shown in Figure 12.10. To use the value from the previous iteration, one draws wire from the left side of the loop box to the terminal of one's choosing. In addition, elements initially wired together can be included in a loop simply by creating one around these elements. The wiring initially in place will be preserved. Finally note that the metronome in the loop times the loop so that it runs every 100 ms. This element is available from the Timing subpalette in the Programming Section of the Functions palette.

12.9 Case Structure in LabVIEW

The *case structure* is a programming construct that is useful in emulating a switch, similar to what appears on the Front Panel of a hard instrument to allow the user to select one of the multiple available tasks. In terms of its appearance in LabVIEW, a case structure works much like a while loop as it is evident in Figure 12.11. However, there are significant differences between a case structure and a while loop in that a case structure performs a *separate* operation for each case of the *conditional statement* that drives this

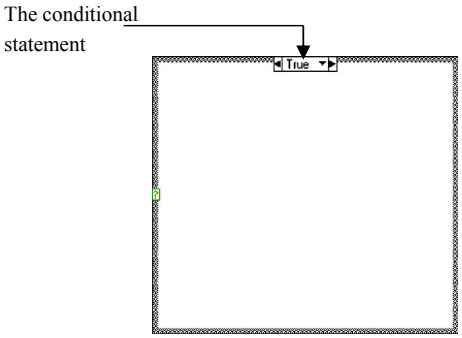

Figure 12.11
A case structure in LabVIEW.

structure. The conditional statement takes a value that is chosen by the user at run-time from among the set of values for which the given case structure is programmed to execute. This set can be {0,1} as is the case initially when a case structure is added to a VI and can be expanded during the programming stage by right-clicking on the conditional statement and choosing "Add case" as necessary. For each case, the VI must include an appropriate execution plan that is placed within the bounding box of the case structure. The conditional statement associated with a case structure is typically driven by a *ring*, which is placed on the Front Panel of the given VI, and which appears outside the bounding box of the case structure in the Block Diagram panel of the VI.

This ring is connected to the question mark box on the right side of the case structure in the Block Diagram. This is illustrated in Figure 12.12 in conjunction with a four-function calculator implemented in LabVIEW. It is evident in the figure that the *ring*, acting as an operation selector, drives the condition statement while the variables *X* and *Y*, implemented via numeric controls on the Front Panel, pass their value to the case structure, which embeds the actual mathematical operation for each of the four functions (Add, Subtract, Multiply, and Divide) in a dedicated panel. The figure depicts the panel associated with the Divide operation. The arrows in the condition statement of the case structure can be used during the programming stage to open each of the cases that the case structure is intended to implement. The ring outside the case structure must have as many elements as there are cases in the respective case structure. Here, it is important to make sure that the order of the selections in the ring and the order of the functions in the case structure correspond to each other. If the first option on the ring is "Add," the first function of the case structure needs to implement the addition function. Exercise 12.6 deals with this issue at more length. Note that as is the case with other LabVIEW elements, right-clicking on a given element allows the user to view a detailed description and examples associated with the given element.

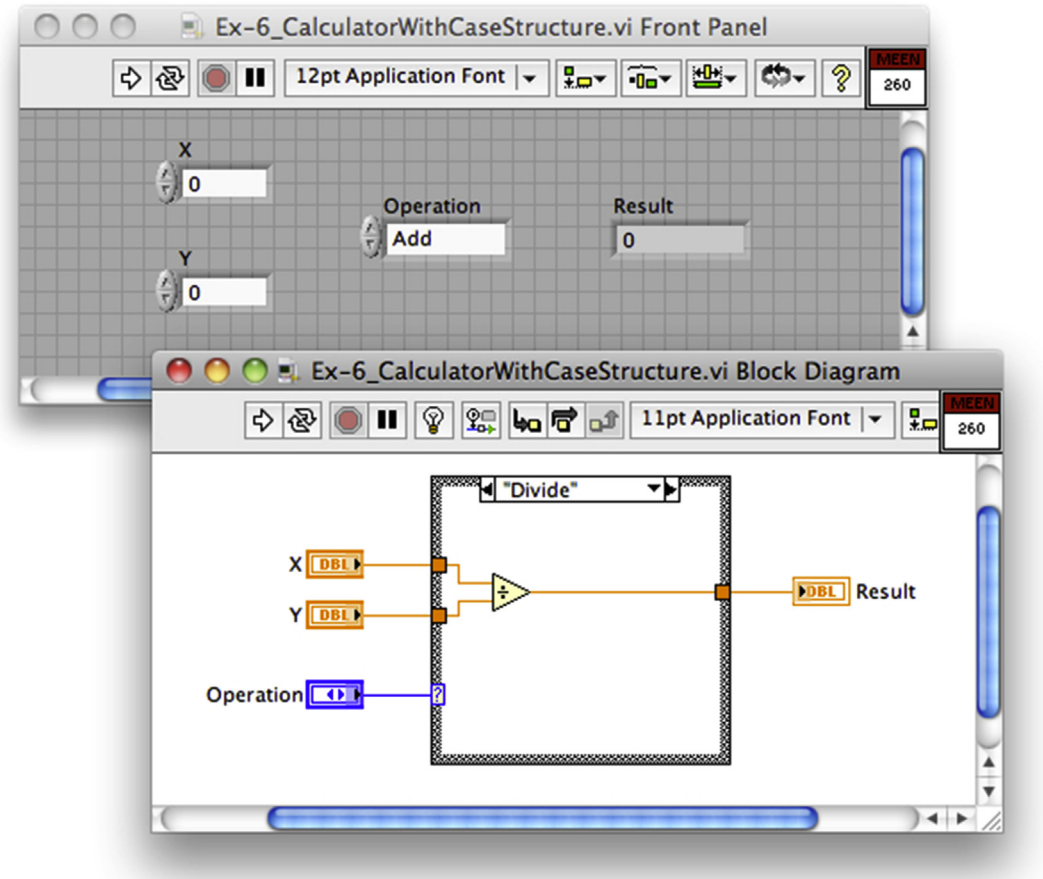

Figure 12.12
A menu ring used in conjunction with a case structure.

12.10 DAQ Using LabVIEW

LabVIEW is primarily a DAQ tool. The typical setup in the lab is shown in Figure 12.13. Aside from the hard instruments shown in the figure, which are also used to perform DAQ and monitoring tasks, a terminal box is needed to obtain the input from sensors and to allow a way to produce output voltages from the DAQ card.

