ECE 463 Lab 1: Introduction to LabVIEW

1. Introduction

The purpose of the lab session of ECE463 is to apply/practice the digital communication theory on software-defined radios (USRPs). USRP is coupled with LabVIEW software, a graphical programming language that uses blocks and wires instead of lines of texts. Although the course goal is not to specifically train students to be efficient programmers in LabVIEW, it is necessary that all students become comfortable with the semantics of LabVIEW and become capable of writing some of their own code.

1.1. Lab goals

- Become familiar with LabVIEW Communication Suite.
- Use While/For loops to process data.
- Work with various data types.
- Use arrays and access/process element-wise.
- Generate waveforms using MathScript.

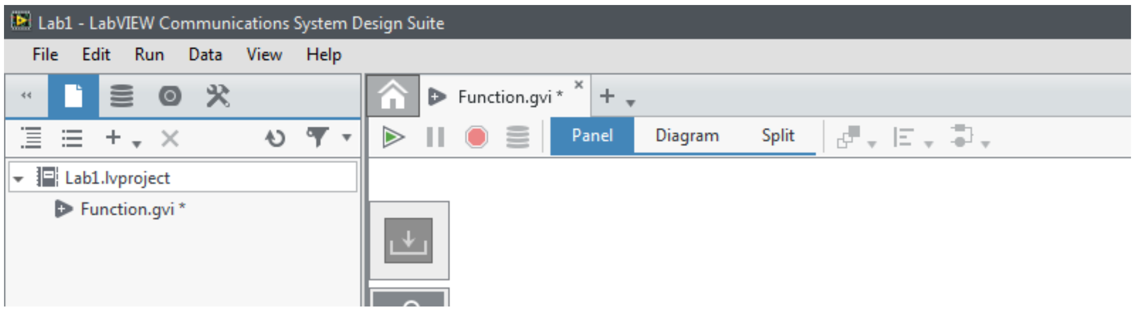
1.2. Contents

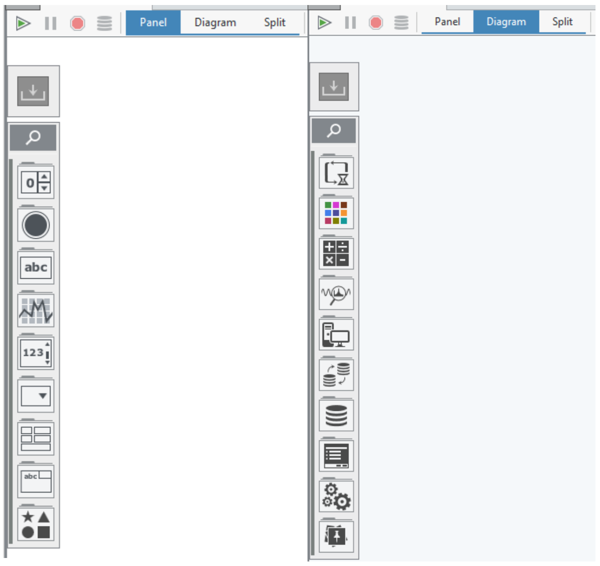
1.3. Report

Submit the answers, figures and the discussions on all the questions. The report is due as a hard copy at the beginning of the next lab.

2. Getting Started

1. Find a LabVIEW Communications 2.0 shortcut in your Desktop and run it.
2. Open a new project by clicking File→New→Project. It will open an untitled project.
3. Add a new VI by clicking File→New→VI.
4. Rename the project name as **Lab1** by right-clicking the project in the left window and clicking Rename. The windows will be similar to the following figure.





5. Each VI has two windows; Panel and Diagram. The Panel window is where the user interacts with the VI. It can contain as many controls (inputs) or indicators (outputs). The Diagram window contains the actual code (flowchart) that receives the input data from the controls, processes them by wires and blocks, and then displays the results through the indicators.

6. There are two palettes: the Controls palettes in Panel window and the Functions palettes in Diagram window. Both the controls and the indicators are available in the Control palettes. The Functions palettes contains the blocks and the functions used to create a flowchart.

