Introduction to G Programming

Jeff Kodosky
Eduardo Pérez, Ph.D.

Table of Contents

Introduction to G Programming

1 Introduction Tutorial

G is a high level, data-flow graphical programming language designed to develop applications that are

- Interactive
- Execute in Parallel
- Multicore

The program is a block diagram edited in the G Programming window.

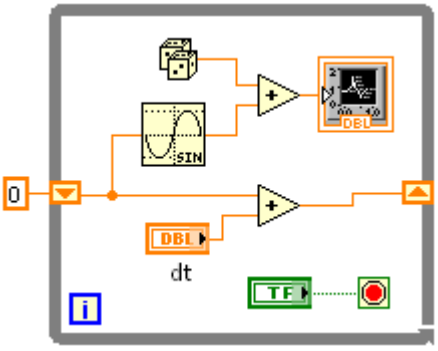


Figure 1: G Block Diagram

The program input data and results are manipulated and displayed in the GUI Window.

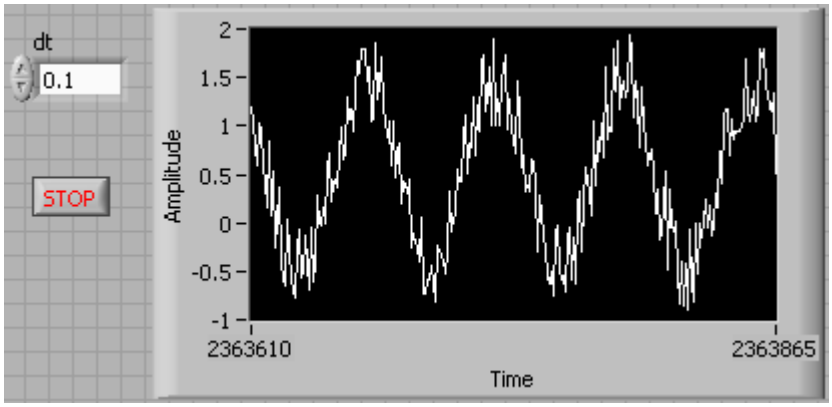


Figure 2: G User Interface

1.1 Hello Graphical Interactive Parallel Multicore World

The first program is to display the text "Hello graphical interactive parallel multicore world" in the GUI Window.

Right click on the G Programming Window and select **String Constant** from the **Functions >> Programming >> String** menu.

Drag and drop the **String Constant** onto the G programming window as show in Figure 3.

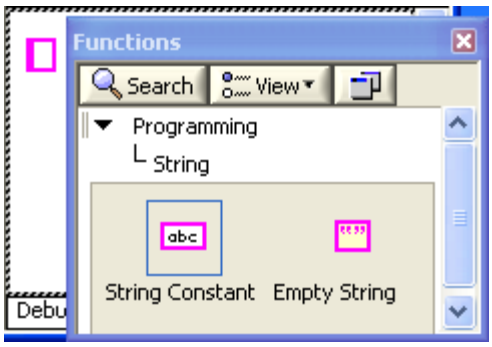


Figure 3: String Constant

Type in "Hello graphical interactive parallel multicore world." in the **String Constant**.

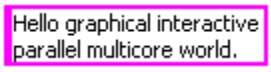


Figure 4: "Hello…world" String Constant

Right click in the GUI Window and select a **String Indicator** from the **Controls >> Modern >> String & Path** menu.

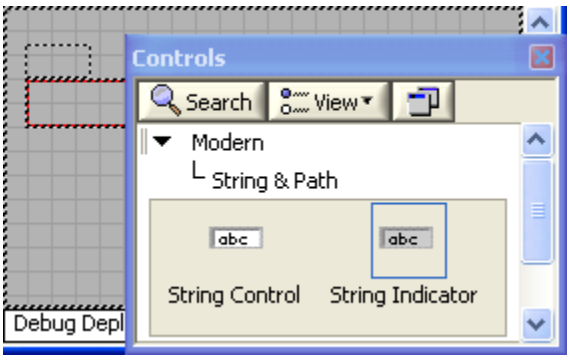


Figure 5: String Indicator

Drop it into the GUI Window.

Figure 6: String Output

Return to the programming window. Notice the string terminal corresponding to the **string indicator** in the GUI Window. As you approach the string constant from the right, the wiring terminal is highlighted and the pointer turns to wire spooler.

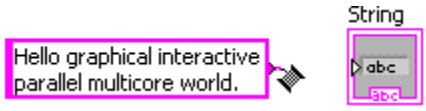


Figure 7: Wiring the G Diagram

Click the **"Hello graphical interactive parallel multicore world"** terminal and then click on the **String Indicator** triangular terminal to wire the terminals.

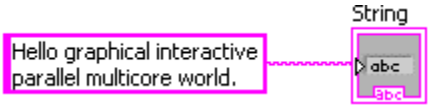


Figure 8: Wired G Block Diagram

Save your program as **Hello, World.vi**.

Return to the GUI Window. Click the run button (⇨).

You have successfully completed and executed your first G program.

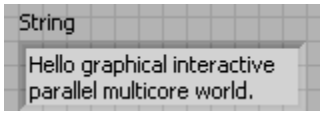


Figure 9: Hello, World G Program Executed

1.2 Arithmetic Expressions

The next program converts degrees from Fahrenheit to Celsius using the formula $C = \frac{5}{9}(F - 32)$

In the G Programming Window, select the subtract, multiply and divide from the **Functions >> Mathematics >> Numeric** menu

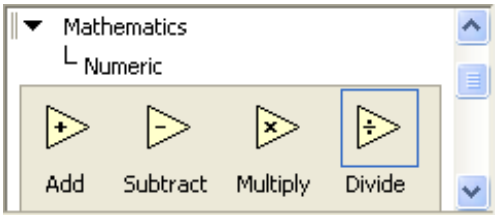


Figure 10: Numeric Operations

Wire the subtract, multiply and divide functions as shown in Figure 11.