A typical connector block is shown below in Figure 12.14. This is a National Instrument BNC-2120. The pin numbers and the style of the terminal box will vary depending on the model that you may be using. The experiments described later in this chapter use BNC-2120, but several will use different terminals. For the exercise discussed below, the DAQ card's differential mode is used; meaning that it reports the voltage difference across a channel (channel 0 in our case).

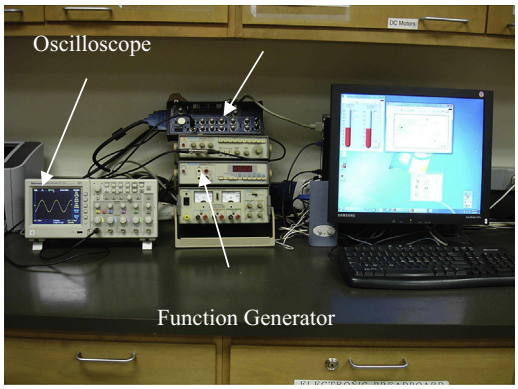

Figure 12.13
Typical setup for LabVIEW-based data acquisition.

Figure 12.14
NI BNC-2120 connector block.

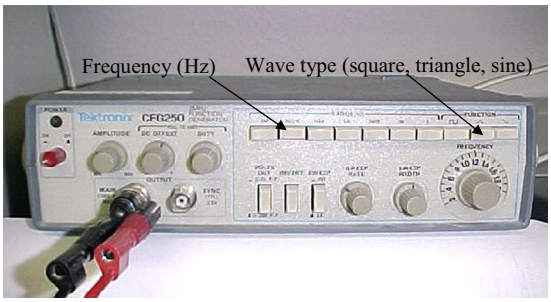

Figure 12.15
Typical function generator.

The card can also operate in an absolute mode where it takes only one input pin and compares this to the ground level. Finally, there is one last piece of equipment, which will be needed in the subsequent discussion: the function generator. This can be seen in Figure 12.13, but a larger picture of a different type of this device is shown in Figure 12.15.

As the name suggests, this piece of equipment is used to generate a sine wave, a triangle wave, or a square wave. The amplitude of the wave can be adjusted by using the amplitude knob. The frequency of the wave can also be adjusted using the relevant buttons. The main use of this device is to produce an external input (a sinusoidal function, a triangular wave, or a square wave) to a custom VI as shown for instance in Figure 12.16, which converts the voltage produced by the function generator into a temperature reading in Celsius and Fahrenheit units.

The functions relating to communication with the DAQ card are handled by the "DAQ Assistant" function as depicted in the figure. The board ID and channel settings specify information about the DAQ card that tells the subprogram where to look for the data. For instance, if the function generator is connected via BNC-2120 (Figure 12.14) to channel 0 on the DAQ card inside the PC, then the DAQ Assistant can be used to set the channel to 0 as well. It is important to check this prior to using the VI. The algebraic operations that implement the conversion of the data are straightforward as depicted in Figure 12.16.

12.10.1 LabVIEW Function Generation

In this section, we will be able to create a VI that emulates a function generator. That is, the VI should produce a periodic signal with a user-selected shape, frequency, and amplitude to an output channel on the National Instruments DAQ card. In addition, the user should be able to select the resolution of the waveform (i.e., the number of data points per period). The VI's Front Panel should also include a waveform graph that displays the selected waveform as it appears in Figure 12.17.

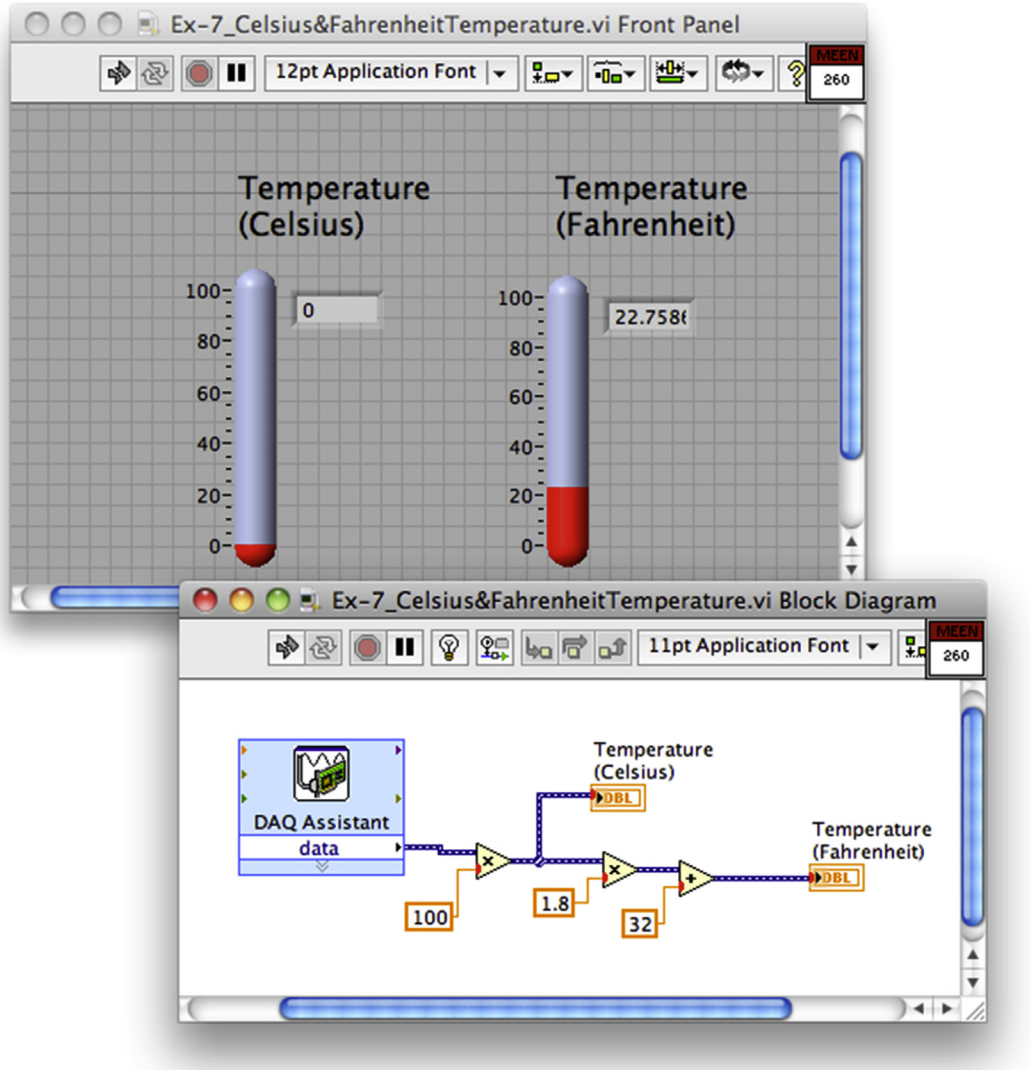

Figure 12.16
Thermometer Virtual Instrument.

