Introduction to LabView, Version 7

Section 1: Creating a VI

There are two main parts of a VI, the front panel & the block diagram: user interface & the source code.

From the front panel, you can access any object thorough the control palette, which allows you to access custom controls & indicators that you have created.

When an object is selected from the Controls palette and placed on the front panel, a terminal is placed on the block diagram. The terminal is the representation of the front panel object in source code.

An Example:

1- Select a Thermometer indicator from **Controls/Numerical Indicators (Num Inds)**, and place it on the front panel.

2- Double click on its label and type "Temp" inside the label text box and click on the "Enter" or anywhere outside the box.

3- **Go to the block diagram. From the Functions/All Functions** choose **"Select a VI"** icon**.** It opens a window; go to

C:\Program Files\National Insturments\LabView7.0\vi.lib directory and choose **tutorial.llb**, then inside the tutorial library window choose the "**Demo Voltage Read.vi"** and place it on the block diagram. For this subVI, we need a channel ID (a constant numeric) and a device number.

For the channel ID first create a constant by numeric and show that it doesn't work, then use the wiring tool and click on the right button and choose create constant. It needs string.

4- Use the **Functions/Arithmetic & Comparison/Numeric** to choose a multiplier and also a constant. Type 100.0 in the constant box. Then use the wiring tool to connect the constant and the output of the voltage reader. Connect the output of the multiplier to the indicator, which you labeled as Temp before. This gives the temperature in ^{o}F.

5- Run the VI by clicking the on the RUN button in front panel. You may watch the flow of execution by clicking on the light button in block diagram and then running the VI.

6- You may write a description for your VI by clicking on *"Tools"* from the menu bar and choose *"Vi Revision History"* and then type the description of the VI in the "*Comment"* dialog box.

7- Save your VI by clicking on the File from the menu bar and then click on save as. Choose a directory with your name and then click on *"create new library"* to create a library in your folder. Name it as "my library" and then in the save dialog, name your VI as *"my Thermomether.vi"*.

Section 2: Creating a SubVI

A subVI is analogous to a subroutine in any other programming language. You can call a subVI inside another subVI or a VI. Now, we use the VI that you created in previous section and make it as a subVI. To use a VI as a subVI, you must create an icon to represent it on the block diagram of another VI and a connector pane to which you can connect inputs and outputs. In order to do that:

1. Open the "*my thermometer"* VI by clicking on **File/Open** from menu bar.
2. Click the right button on the icon of the VI on the top right corner of the front panel and choose *"edit icon"*. The edit dialog opens and then erase the drawing and draw your own design or write the name of your subVI, then click "ok" to close it.
3. In front panel, again click the right button on the icon and this time choose "show connector" option. Labview selects a suitable terminal pattern according to the number of controls and indicators on the front panel. This example has only one indicator and therefore, there is one terminal.
4. Assign the terminal to the thermometer indicator by clicking on the terminal first (the terminal turns black), then click on the thermometer indicator (a moving dashed line frames the indicator). Now, click anywhere outside the indicator and the dashed line disappears. If the terminal remains white, it means that you haven't made the connection correctly. Repeat this step if necessary.
5. Save and then close the VI.

Using a VI as SubVI

1. Open a new front panel, choose the thermometer from the **Controls/Numerical Indicators (Num Inds)**, and label it "Temp in deg C". Change the range of it to be between 20 and 40.
2. In the block diagram window, click on **Functions/All Functions/Select a VI** and then find and select the *"my thermometer.vi"*, the VI that you created in previous section. The icon of your VI appears on your block diagram.
3. Use the right button of the mouse to click on the block diagram window and choose subtraction, multiplication and division tools from Function/Numeric. Also create 3 constants: 32, 9 and 5. Use the formula of C=(F-32)*5/9, to convert a degree in F to a degree in C. Use the wiring tool to properly connect these controls and constants to the indicator "Temp in deg C".
4. Run the VI a few times. In block diagram window, click on the bulb and then run the VI and see the flow of the process.
5. Save this VI in the same folder and library as before.