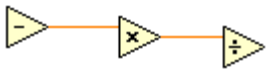


Figure 11: Subtract, Multiply and Divide

Right click on the upper left terminal of the subtract function and select **Create >> Control** from the pop-up menu.

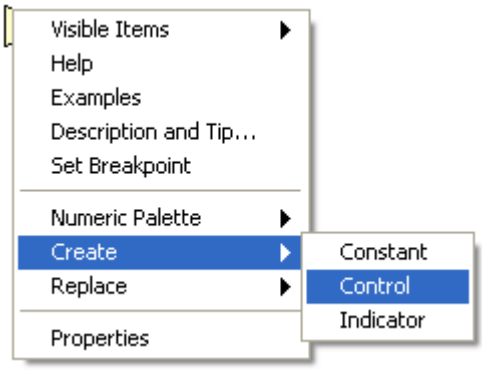


Figure 12: Create Control

Re-label **x** as **Fahrenheit** and wire the terminal as shown in Figure 13.

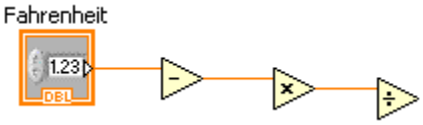


Figure 13: Fahrenheit Input Control

Right click on the lower left terminal of the subtract function and select **Create >> Constant** and type **32.0**.

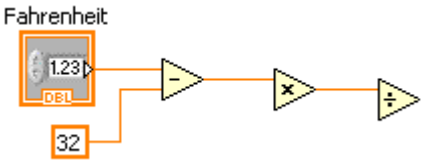


Figure 14: Fahrenheit Numeric Constant

Repeat the process to generate numeric constants for the multiply and divide function with **5.0** and **9.0** respectively.

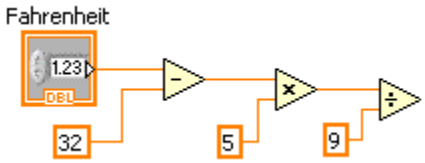


Figure 15: Fahrenheit Numeric Constants

To complete the program, right click on the right terminal of the divide function and select **Create >> Indicator**. Re-label **x/y** as **Celsius**. The final diagram is shown in Figure 16.

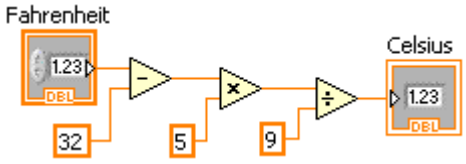


Figure 16: Fahrenheit to Celsius G Diagram

Switch to the GUI Window to run the program. Save the program as **Celsius.vi**. Try various Fahrenheit values to see the corresponding Celsius values. You have successfully finished a Fahrenheit to Celsius calculator.

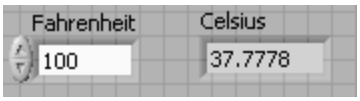


Figure 17: Fahrenheit to Celsius G Program

1.3 Functions

Click on empty space and drag to select the entire diagram.

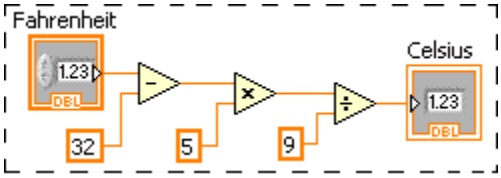


Figure 18: Select G Block Diagram

The selected diagram is highlighted as shown in Figure 19

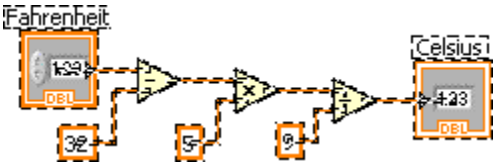


Figure 19: Selected G Block Diagram

From the **Edit** menu select **Create SubVI** to create a G function. The resulting diagram is shown in **Error! eference source not found.**.

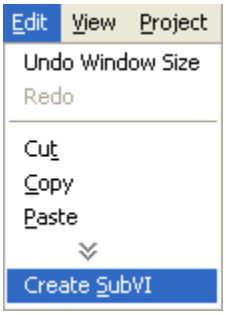


Figure 20: Creating a Function

From the **File** menu select **Save All** and save the **Untitled** function as **Fahrenheit to Celsius.vi**.

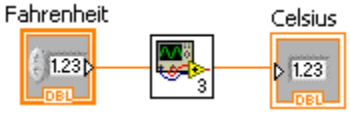


Figure 21: Diagram with Function

Open the **Fahrenheit to Celsius.vi** by double clicking on the icon. Right click on the icon editor (upper right corner) and select **Edit Icon…**

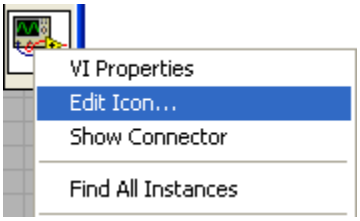


Figure 22: Edit Icon

This pops-up the **Icon Editor**. Edit the function's icon.