The VI produces a sine wave, triangle wave, or square wave as selected by the user. This requires that a case structure to be wrapped around the signal-generation block and a ring block to communicate between the Front Panel and the case structure. This is accomplished as depicted in the Block Diagram for the above function generator in Figure 12.18. Note the use of the case structure, the signal graph, and DAQ Assistant.

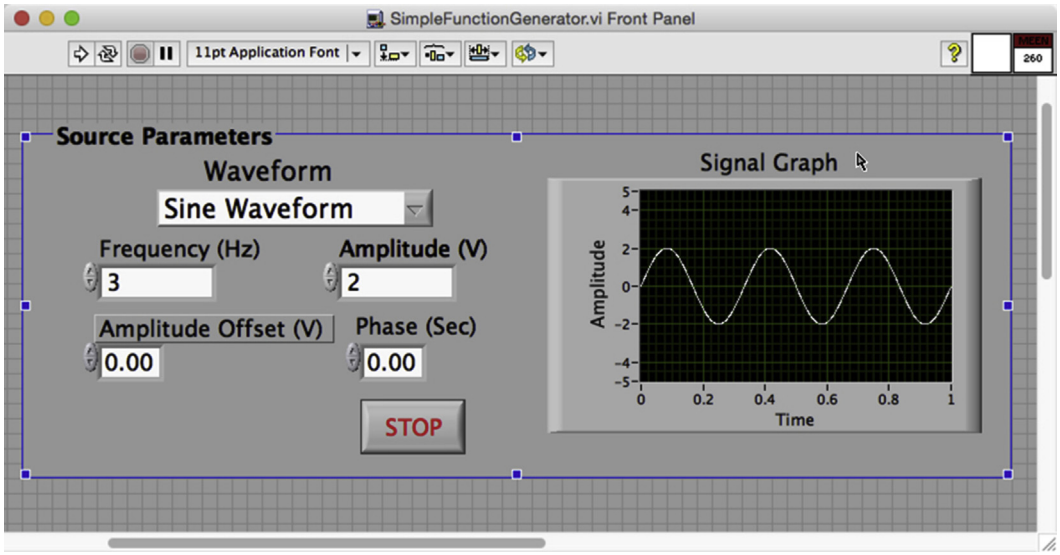

Figure 12.17
Simple function generator Front Panel.

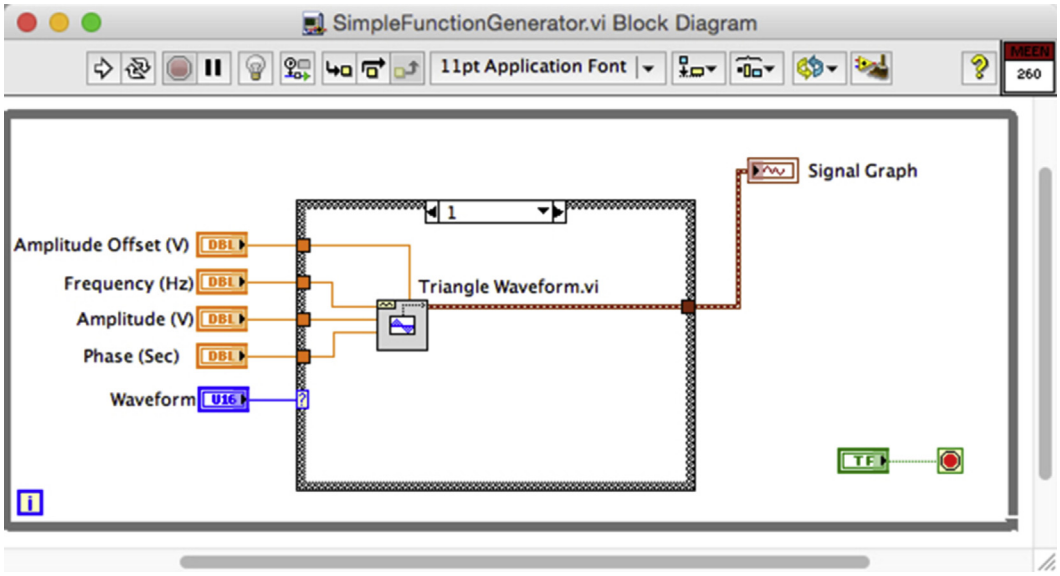

Figure 12.18
Simple function generator Block Diagram.

12.11 LabVIEW Implementation of Digital Filters

One can implement a digital filter in LabVIEW as shown in Figure 12.19. The Block Diagram of this system appears in Figure 12.20. The filter parameters are set to 0.5 and 0.5 in the lower left corner of the Front Panel but can be changed if necessary. The Block Diagram depicts a sinusoidal signal generator as well as a noise generator on the left side.

The in-place element structure allows simple operation such as addition to occur on the corresponding elements of two dynamic arrays produced by the sinusoidal and noise signal generators. This node can be found in the Structures subpalette of the Programming section of the Functions palette as depicted in Figure 12.21.