Section 3: Loops, Charts

Objective: Using a while loop to display data in real time

1. Open a new front panel.
2. Place a waveform chart **Controls/Graph Inds** in the front panel. Label the chart as the Random Signal. Change the vertical axis of the chart from 0 to 1.
3. Place a knob (**Controls/Num Ctrls/knob**) in the front panel. Label the knob as Loop Delay (sec). With this knob you can control the timing of the while loop. Pop up on the knob and select (deselect) **Visible Items/Digital Display** to show (hide) the digital display that is hidden by default.
4. Using the labeling tool, change the scale around the knob to show between 0 and 2.
5. Open the block diagram.
6. Place the While loop by selecting it from **Functions/Exec Ctrl/While Loop**. Drag the While loop to encompasses the terminals. A while loop has a conditional terminal and an iteration terminal, which always starts counting the loops at zero. The while

loop keeps executing until the conditional terminal terminates the execution. In LabView version 7, when you select a while loop, by default a stop button appears on the front panel, which is the condition for execution of the loop, and it is also wired to the conditional terminal of the loop. In this example, wire the "stop" terminal of the loop to the conditional terminal of the while loop (the red stop sign).

7. Select the Random Number function from the **Functions/Arithmetic & Comparison/Numeric** and wire it to the chart in the block diagram. Leave the loop delay (the knob) terminal unwired for now. You can use the moving tool to move the blocks to your desired locations inside the loop.
8. Run the VI. Click on the stop button to stop the loop. To run again, first pop up on the chart and choose **Data Operations/Clear Chart** and then run again.
9. By default, the Stop terminal of while loop is a "Control" button, which means it does accept input from the front panel. In order to understand the difference between a "Control" and an "Indicator" terminal, on the front panel, pop up the window of the "stop" button and select "change to indicator" and see what happens. Your VI shows a broken arrow for the run and gives you error because the stop terminal has become an indicator that cannot function any more. Change it back to "Control".
10. Now in block diagram select the "stop" terminal of the while loop and remove it. Then in front panel, place a vertical switch (**Rocker**) **Controls/Buttons & Switches** in the front panel. Then in block diagram wire this switch to the conditional terminal of the while loop.
11. Use the labeling tool to create the free label for On and Off. Use the color tool (from the *Window* menu bar choose *"Show Tools Palette*" and then choose coloring tool), to make the free label border transparent.
12. Repeat step 8.

Mechanical Action of Boolean Switches

You might have noticed that each time you run the VI, you first must turn on the vertical switch and then click on the run button, in the toolbar. You can modify the mechanical action of Boolean controls. Click the right button of the mouse when pointing to the Boolean switch and observe the six available mechanical actions:

- **Switch When Pressed** action changes the control value each time you click on the control with the Operating tool. The action is similar to that of a ceiling light switch and is not affected by how often the VI reads the control.
- **Switch When Released** action changes the control value only after you release the mouse button, during a mouse click, within the graphical boundary of the control. The action is not affected by how often the VI reads the control. This action is similar to what happens when you click on the check mark in a dialog box; it becomes highlighted but doesn't change until you release the mouse button.
- **Switch Until Released** action changes the control value when you click on the control. It retains the new value until you release the mouse button, at which time the control reverts to its original value. The action is similar to that of a doorbell, and is not affected by how often the VI reads the control.
- **Latch When Pressed** action changes the control value when you click on the control. It retains the new value until the VI reads it once, at which point the control reverts to its default value. (This action happens whether or not you continue to press the mouse button.) This action is similar to that of a circuit breaker and is useful for stopping While Loops or having the VI do something only once each time you se the control.
- **Latch When Released** action changes the control value only after you release the mouse button. When your VI reads the value once, the control reverts to the old value. This action guarantees at least one new value. As with **Switch When Released**, this action is similar to the behavior of buttons in a dialog box; clicking on this action highlights the button, and releasing the mouse button latches a reading.
- **Latch Until Released** action changes the control value when you click on the control. It retains the value until your VI reads the value once or until you release the mouse button, depending on which one occurs last.

Try these mechanical actions with the vertical Boolean switch in your VI and see their differences.

Adding Timing

When you ran the VI, the While Loop executed as quickly as possible. However, you may want to take data at certain intervals. For this purpose you can use LabVIEW's timing functions. Modify your VI as the following:

1. In the block diagram, insert **Wait Until Next ms Multiple** function from **Functions/All Functions/Time & Dialog**. In windows 95/NT the resolution of this function is 1 ms.
2. Insert a Multiply and a Constant from **Functions/Arithmetic & Comparison/Numeric**. Type 1000 in the constant terminal. Wire the Loop Delay terminal and the constant to the Multiply function and their output to the **Wait Until Next ms Multiple** function.
3. Rotate the knob to get different values for the number of seconds of delay and run the VI a few times.
4. Save and close your VI. Name your VI as "Plot Random Signal.vi".