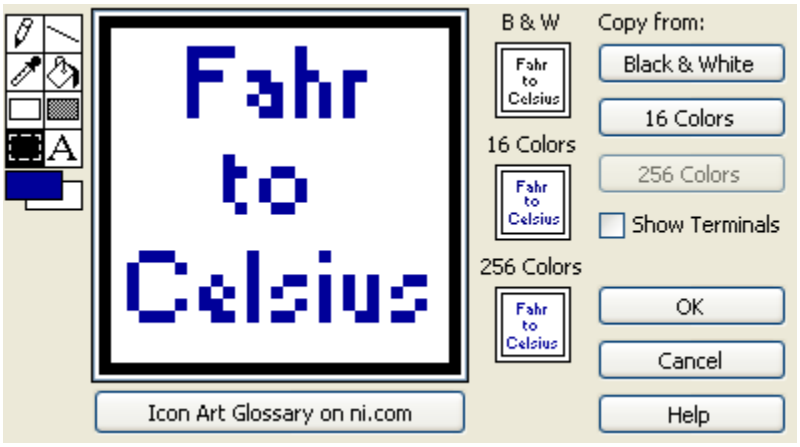


Figure 23: Icon Editor

After editing the icon, the function's icon is shown in the upper right corner of the GUI window. Save the function, plug in various input values and run the function. Save the function.

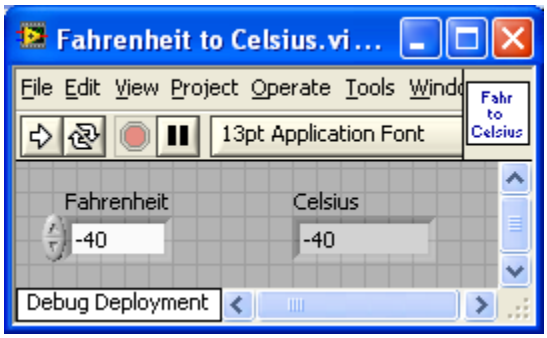


Figure 24: Edited Icon

Close the **Fahrenheit to Celsius** function and return to the **Celsius** G programming windows. The **Celsius** diagram reflects the updated **Fahrenheit to Celsius** icon

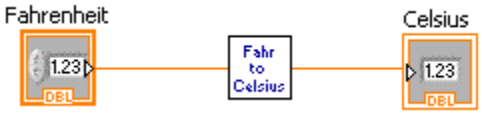


Figure 25: Function Calling

1.4 Case Selection

This program determines if a year is a leap year or not. A leap year is divisible by 4 but not by 100, except when it is divisible by 400. A number **x** is divisible by a number **y** if the remainder of **x/y** is identical to zero, i.e. $Rem\left(\frac{x}{y}\right) \equiv 0$, therefore

$$Leap\ Year = \left\{\left(Rem\left(\frac{Year}{4}\right) \equiv 0 \ \wedge \ \overline{Rem\left(\frac{Year}{100}\right) \equiv 0}\right) \vee Rem\left(\frac{Year}{400}\right) \equiv 0\right\}$$

For example: 1900 is not a leap year because it is divisible by 100
1970 is not a leap year because it is not divisible by 4
1980 is a leap year because it is divisible by 4 but not by 100
2000 is a leap year because it is divisible by 400

Start a new G program and right click on the G programming window. Go to the **Functions >> Programming >> Numeric** menu in the G programming window.

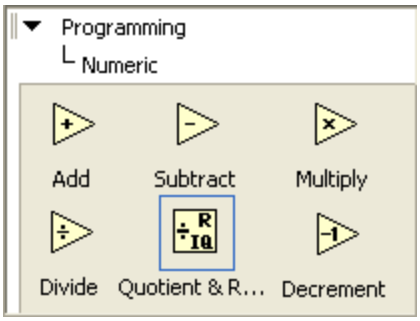


Figure 26: Quotient & Remainder Function

Select three copies of the **Quotient & Remainder** function and three numeric constants. Type in 4, 100 and 400 for the numeric constants and wire these constants to the lower input terminal (corresponding to the dividend) of the **Quotient & Remainder** function.

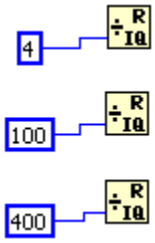


Figure 27: Leap Year Numeric Constants

From the **Functions >> Programming >> Comparison** menu, select 2 copies of the **Equal to Zero** function and one copy of the **Not Equal to Zero** function.

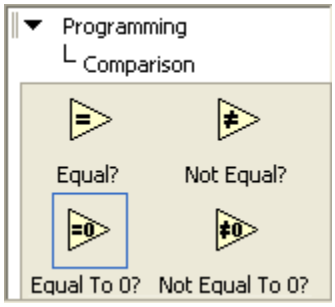


Figure 28: Comparison Functions

Organize the comparison operations as show in the diagram.

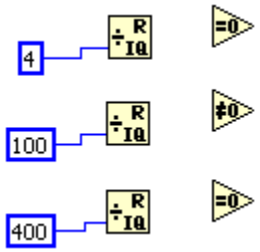


Figure 29: Q&R Comparison

From the **Functions >> Programming >> Boolean** menu select the **AND** and **OR** operators

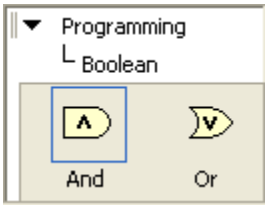


Figure 30: Boolean Operators

Place the Boolean operators as shown in Figure 31.

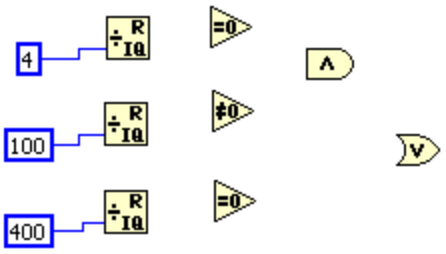


Figure 31: Q&R, Comparison & Boolean Functions

From the **Functions >> Programming >> Structures** menu, click on the **Case Structure**.