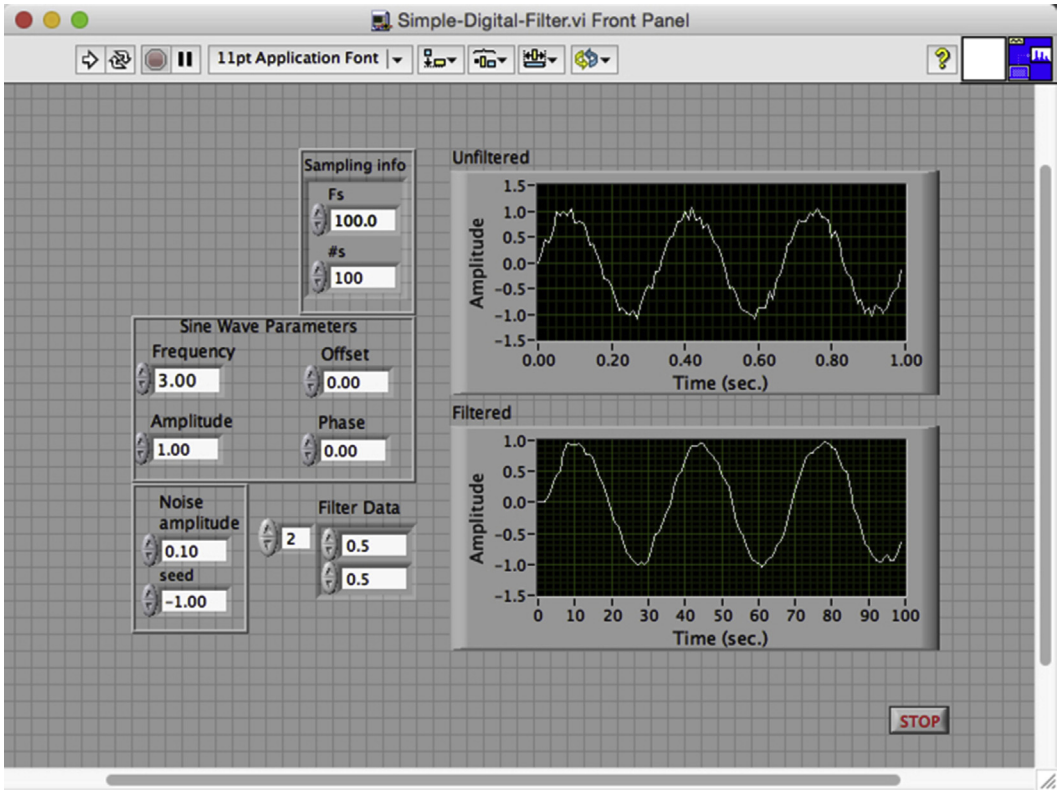

Figure 12.19
Front Panel of a simple digital filter.

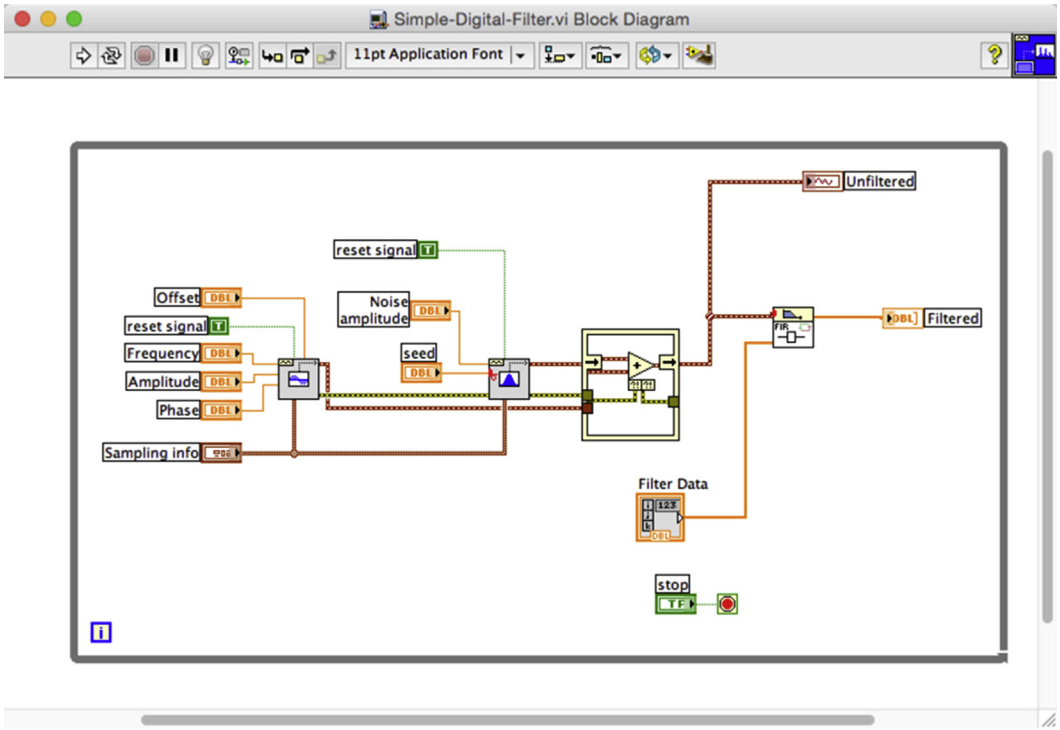

Figure 12.20
Diagram of the simple digital filter.

The filter itself is implemented as a *finite impulse response* (FIR) filter node (found in Advanced FIR Filtering section of the Filters subpalette of the Signal Processing section of the Functions palette[1]), which effectively implements a *moving average* (MA) filter type. The array of filter coefficients appears in the Front Panel of Figure 12.19. Note that in this case the source signal is generated internally but it is possible to do the same task using an external input signal, as for instance, generated by a function generator.

12.11.1 Higher Order Digital Filters in LabVIEW

While the simple filter in the previous section works reasonably well, one can build more effective filters using a combination of autoregressive terms and moving average terms, via

[1] The location of these nodes in the respective palette may change depending on the version of LabVIEW. It is, however, possible to search for a specific type of node by name and locate it irrespective of its location.

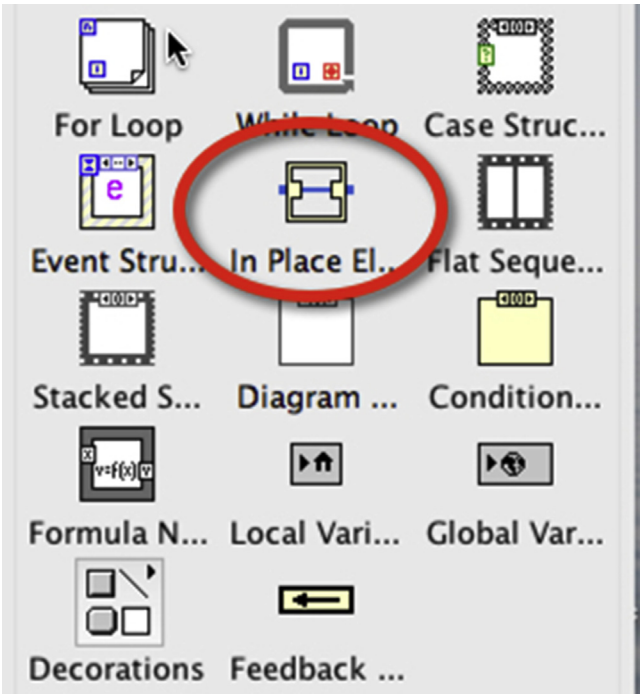

Figure 12.21
Structures subpalette of the Functions palette.