3. While Loop

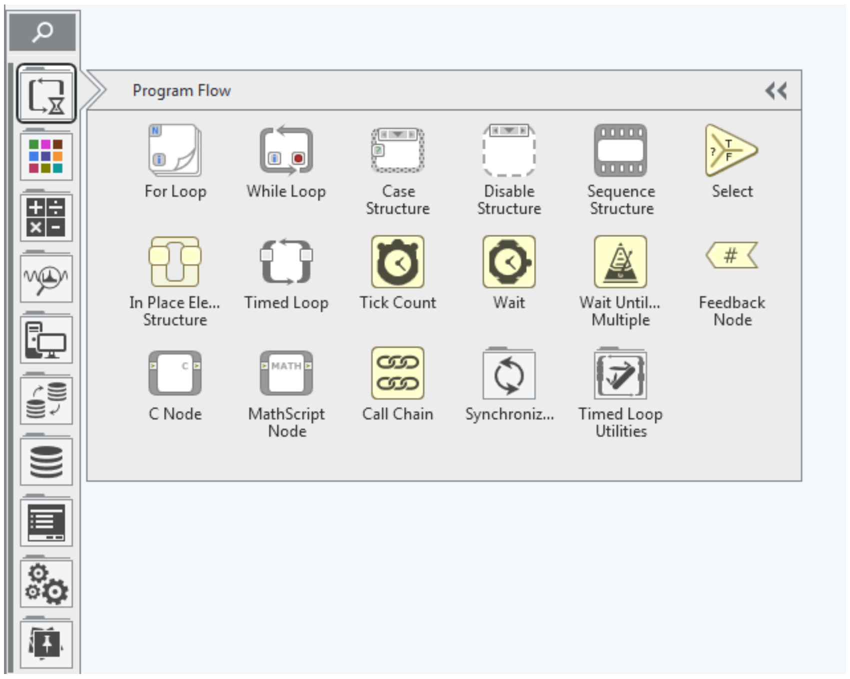
3.1. Terminology

A while loop is a control flow statement to execute a function repeatedly until a given conditional terminal is met. The conditional terminal defines when the loop stops. There are two settings for the conditional terminal: Continue if True and Stop if True. If it is set to Continue if True, the loop runs only if a Boolean control returns true. If the conditional terminal is set to Stop if True, and a Boolean value of true is sent, the loop stops execution. By default, the conditional terminal is Stop if True.

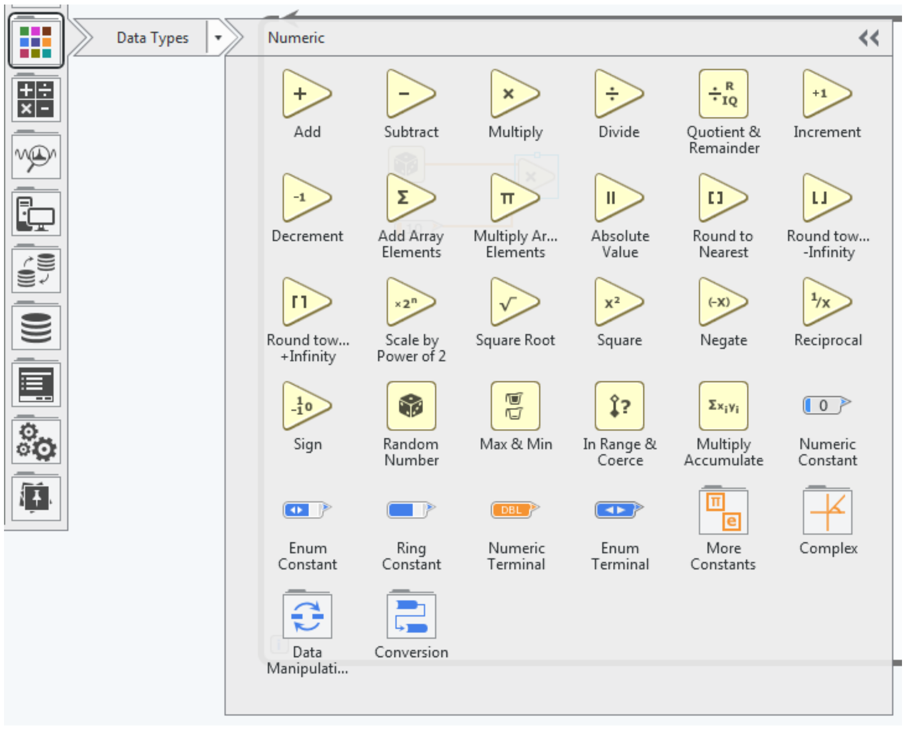
3.2. Example

The following example will explore the use of While loops. The example VI will generate random integers between 0 and 10, and it will stop if it is matched with the integer given by the user.

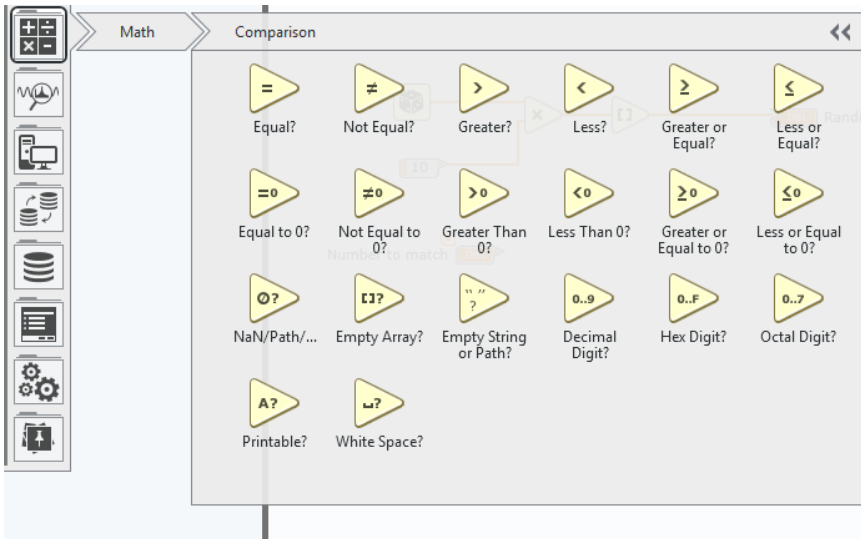
1. Open a blank VI.
2. Select Diagram tab and place a while loop in Program Flow subpalette.