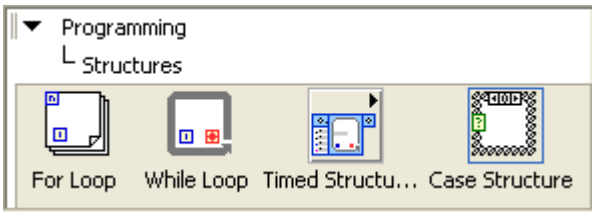


Figure 32: Case Structure

Click and drag on the G programming window to create the **Case Structure**.

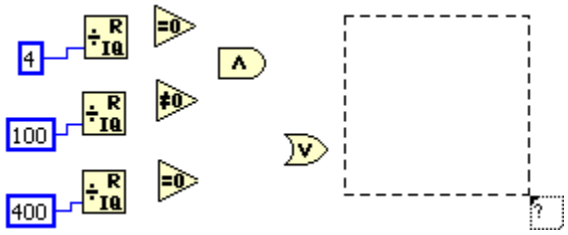


Figure 33: Creating a Case Structure

The **True** diagram option is indicated at the top of the case structure.

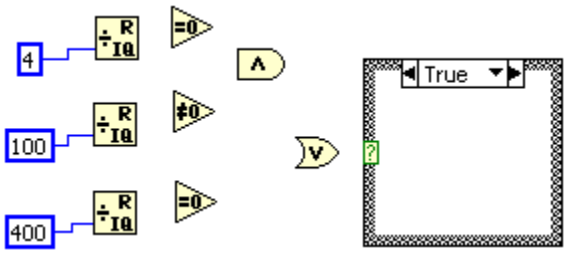


Figure 34: Created Case Structure

Drop a string constant and type “Is a Leap Year”.

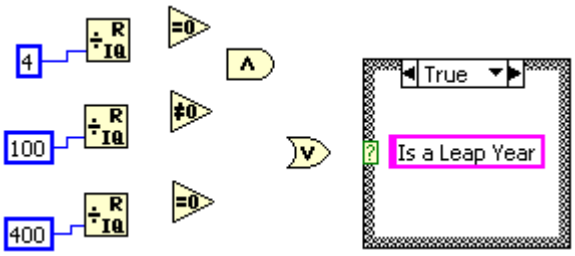


Figure 35: True Case Editing

Click on the down arrowhead next to the **True** label and select the **False** option.

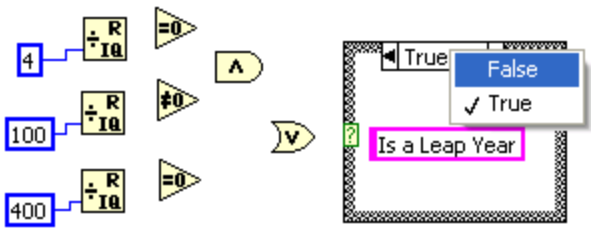


Figure 36: Selecting the False Case

Drop another string constant and type "Is not a Leap Year".

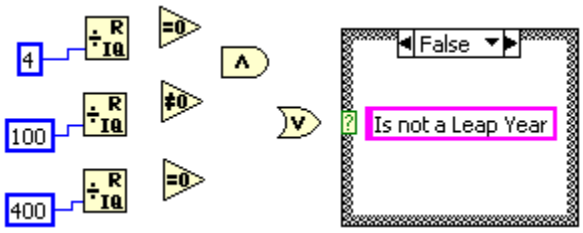


Figure 37: False Case Editing

Go to the GUI window and place a numeric input and an output string. Re-label the numeric input to **Year** and the output string to **Message**.

Figure 38: Leap Year GUI

Right click on **Year** and select **Representation >> I32** from the numeric pop-up menu.

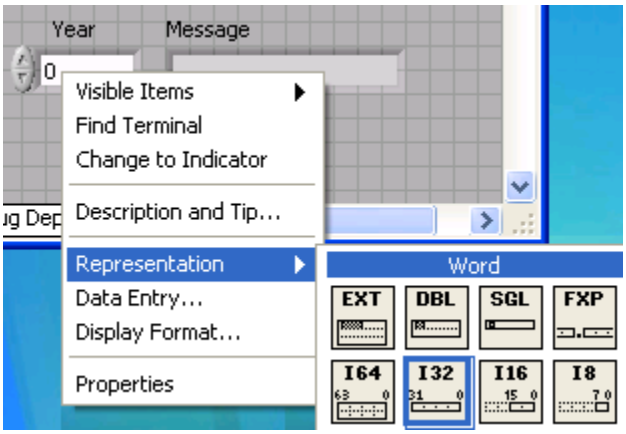


Figure 39: 32-Bit Integer Numeric

Arrange the **Year** and **Message** terminals in the G programming window as shown in the figure.

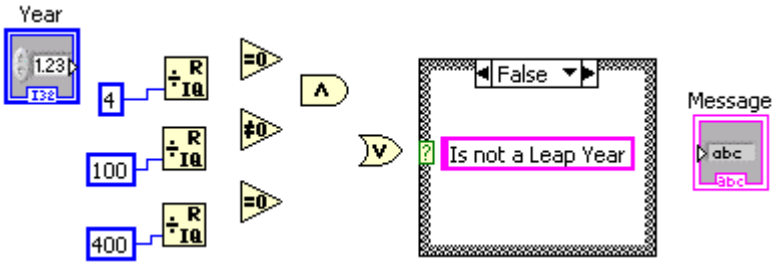


Figure 40: Unwired Leap Year Diagram

Wire the **OR** operator is to the "?" in the case structure and the string constant "Is not a Leap Year" is wired to **Message**.