an autoregressive moving average (ARMA) model, also referred to as an infinite impulse response (IIR) filter. The general equation for such a filter is given below:

$$y_k = -a_1 y_{k-1} - a_2 y_{k-2} - a_3 y_{k-3} + \ldots + b_0 u_k + b_1 u_{k-1} + b_2 u_{k-2} + b_3 u_{k-3} + \ldots \quad (12.1)$$

Note that the a_i and b_i must be properly chosen for stable and effective performance. This is not a trivial task and requires advanced techniques that extend beyond the scope of the present text. The essential idea is to place the so-called poles and zeros (the roots of the denominator and numerator of the corresponding discrete time transfer function) in reasonable locations in the complex plane. However, there are well-known design strategies that have performed well, including Butterworth, Chebyshev, and Bessel that are programmed in LabVIEW. It is also possible to produce the filter coefficients in Matlab or a similar tool and use a similar technique as in the previous section (albeit using an IIR filter node) to implement the given filter.

Implementation using LabVIEW built-in functions is depicted in Figure 12.22. The filter parameters include the sampling frequency, which in this case is the same as the

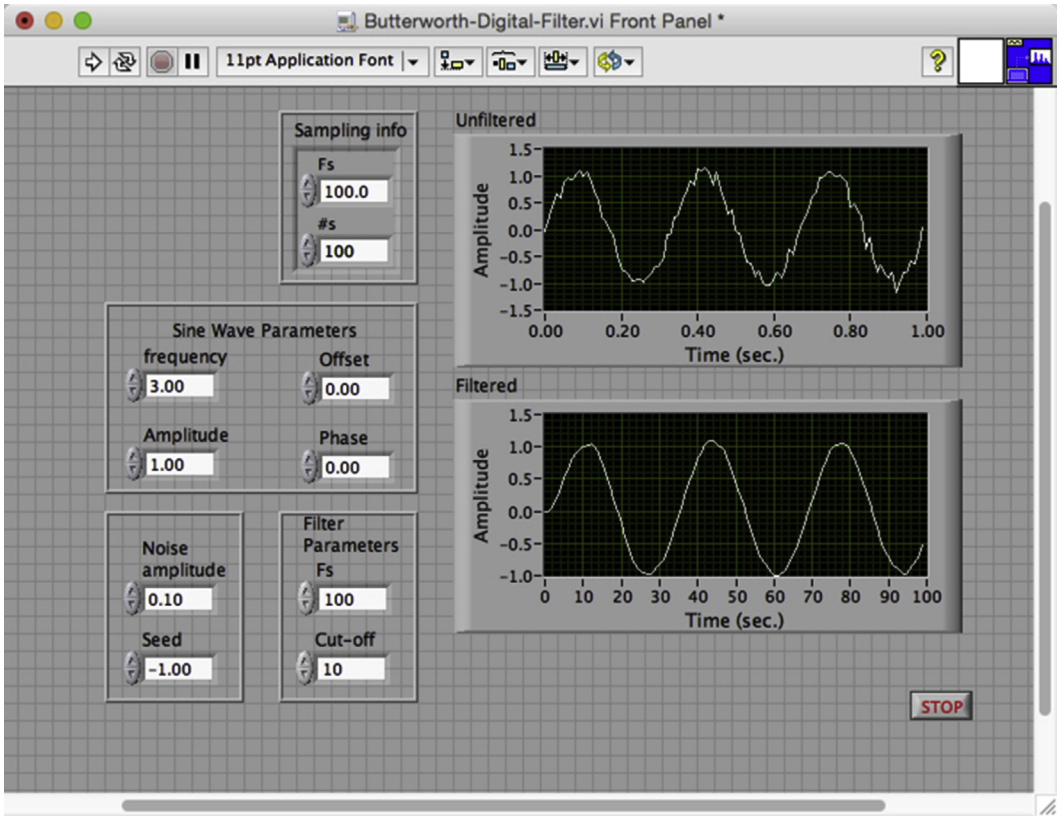

Figure 12.22
Front Panel of Butterworth filter design.

frequency used to generate the signal in the first place. The cut-off frequency is also another required parameter, which is chosen to correspond to the frequency at which the noise components start to dominate the real signal. The Block Diagram is depicted in Figure 12.23. The filter node (from the Signal Processing section of the Function palette) requires filter type (set to zero for a low-pass filter) as well as the two parameters mentioned above. Note that the unconnected terminal of the filter node (the high-frequency cut-off) is not required for a low-pass or high-pass filter but is required for a band-pass filter.

The response of the filter depicted in the Front Panel diagram in Figure 12.22 typifies the low-pass filtering effect of a Butterworth filter, which indeed performs better than the simple filter we designed earlier.

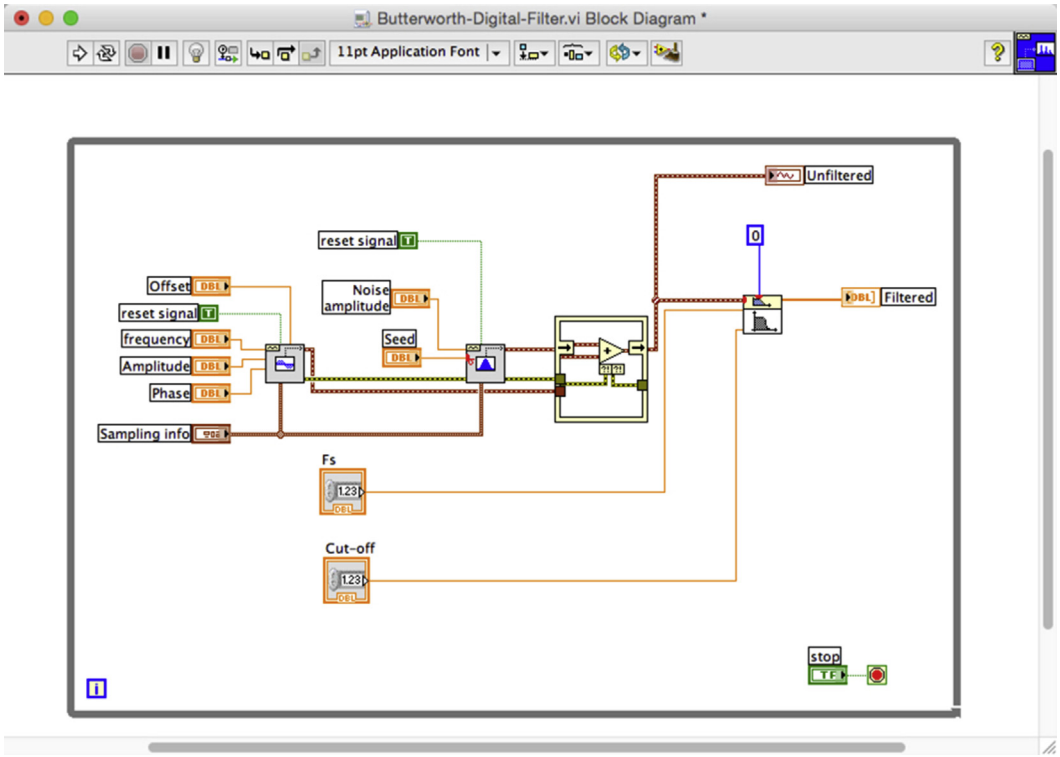

Figure 12.23
Block Diagram of Butterworth filter design.