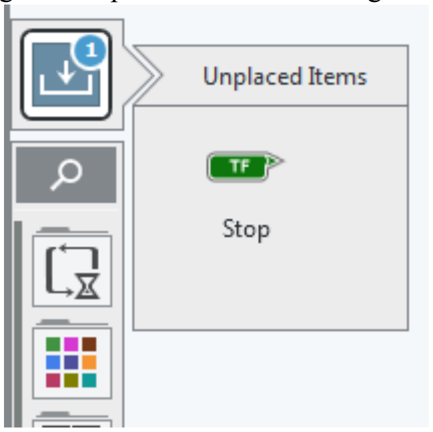
3. To generate random numbers, place a Random Number from Data Types → Numeric. The Random Number block generates random floating-point numbers between 0 and 1.



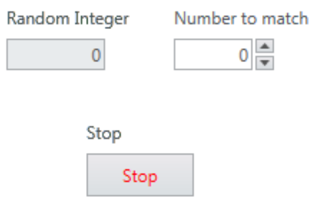
4. To generate random numbers between 0 and 10, use a Multiply and a Numeric Constant from Data Types → Numeric. Then, use Round to Nearest to make the number integer.
5. Create a Numeric Indicator to monitor the random integer every loop. Place a Numeric Terminal from Data Types → Numeric. The default behavior of the terminal is Control (input from Panel). Change to Indicator (output to Panel). Rename the indicator as Random Integer.
6. Create a Numeric Control to receive an integer to match. Rename the Control as Number to match.
7. Use Equal? to compare two numbers from Math→Comparison.

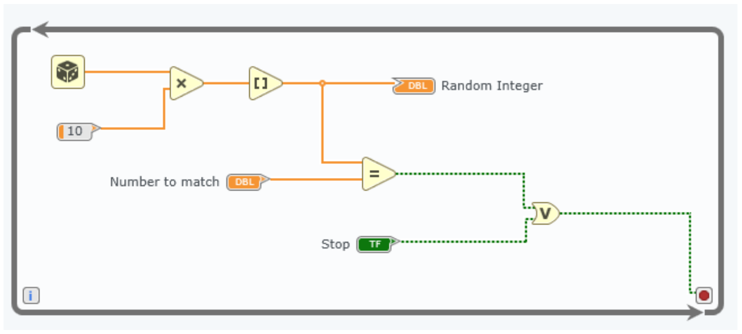


8. Now, we are ready to connect the output of the comparator to the conditional terminal. However, it is a good programming practice to allow users to terminate the loop by a control in the panel. Go to the panel and create a stop button from Buttons→ Stop Button. Go back to the diagram and you will see unplaced items in the top of the palette. Drag and drop the item into the diagram.



9. Place OR from Data Types → Boolean, then connect the Boolean control with the output of the comparator. In this way, the loop terminates either the random integer is matched or the STOP is clocked by the user.
10. Go back to the panel. Drag and drop the unplaced items in the palette.
11. The final panel and the diagram will look like the following.





12. Execute the code by clicking Run button.
13. Click the light bulb ("Highlight Execution") button at the top of the screen. Then, execute the code by clicking Run button. This will show you the data flow throughout the diagram which is very useful for debugging.

3.3. Questions

For all the below questions, include a figure of the diagram and panel in your report.

3.3.1. Finish the above example.

3.3.2. Add some delay (1 second) in the while loop so that you can see the number is changing.

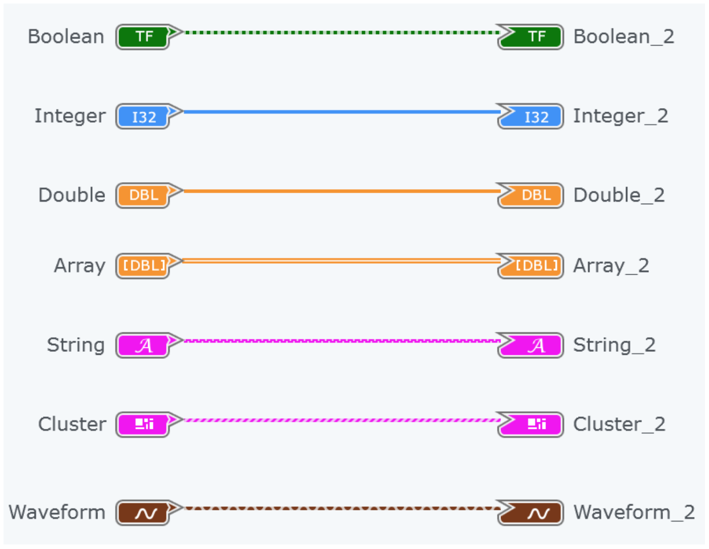
3.3.3. Add a Boolean indicator that turns on when the random integer is bigger than 5 and turns off when it is less than or equal to 5.