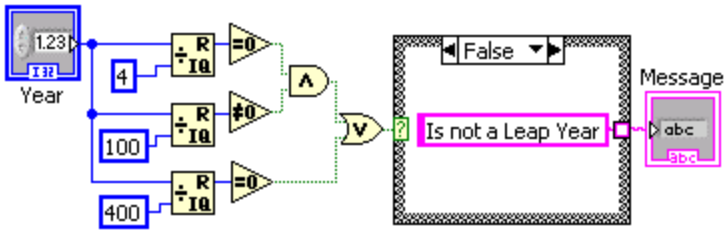


Figure 41: Leap Year False Case

Select the **True** option and Wire the "Is a Leap Year" string constant to the output terminal of the **Case Structure**.

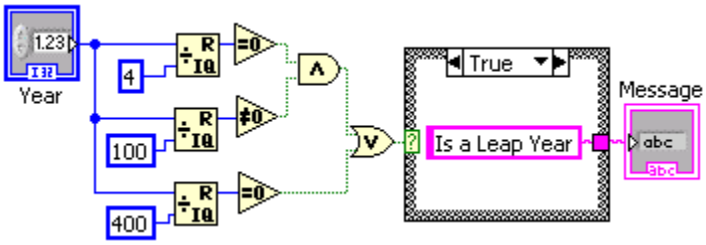


Figure 42: Leap Year True Case

Save the program as **Leap Year.vi**, enter **Year** values and run the program to determine whether the value of **Year** is that of a leap year or not.

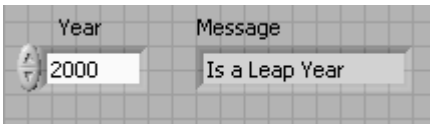


Figure 43: Leap Year Program

1.5 Arrays

Right click on the GUI window and select **Array** from the **Controls >> Modern >> Arrays, Matrix & Cluster** menu, and drop an array onto the GUI window.

The array structure consists of an **index** or **element offset** (left portion of the structure) and the array elements (right portion of the structure). When the array structure is placed on the GUI window, the data type of the array is undefined as indicated by the grayed out portion of the array.

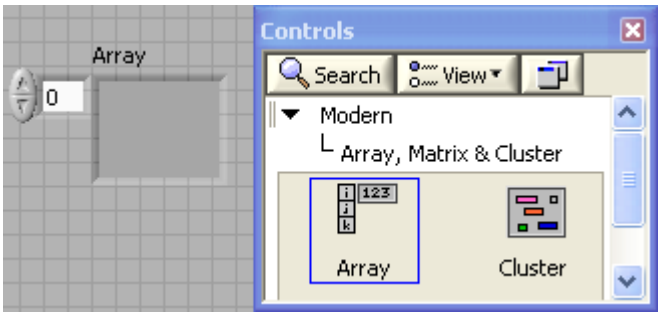


Figure 44: Arrays

To define the array data type, drag and drop a data type onto the array structure. For instance, to create an input array of numbers, place **Numeric Control** into the array structure.

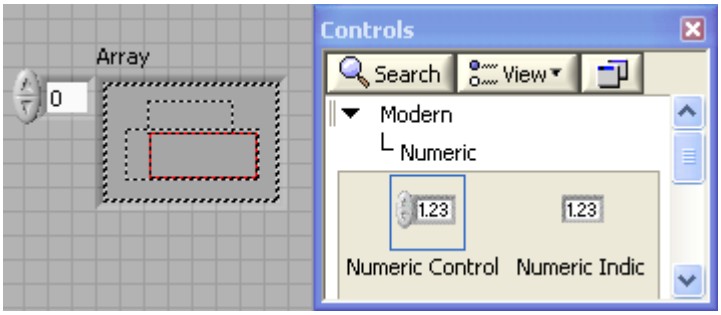


Figure 45: Creating a Numeric Array

At this point, the numeric array is an **Empty** or **Null** array because no elements of the array have been defined. This is indicated by the grayed out numeric control within the array structure.

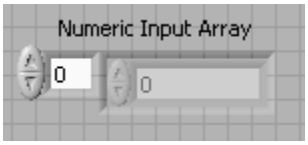


Figure 46: Empty Numeric Array

Define elements of an input array by selecting the offset and entering its value. For instance, at offset = 4, enter the value 0.0. This defines **Numeric Input Array** as **{0, 0, 0, 0, 0}**.

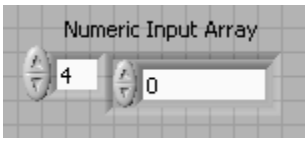


Figure 47: Defining Numeric Array Elements

An output array is created similarly to an input array with the exception that an output data type needs to be dropped into the array structure.

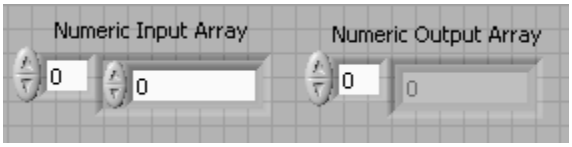


Figure 48: Creating Output Numeric Arrays

1.6 For Loop

This program converts an array of Fahrenheit values to Celsius. Create numeric input and output arrays and label them **Fahrenheit** and **Celsius** respectively. In the **Fahrenheit** array enter the values 0, 20, 40, 60, 80, 100, 120, 140, 160, 180 and 200 at offsets 0 through 10 as shown in Figure 49.