12.12 Summary

This chapter was meant to introduce the reader to the usage of LabVIEW in computer-based DAQ and signal processing. Basic LabVIEW programming concepts, namely controls, indicators, and simple nodes used to implement algebraic operations as well as program flow control constructs; i.e., while loop and case structure, were introduced. These building blocks were used to implement increasingly more complex LabVIEW VIs. These VIs can be used as stand-alone LabVIEW programs or form sophisticated VIs in conjunction with additional LabVIEW elements. Digital filters are also relatively easily built using LabVIEW and can be used to augment LabVIEW DAQ tools to enable the user to perform meaningful experiments involving mechanical systems. The exercises that follow enable the reader to practice constructing a number of VIs that are of value in laboratory setting.

12.13 Exercises

12.1 In order to demonstrate your understanding of LabVIEW programming, in this exercise you will build the VI depicted previously in Figure 12.4. We assume that

LabVIEW is installed on your computer. Start LabVIEW and select New VI from the File menu. (Note that the start-up screen allows you to create new projects but at present we will focus on creating only a new VI.) The blank VI Front Panel and blank Block Diagram window should appear in the background as for instance are depicted in Figure 12.24. If the Block Diagram window is not visible, choose the Show Block Diagram from the Window menu (Ctrl-E in Windows or Cmd-E on a Mac should also do the same thing).

On the Front Panel diagram, you will need to add the required *Numeric Controls* for X and Y and *Numeric Indicators* for $X + Y$ and $X - Y$. These are available in the Modern subpalette of the Controls palette, which is available by right-clicking in the Front Panel. You will then use the Block Diagram window and implement the *add* and *subtract* nodes (available in the Numeric Section of the Programming subpalette of the Functions palette, which itself can be viewed by right-clicking in the Block Diagram window) and connect them to the appropriate controls and indicators. Note that you can change the labels of each entity by using the Text tool from the Tools palette and further note that changing the label of an indicator or control in the

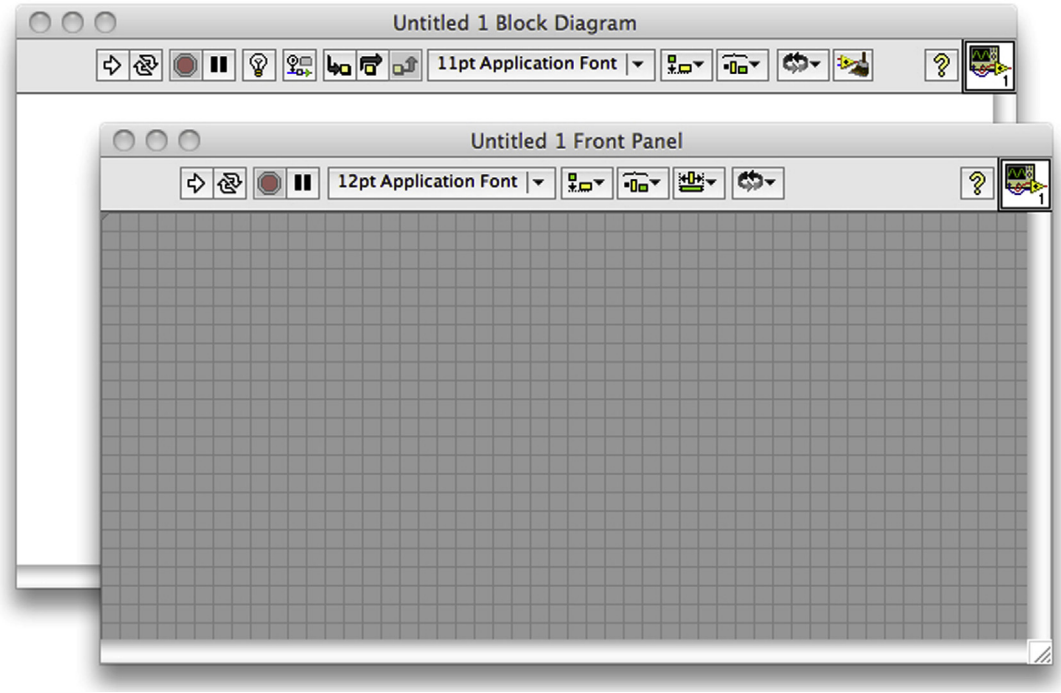

Figure 12.24
Blank Virtual Instrument panels.

Block Diagram window changes its label in the Front Panel as well. Once you have inserted the necessary nodes and wired them together properly, you can run the VI by pressing the Run button  at the upper left corner of the Front Panel window. The digital indicators should then display the results of the addition and subtraction operations. You can also use Continuous Run button  , which is next to the Run button.

12.2 Create a program that will take the slope of a line passing between any two points. In other words, given two points (X_1,Y_1) and (X_2,Y_2), find the slope of the line, $Y = mX + b$, which passes through these points. This program should have four digital inputs (as Numeric Controls) that allow the user to select the two coordinates. It should have one Numeric Indicator to provide the value of the slope of the line between them.

12.3 Using ideas similar to those in Figure 12.9, demonstrate your understanding of the logic operations by adding an additional Numeric Indicator that will display the *greater* of the two numbers determined from your addition and subtraction operations in Exercise 12.1. Is the number obtained by the addition operation always the greatest?

12.4 Demonstrate your in-depth understanding of *logic* operations by building a VI that takes as input three numbers and outputs the *greatest* and the *smallest* value of the three. This will involve very similar logic operations as the one performed in the earlier VI but requires cascading these operations and generally a more sophisticated use of the LabVIEW logic operations.