3.3.4. Add an indicator to show the number of iterations. For example, the indicator displays '1' at the first trial.

4. Data Type

4.1. Terminology

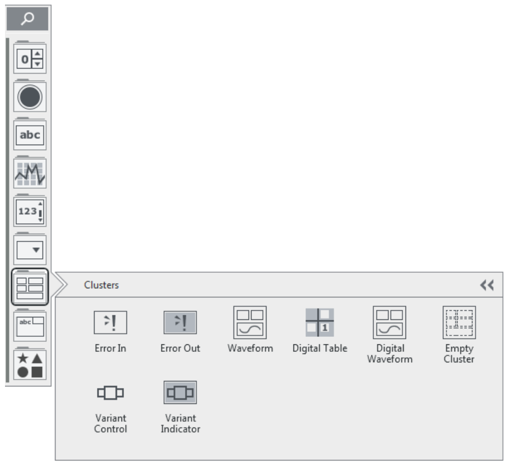
In LabVIEW, every object and wire is always associated with some data type. LabVIEW supports a variety of data types, which are distinguished by the different colors and shapes.



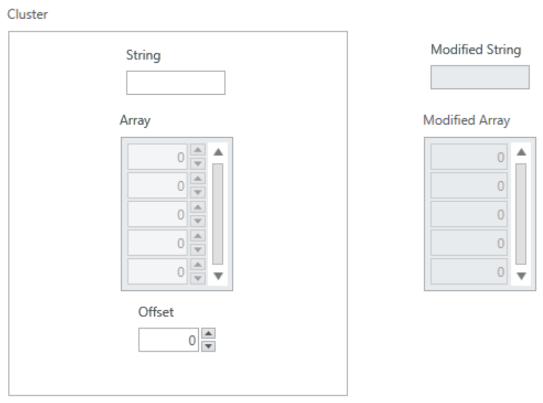
4.2. Example

The following example will demonstrate the use of the cluster data type. The VI will 1) receive a string and reverse it, 2) add an offset to a given numeric array.

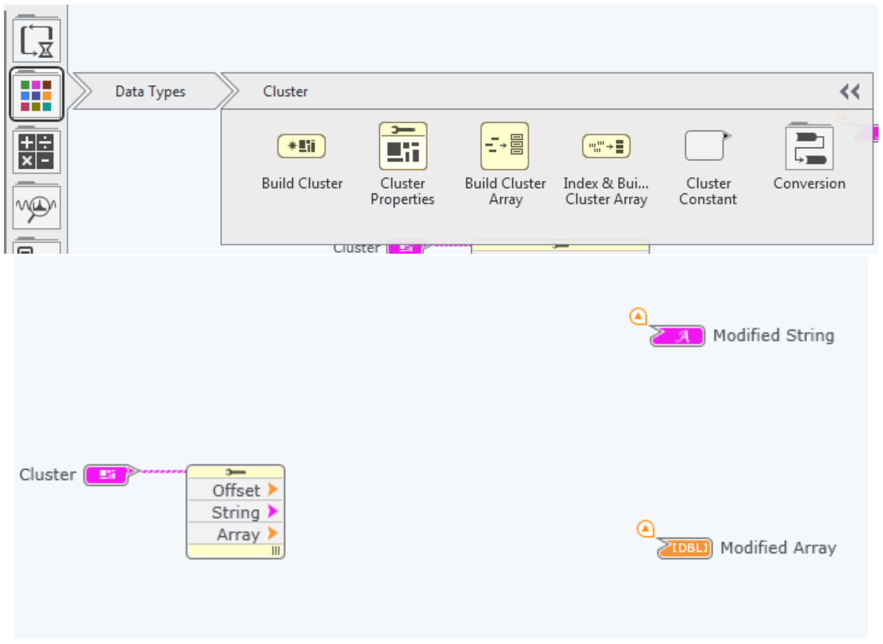
1. Open a blank VI.
2. In Panel, create an Empty Cluster.