Exercise

Modify your "Plot Random Signal" VI such that it also plots a 4 point running average of the random signal in the same chart. For doing this exercise you need to use two new elements: Bundle and shift register.

Section 4: Arrays, Clusters, and Graphs

Objective: Learning about arrays, how to generate arrays on loop boundaries, what's a cluster and how to use graphs to display data.

The difference between a graph and a chart is that a graph plots data as a block whereas a chart plots data point by point or array by array. You may see examples of graph VIs at examples\general\graphs.

1. On a new front panel, place an array shell from **All Controls/Controls/Array & cluster** in the front panel. Label the array constant as Waveform Array.
2. Place a Numeric Digital Indicator from **Controls/Num Inds** inside the element display of the array constant. This indicator displays the array contents.
3. Place a waveform graph from **Controls/Graph Indc** in the front panel. Label the graph as Waveform Graph. A Graph indicator is a 2D display of one or more data arrays. By default graphs auto scale their input. You can disable or enable autoscaling X or Y by popping up on the graph and select or deselect **Y Scale/Autoscale Y**.
4. Go to block diagram and place a For loop (from **Functions/All Functions/Structures/For**) outside the items on the block diagram. Click on **N** and create a constant 100 (the For loop indexes from 0 to 99). Use necessary tools to multiply the index **i** by 2π, then divide by 10, and then wire the output to a Sine function (from **Numeric/Trigonometric/Sine**). Wire the output of Sine function to the Waveform Graph. Note that as the wire leaves the For loop, it becomes thicker to indicate the array. Also wire the sine wave to the Waveform Array.
5. Run the vi and notice that it plots a 100-point sine wave. Watch the values on the Waveform Array indicator. Save your vi under the name "sig_gen.vi".
6. Now, let's assume we want to scale the X axis from 0 to 1 instead of the number of data samples. For doing that, place a Bundle (from **Functions/All Functions/Cluster/Bundle**) and resize it to show 3 input elements (by popping up the right click window and choosing “adding an element”). Bundle function is analogous to Sturct in C programming. Use wiring tool to create 0 and 1 constants for the initial value x_0 and delta x for the X axis. Remove the wire connecting the output of sine function to the graph and rewire it to the 3rd element of the Bundle and then wire the

output of Bundle to the Graph. Note the change of color in the Graph icon. Run your vi and notice that it now scales the X axis from 0 to 1. Change the delta x to 0.05 or any other number and note the difference.

Multiple Graphs

8- Go to the block diagram. From the **Functions/All Functions** choose **"Select a VI"** icon**.** It opens a window; go to **C:\Program Files\National Insturments\LabView7.0\activity** and select **"Generate Wavform.vi".** Modify your VI by placing the Waveform Generator and wire the index **i** to its input. Remove the connection from the Sine output to the Bundle and place a Build Array (from **Array/Build Array**) and resize it to show 2 elements. Connect the output of the Wave and Sine functions to the elements of Build Array. Wire the output of Build Array to the 3rd element of the Bundle and Run your VI again.

7. In the front panel, resize the legend box of the Graph to show both legends. Right click on the legend box and choose different colors for each plot. Use labeling tool to change the name of each plot in the legend box.
8. Save your VI.

Exercise

Instead of generating a sine wave the way you just did in this section, use the Signal Generator function (from Functions/All Functions/**Analyze/Waveform Generation/Basic Function Generator)** outside the For loop to generate the same sine wave and still bundle it with the wave generator and plot the data. Use Show Help to guide you through wiring the terminals of this function.

Section 5: Auto-Indexing on Input Arrays, Cases, Sequences

Attention: You can learn a lot by looking at the many examples of LabView located in C:\Program Files\National Instrument\LabView\Examples

Objective: Create a sine wave as an input array and use auto-indexing to separate the positive and negative values of the input and build half-wave and full-wave rectifiers.