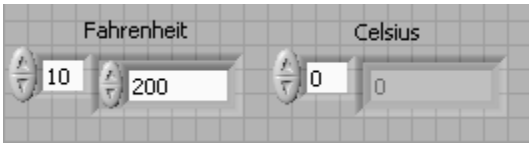


Figure 49: Numeric Input and Output Arrays

Right click in the G programming window, navigate to **Programming >> Structures** and click on **For Loop**.

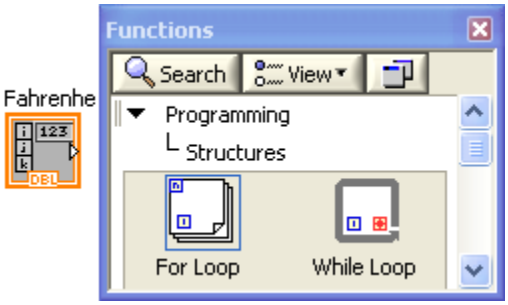


Figure 50: For Loop Structure

Click and drag to create the **For Loop** as shown in Figure 51 and Figure 52.

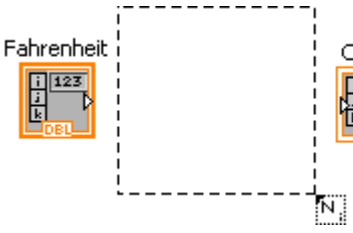


Figure 51: Creating For Loops

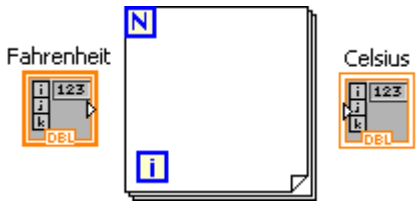


Figure 52: For Loop

Right click inside the **For Loop** and select **Select a VI…** from the pop-up menu. Find the **Fahrenheit to Celsius.vi** and click **OK**. Drop the function inside the **For Loop**.

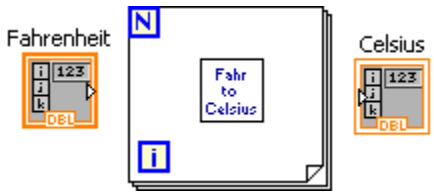


Figure 53: Function in Diagram

To complete the program, wire the **Fahrenheit** input array to the input terminal of the **Fahrenheit to Celsius** function and wire the output terminal of the **Fahrenheit to Celsius** function to the **Celsius** output array.

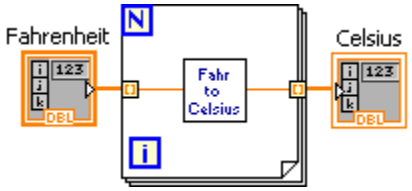


Figure 54: Wired Function in Diagram

This program uses the **For Loop** to select each element in the **Fahrenheit** input array, converts that value to Celsius and saves the results in the **Celsius** output array. Save the program as **Fahrenheit to Celsius For Loop.vi** and run the program.

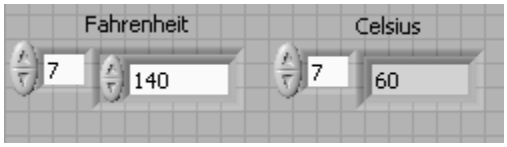


Figure 55: Fahrenheit to Celsius Arrays

The **Celsius** output array contains: **Celsius** = {-17.7778, -6.6667, 4.44444, 15.5556, 26.6667, 37.7778, 48.8889, 60, 71.1111, 82.2222 and 93.3333}

1.7 While Loop

The next program will generate Fahrenheit values and convert them to Celsius until a condition is met to stop the iterations in a **While Loop**. In the G programming window, select the **While Loop** structure by clicking on it from the **Functions >> Programming >> Structures** menu.

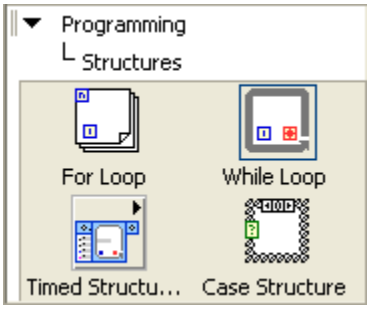


Figure 56: While Loop Structure

Click and drag to create the **While Loop** structure.

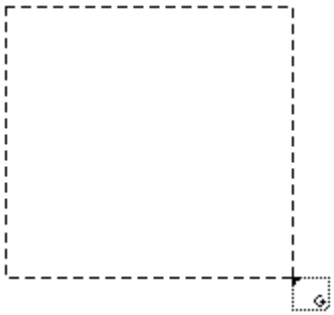


Figure 57: Creating a While Loop

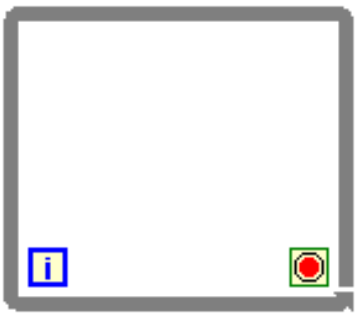


Figure 58: While Loop

In the GUI window, create two numeric output arrays. Label them **Fahrenheit** and **Celsius**.

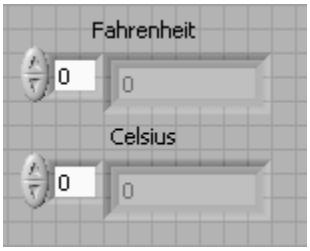


Figure 59: Numeric Output Arrays

Re-arrange the diagram as in Figure 60.