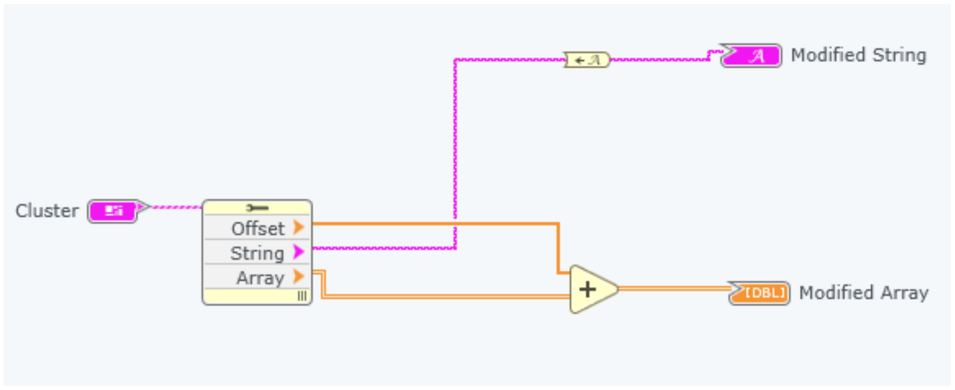
3. Place a String Control, a Numeric Control and a Numeric Array Input inside of the empty cluster. Change the size of the Numeric Array Input to have 5 elements by dragging up/down. Place another String Control and a Numeric Array Input outside out the cluster. Change them to Indicator.



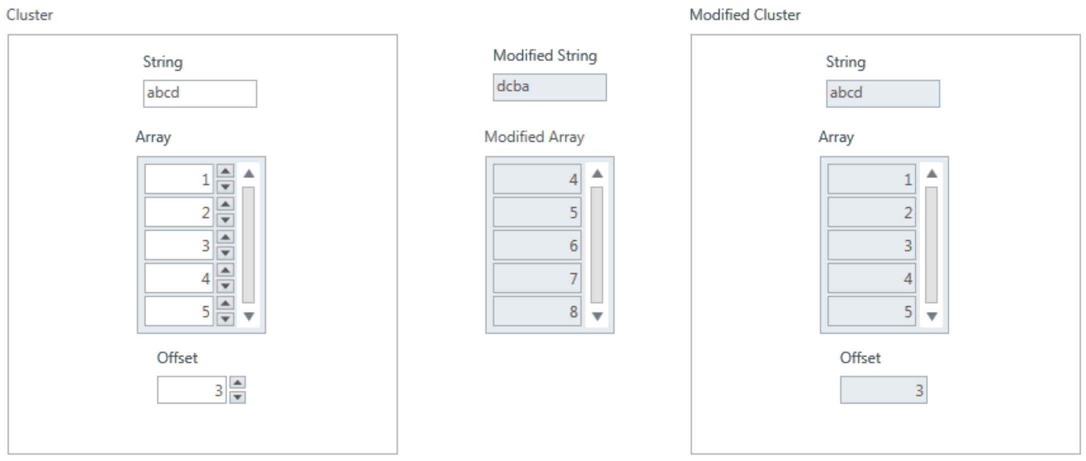
4. Go to Diagram and place the unplaced items. You need to read individual values from the cluster. To do that, use the Cluster Properties function. Connect the cluster control to the Cluster Properties. Grow it down to show more property terminals.



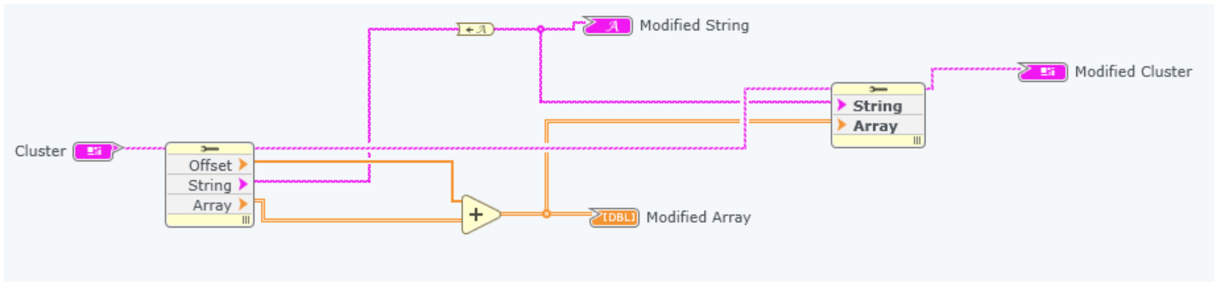
5. To reverse a string, use Reverse String function under Data Types → String. Use Add to add the offset to the array as shown below.



6. Go to Panel and fill the control values, then run the code. Verify the result with the indicators.
7. Now, let’s combine the modified values into the cluster again. Copy and paste the cluster in Panel and change to Indicator.



8. Go to Diagram and place the new cluster. To combine the modified string and the modified array into the new cluster, use the Cluster Properties function again. At this time, we want to write a cluster, so change it to “Set All to Write”. Wire the original cluster value to the “cluster in” terminal and wire the “cluster out” terminal to the new cluster indicator.