12.5 In order to drill you in the operation of loops and related structures in LabVIEW, practice by creating the VI depicted in Figure 12.10. When fully operational, the program should continue to increment by one every 0.1 s until the stop button is pushed. Verify that this is true and that waiting longer to press the stop button results in a larger number. What is the difference between the *inside* and *outside* loop displays? Why do you think this happens? If you let the program run and failed to turn it off, would your program ever be able to reach *infinity*? Explain your answer.

12.6 The specific task that you will perform is to create a calculator that will *add*, *subtract*, *multiply*, or *divide* two numbers using a case structure depicted in Figure 12.12. This will require that you to use two inputs, an output, and a *ring* on the Front Panel. You will need a *case structure* in the Block Diagram window, which initially will have only two cases {0,1} but can be extended by right-clicking on the Condition Statement in the Block Diagram. The case structure itself can be found in the Structures subpalette of the Functions palette. The ring should be added to the Front Panel of the VI and can be found on the Modern subpalette of the Controls palette. The properties dialog box of the ring, which opens up by right-clicking on the ring, allows you to add cases as you need (in this case, four cases).

Make sure the order of the list on the ring corresponds to the same subset in the case list and that you set a *default* case.

12.7 Data acquisition (DAQ) is likely the most important function of LabVIEW. To be able to understand completely what takes place during DAQ, it is important to know what tasks are performed by a VI that acquires data from an external source. The VI in Figure 12.16 can take a temperature measurement every time the run button is pushed. While this is useful for taking measurements where temperature is constant (e.g., the room temperature), it is not useful in tracking temperature changes. (The next exercise addresses this issue via a *loop* structure.) If a low frequency square wave is used, however, it is possible to verify that the VI functions as intended. Assuming you have access to a physical function generator instrument and that your computer is equipped with a LabVIEW compatible DAQ card, connect the function generator to the computer. Make sure that the function generator is set on a square wave and that the frequency is set to be about 0.1 Hz to create a signal that changes infrequently. You may wish to view the signal on an oscilloscope to ensure that the function generator is indeed producing the desired waveform. This can typically be done by connecting the output of the function generator via a BNC connector to Channel A (or Channel 1) on a typical two-channel scope. Be sure to set the trigger level of the oscilloscope to External (EXT) and choose Channel A (or Channel 1) as the trigger source. You may have to initially use the Ground function of the scope to ensure that the beam is zeroed at the mid-level of the screen and that the vertical and horizontal scales are properly set up. The same BNC connector can be used to connect the function generator to the BNC-2120, which acts as the interface between the DAQ card and the function generator (assuming that this is the device that enables you to physically connect a signal source to your computer). The elements of the VI itself are readily obtainable from the Controls and Functions palettes. For instance, the Thermometer Numeric Indicator 

is useful in this context as is the DAQ Assistant from the Measurement I/O subsection of the Functions palette. Once the VI is constructed, it is important to ensure that the settings of the DAQ channel are consistent with the actual channel used in the DAQ process.

12.8 In this exercise we need to add a loop to the VI in the previous exercise so that the VI can take readings continuously until a Stop button on the Front Panel of the VI is pushed and the VI is stopped. The Loop construct is in the Structure subpalette of the Programming section of the Functions palette. You can also find the Stop button in the Classic Boolean subpalette of the Classic section of the Controls palette. Verify that this works and watch how temperature indicator changes with time. Change the frequency of the function generator and observe the effects on the output. Increase the frequency of the signal using the frequency control knob on the function generator instrument and observe the effect on the reading produced by

the thermometer indicator. At what point do you not observe the variation of the external signal?

12.9 In this exercise, we will add more functions to the VI from the previous exercise. If we only want to maintain a given temperature (as for instance, in a thermostat), we might only care about the highest value of the temperature. Use the *logic* operators, used in the previous exercises, to determine the *highest* temperature among all the measurements (Hint: you will need a Shift Register). In LabVIEW there are many different options for data types (Scalar, Dynamic Data, Waveform, etc.). For this part of the experiment you will need to convert your acquired data from Dynamic Data type to Single Scalar type. To do this, right-click in the Block Diagram select Express, Signal Manipulation, From DDT, which appears as  in icon form. The respective dialog box enables you to accomplish the required task.

12.10 This exercise focuses on building a digital filter using LabVIEW built-in filtering functions. Specifically, we will build the VI depicted in Figure 12.25, which is

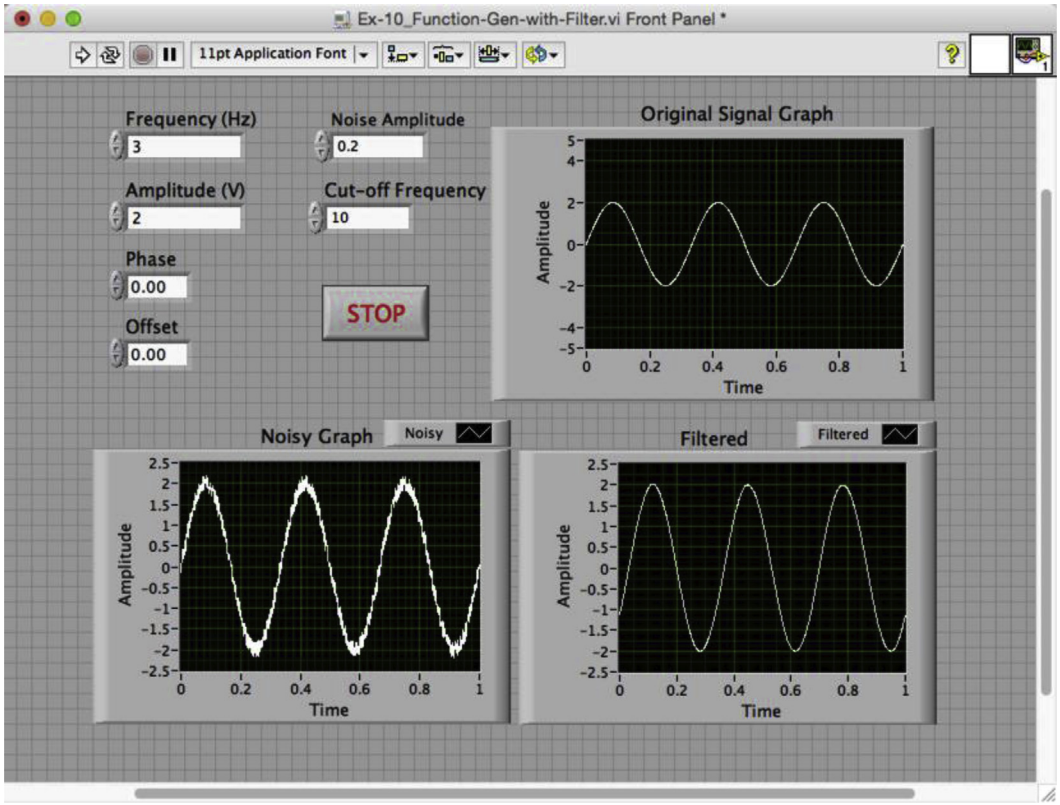

Figure 12.25
Function generator and filter Front Panel.

used to generate a sinusoidal function and after adding noise, filtering the signal, and displaying the filtered signal alongside the original signal. Note that in spite of the scaling factors, the filtered and original signals are of the same range in magnitude. The figure shows the noise amplitude to be about 10% of the signal amplitude but this can be changed as necessary. The filter cut-off signal can also be changed at will.