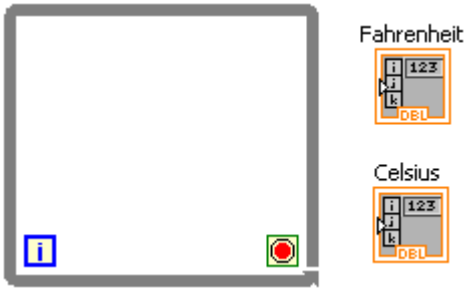


Figure 60: While Loop Diagram

From the **Functions** menu, select **Multiply** function and a couple of numeric constants. Type in **20.0** and **300.0** for the numeric constants. Select the **Fahrenheit to Celsius.vi** and drop it inside the **While Loop**. Re-arrange the diagram to look like Figure 61.

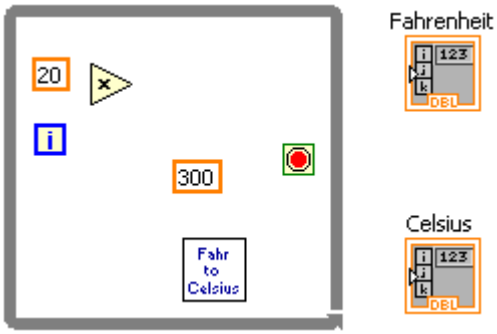


Figure 61: Generating Fahrenheit Values

From the **Functions >> Programming >> Comparison** menu select the **Greater or Equal** operator.

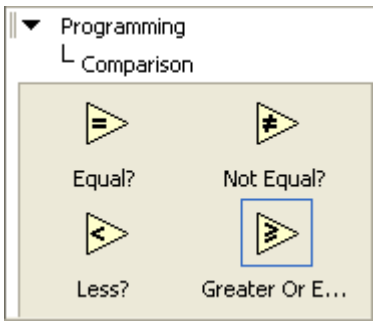


Figure 62: Greater or Equal Function

Wire the **While Loop** components as shown in Figure 63.

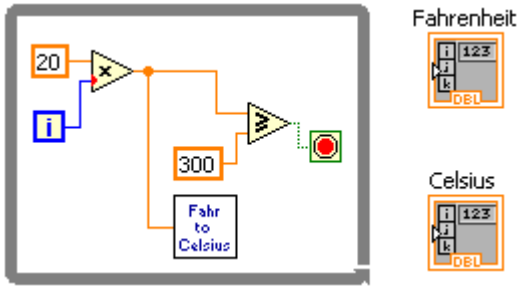


Figure 63: Generating Fahrenheit Values & Stop Condition

Wire the output of the **Multiply** operation to the **Fahrenheit** and the output of the **Fahrenheit to Celsius** function to the **Celsius** numeric output arrays. The connections between the **While Loop** and the **Fahrenheit** and **Celsius** arrays are broken (see Figure 64).

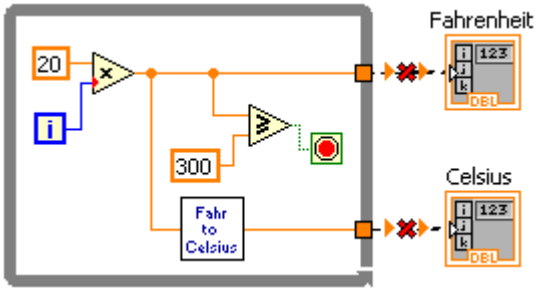


Figure 64: Broken Wires

To repair the broken connections, roll over the mouse pointer to the **Loop Tunnel**.

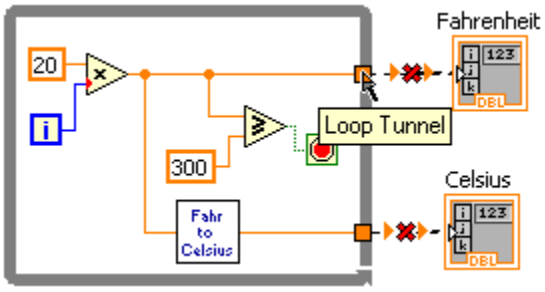


Figure 65: Loop Tunnel

Right click on the **Loop Tunnel** and select **Enable Indexing** from the pop-up menu.

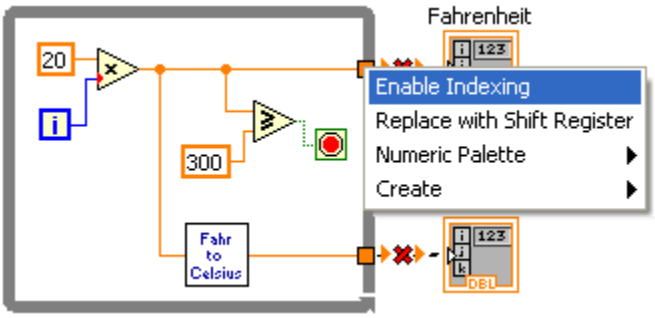


Figure 66: Enable Loop Indexing

This enables values to accumulate and store the results into an array.

Repeat for the **Celsius** array.