9. Run the code and compare the results.

4.3. Questions

4.3.1. Finish the above example. Provide a figure of your panel and diagram in the report

4.3.2. Can you simply wire an integer type control to a double type indicator? How about an integer type control to a string type indicator? If it is not possible, what block should you use to convert the type?

5. Arrays

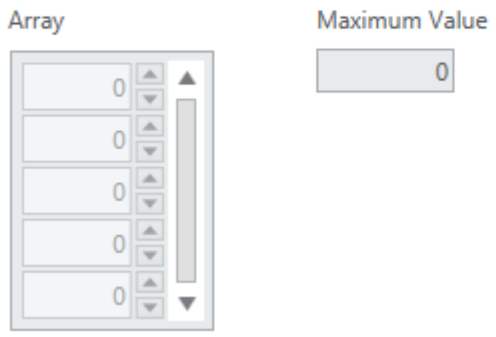
5.1. Terminology

Arrays combine data points of the **same** data type into one data structure (c.f. clusters combine data points of **multiple** data types into one data structure). You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types. Consider using arrays when you work with a collection of similar data points and when you perform repetitive computations. Arrays are ideal for storing data you collect from waveforms or data generated in loops, where each iteration of a loop produces one element of the array.

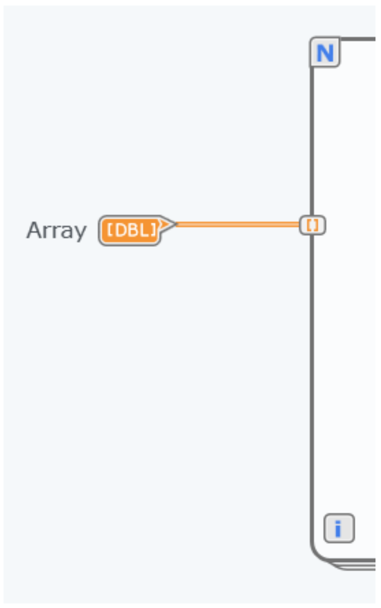
5.2. Example

The following example will explore the use of arrays. The example VI will find the maximum element in a given numeric array.

1. Open a blank VI.
2. Add a Numeric Array Input in the panel under Array. Increase the size of array to 5 by dragging down the Numeric Array Input. Rename as "Array".
3. Add a Numeric Control. Change it to Indicator. Rename it as "Maximum value".

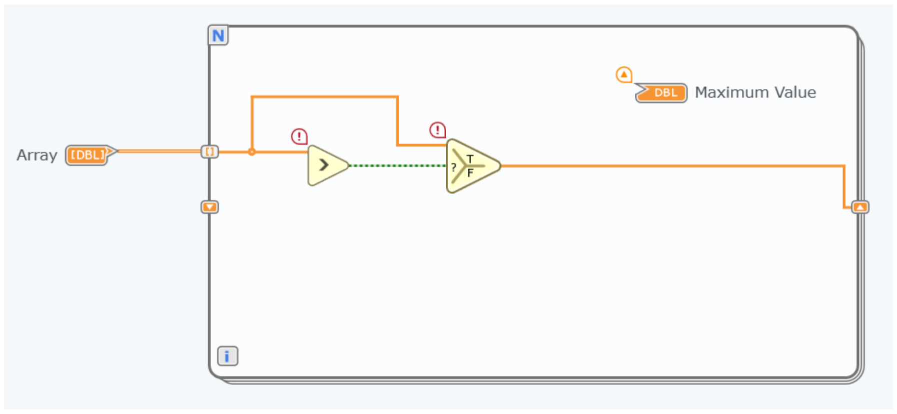


4. Go to Diagram. Add a For loop under Program Flow. Place the Array terminal outside of the loop and the Maximum Value terminal inside of the loop.
5. Wire the Array terminal to the left edge of the For loop. A tunnel will be created. The tunnel allows a loop to communicate with the blocks outside of it. Notice that the tunnel has indexing enabled (▣). When an incoming tunnel has indexing enabled, the loop structure (either a For Loop or a While Loop) will take the elements of the incoming array one-by-one. If indexing is disabled (■), the loop structure will take the array in as a whole.
6. We will compare the elements in the incoming array one-by-one with the maximum value we found. Add a Greater? in Math→Comparison and a Select in Programing Flow.
7. If the current element is greater than the maximum so far, we want to update the maximum. Connect the tunnel to the X terminal of the Greater? and the True case terminal of the Select. Connect the output of the Greater? to the Selector terminal of the Select.