1. Open a new VI and place three arrays with digital indicator inside. Label them as Input Array, Positive Array and Negative Array. (An input Array should have a digital control as its element, however since we do not have a data input, we create a sine wave as input and therefore the input array must have digital indicator as its elements.)
2. Go to block diagram. Place a For loop (from **Structure/For**). Wire a constant of 100 to N. Inside the For loop, multiply the **i** by 2π, then divide by 20 and wire the output to the sine function. This way the frequency of the resulted signal is 5 Hz. Wire the output of the sine function to the Input Array indicator outside the For loop. (Note that the wire becomes thick when leaving the loop to indicate that it is an array.)

In order to rectify the sine wave, you have to separate the negative and positive values and move them into the two Positive and Negative Arrays. Therefore, you need to check every element of the input array.

3. Place another For loop and place a Case structure inside the For loop.
4. Place a Greater? from **Functions/Arithm-Compare/Comparison/Greator?** inside the For loop but outside the Case. Place a Build Array with two elements inside the Case in both True and False cases. You can toggle between the cases by popping up the top of the Case and choose Show Case True or Show Case False.
5. Wire the Input Array to the Greator? function and compare it with zero (click on the other input pin and choose create constant). Wire the output of the Greator? function to the **?** of the Case.
6. Add two shift registers for previous values of the Input Array signal. The shift registers should be initialized, otherwise, your program runs correctly only for the

first time. Initialize them by placing the Initialize Array (from **All Functions/Array**). Use the right button click and choose create constant for input pins of this function (Element Data and Dimension size).

7. Wire the Input Array to the second element of the Build Array inside the Case (in both True and False cases). In Case True connect one of the shift registers of the left side to the first element of the Build Array and wire the output to the current value of the same shift register on the right side of the loop and wire it outside to the Positive Array indicator. In Case False, do the same but with another shift register. In both cases pop up the first element of the Build Array and choose "Change to Array".
8. By now, you notice that the place that the wire leaves the Case (called the tunnel), is white and the run button is broken. Every tunnel has to be connected in both cases. Therefore, in Case True, connect the previous negative values to the current negative values and vice versa in Case False. By doing so, you notice that the tunnels turn black.
9. Place a Build Array (with 2 elements) outside the For loop and connect the Positive and Negative Arrays to the elements of Build Array to create a 2-dimensional data. Wire this data to the Graph and run the program.
10. Save your VI and call it half rectifier.vi.
11. Change the constant 20 in the first generating sine wave to 10 or 5 and observe the results. How can you explain the differences?

Exercise

Modify your half-wave rectifier to become a full-wave rectifier. For doing this, you need only one shift register.

Section 5: Data Acquisition

If you want to write a data acquisition VI, you should read almost the entire Data Acquisition Tutorial book of LabView and start modifying or building your own VI around one of the written examples the LabView. However, in this note, we briefly point out the main components of a data acquisition VI for the most common signal recordings.

- **AI Config VI-** This VI (**Function/Data Acquisition/Analog Input/AI Config)** is the primary component of a data acquisition VI, which configures the channels, selectes the input limits and generates a **taskID**. This VI is needed to be called only once to configure the channels and therefore it should be placed outside the loop for data acquisition.
- **AI Start-** This VI **(Function/Data Acquisition/Analog Input/AI Start)** starts a buffered analog input operation. It sets the scan rate (sampling rate) and trigger condition and then starts an acquisition. This VI can also be set for a continuous acquisition. This VI should also be placed outside the loop for data acquisition.
- **AI Read-** This VI **(Function/Data Acquisition/Analog Input/AI Read)** reads data from a buffered data acquisition. It should be placed inside the loop of data acquisition.
- **AI Clear-** This VI **(Function/Data Acquisition/Analog Input/AI Clear)** clears the analog input task associated with the **TaskID.**

Buffered versus Continuous Acquisition

Designing a buffered data acquisition VI is simpler than continuous one because you just have to specify one buffer with a size of at least equal to the number of channels multiplied by the scan rate and the duration that you need the data in seconds. However, obviously it is limited because of the limitation of the buffer size depending on the computer in use. It is good only for short data recording. If you need longer recording without being worried about the buffer size, you have to design your data acquisition VI in continuous mode. In this case, you have to use a circular buffer.

A circular buffer is filled with data, just as a simple buffer; however, when it gets to the end of the buffer, it returns to the beginning and fills up the same buffer again. This means data can read continuously into computer memory, but only a defined amount of memory can be used. Your VI must retrieve data in blocks, from one location in the buffer, while the data enters the circular buffer at a different location, so that unread data is not overwritten by newer data.