Since the design is rather complicated, the Block Diagram is shown in Figure 12.26. Note that each of the elements used in the Block Diagram can be obtained from the subpalettes of the Function palette although some require a bit of digging in the various subpalettes, which also helps familiarize you with the various functionalities of LabVIEW.

12.11 The VI in the above exercise can be used to generate external signals that can also be displayed using an oscilloscope. The Front Panel would look exactly the same as in Figure 12.25. However, the Block Diagram would need to be modified as shown in Figure 12.27 to incorporate a DAQ Assistant.

12.12 Implement the VI in Figures 12.19 and 12.20. The steps are clear from the Block Diagram that is shown.

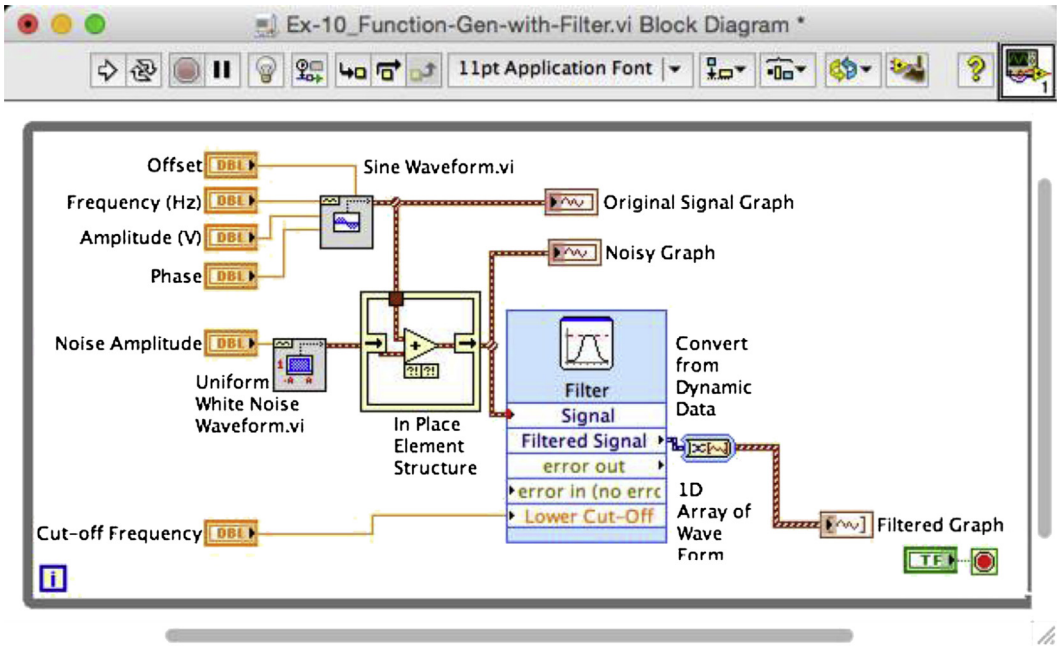

Figure 12.26
Function generator and filter Block Diagram.

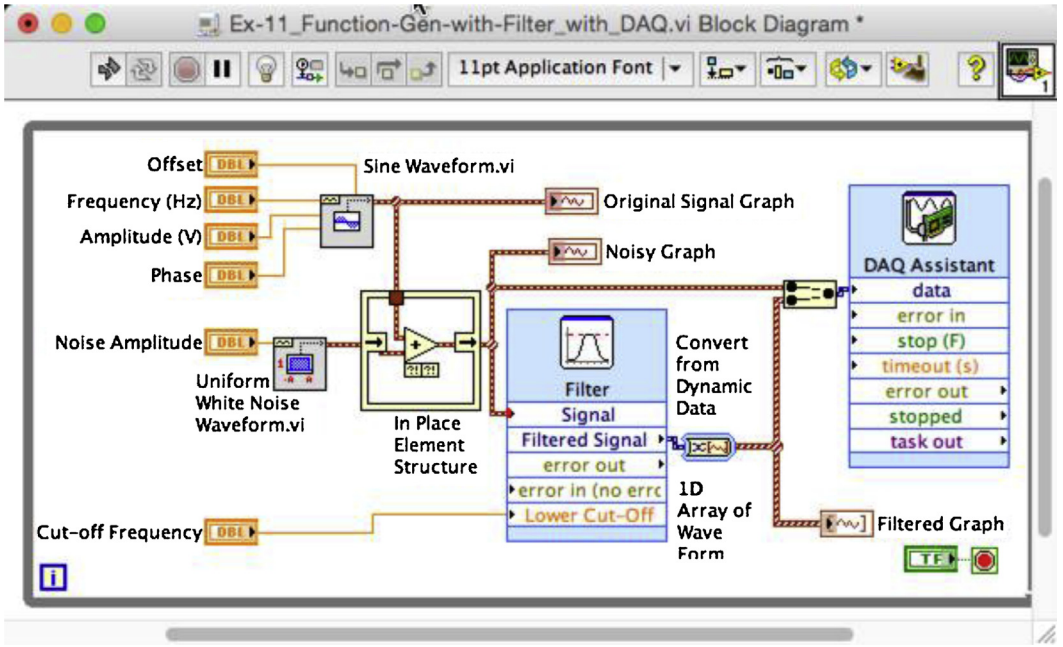

Figure 12.27
Function generator and filter Block Diagram.

12.13 Implement the VI in Figures 12.22 and 12.23. The steps are clear from the Block Diagram that is shown.

12.14 What are the potential problems of using filters in real systems? Hint: In the earlier exercises, we knew *exactly* what we were looking for. What if we do not?