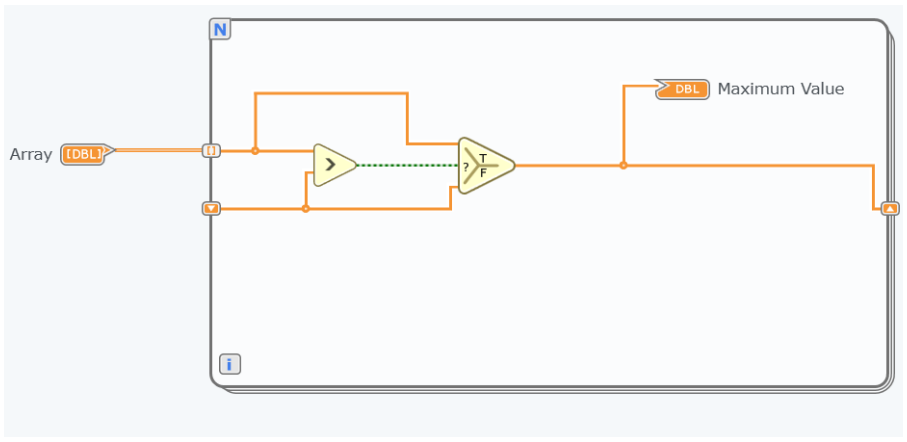


8. We want to store the current maximum value so that the next iteration can use it. Shift register is a data pipeline that can connect two adjacent iterations together. Connect the output of the Select to the right edge of the loop. It will create a tunnel by default. Change it to the Shift Register ( ) by right-clicking the tunnel. Notice that your mouse cursor changes to the Shift Register icon. Click the left edge of the loop ( )to finish the placement.

   **The left square of the shift register can be seen as an input from the previous iteration, and the right square of the shift register can be treated as an output for the current iteration.*



9. We now know the left square of the shift register is the maximum value from the previous iteration which is to be compared with the current element. Connect the left square of the shift register to the Y terminal of the Greater. If the current element is not greater than the previous maximum value, nothing will be updated. Connect the left square of the shift register to the False case terminal of the Selector.
10. Connect the output of the Selector to the Maximum Value indicator.



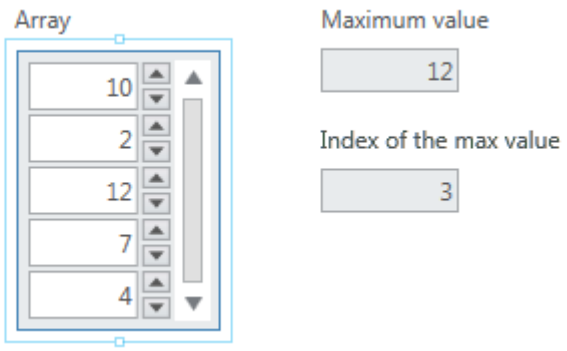
11. Input numbers into the array in the panel. Run the code and verify the result.

5.3. Questions

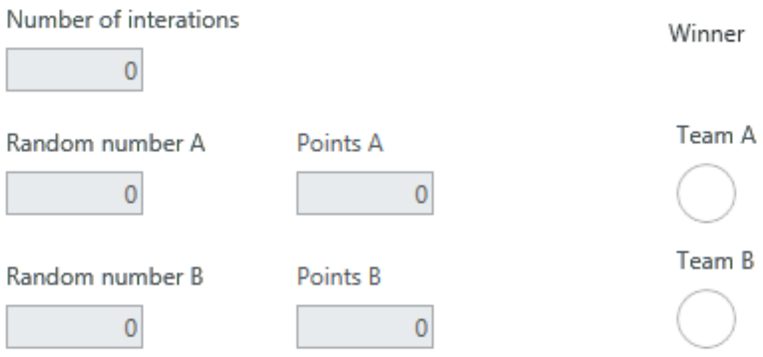
5.3.1. Finish the example. Show a figure of the diagram and panel.

5.3.2. Change the maximum element in the array to 0. Run the code again. Does the example VI find the correct value? Modify the example VI to find the right maximum value in every run.

5.3.3. Modify the above VI to output the index of the maximum value as well. The result will be like the following figure.



5.3.4. Implement the following VI: Generate two random numbers between -0.5 and 0.5. Compare the two numbers and if the first number is greater than the second one, Team A scores a point. When the second number is greater, Team B scores a point. The program keeps running until any team reaches 50 points first. Display the two random numbers, the number of iterations, the points of Team A and B, and the winner of the game. The panel will be like the following figure.



5.3.5. **(Optional: do this only after you are done with the lab)** Use arrays and for-loop to implement the system H where impulse response is $\boldsymbol{h[n] = \delta[n] + \delta[n-1]}$.

6. MathScript

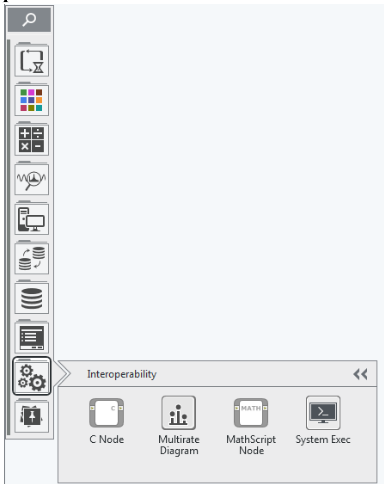
6.1. Terminology

Certain tasks arise in LabVIEW, such as scaling signals and solving mathematical equations, during which it can be beneficial to implement text-based programming in LabVIEW. LabVIEW provides the option to use C++ syntax with the Formula Node and .m file script syntax with the MathScript Node for signal processing, analysis, and math functions.

6.2. Example

In this example, we will use MathScript to generate a sin wave and control its amplitude and frequency by the controls.

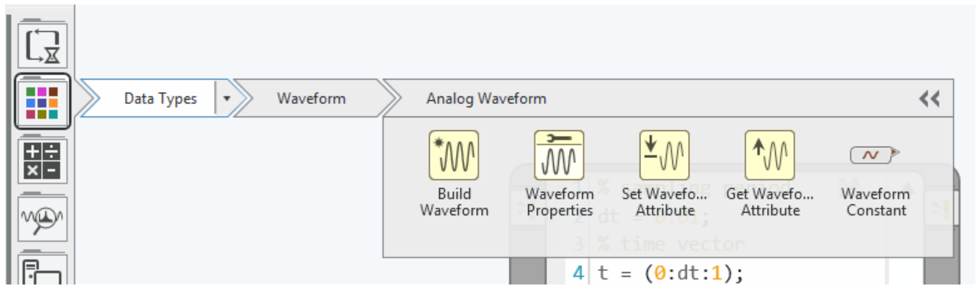
1. Open a blank VI.
2. Go to Diagram and place a MathScript Node.



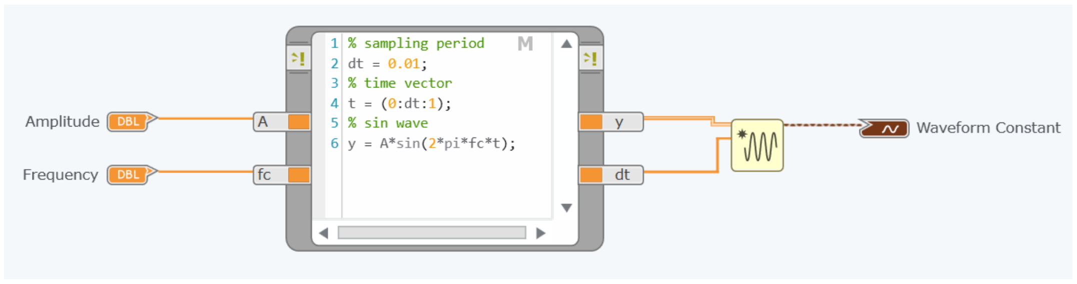
3. Type the following m-file code to generate a sin wave.

```
% sampling period
dt = 0.01;
% time vector
t = (0:dt:1);
% sin wave
y = A*sin(2*pi*fc*t);
```

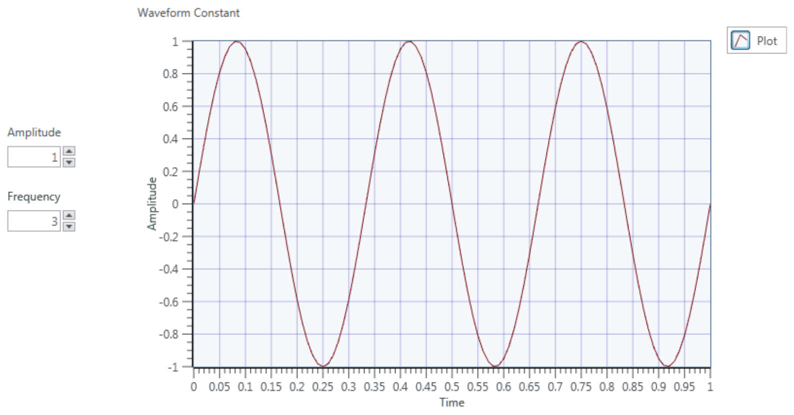
4. Create inputs for variable “A” and “fc” and outputs for “dt” and “y” by right-clicking the mouse on the MathScript Node.
5. Create the controls for the amplitude and the frequency of the sin wave and connect them to “A” and “fc” of the MathScript Node.
6. Place Build Waveform function from Data Types → Waveform →Analog Waveform. Connect the output “y” and “dt” of the MathScript Node to the input “Y” and “dt” of the Build Waveform.



7. Create Waveform Constant and change to Indicator and connect to the output of the Build Waveform.
8. Run the code.



```
% sampling period
dt = 0.01;
% time vector
t = (0:dt:1);
% sin wave
y = A*sin(2*pi*fc*t);
```



6.3. Questions

6.3.1. Finish the example.

6.3.2. Can you keep increasing the frequency and get the right sin wave? If not explain why.

6.3.3. Implement a sin wave using LabVIEW blocks only (without MathScript).