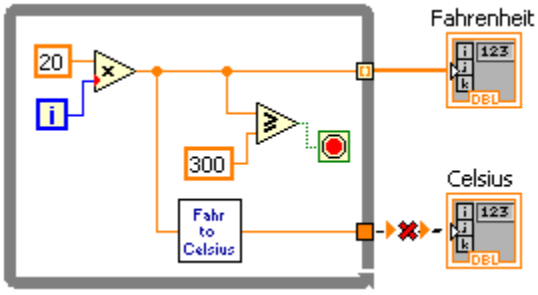


Figure 67: Broken Wire Repaired

Each iteration of the **While Loop** in this program generates an **i** × **20** Fahrenheit value and converts it to Celsius. The **While Loop** stops iterating when the generated Fahrenheit value is greater than or equal to 300. The resulting arrays are stored in the **Fahrenheit** and **Celsius** numeric output arrays.

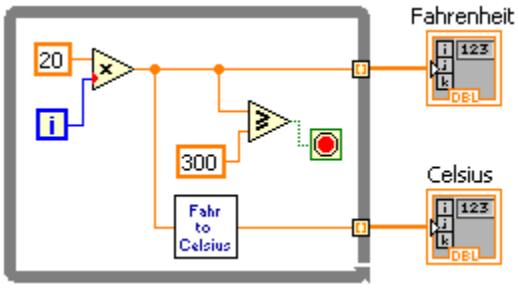


Figure 68: Fahrenheit to Celsius While Loop

Save the program as **Fahrenheit to Celsius While Loop.vi** and run it. The program generates the following results:

Fahrenheit = {0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 240, 260, 280, 300}

Celsius = {-17.7778, -6.6667, 4.44444, 15.5556, 26.6667, 37.7778, 48.8889, 60, 71.1111, 82.2222, 93.3333, 104.444, 115.556, 126.667, 137.778, 148.889}

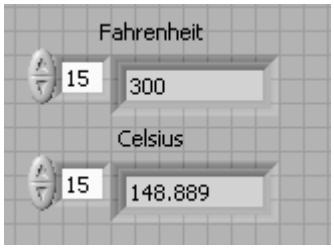


Figure 69: Fahrenheit and Celsius Arrays

1.8 Graphs

Using the previous G program example, we will now visualize the results by adding a graph to the GUI windows. Right click on the GUI Window. Select **XY Graph** from the **Controls >> Modern >> Graph** menu.

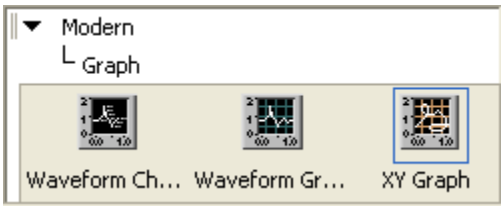


Figure 70: XY Graph Selection

Drop the **XY Graph** in the GUI window. Double click on the x and y axis labels and rename **Time** to **Fahrenheit** and **Amplitude** to **Celsius**.

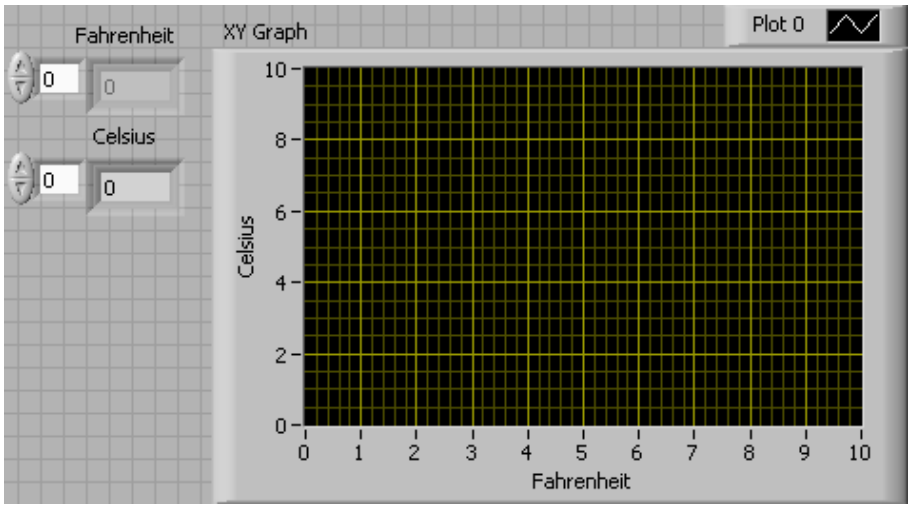


Figure 71: XY Graph in GUI Window

The G programming window contains the **XY Graph** terminal.

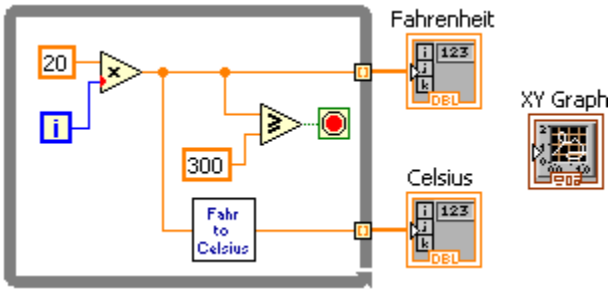


Figure 72: XY Graph Terminal in Diagram

Select **Bundle** from the **Functions >> Programming >> Cluster, Class & Variant** menu

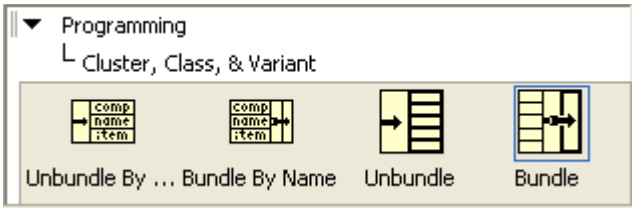


Figure 73: Bundle Operator

w

Drop it on the diagram as shown in Figure 74.

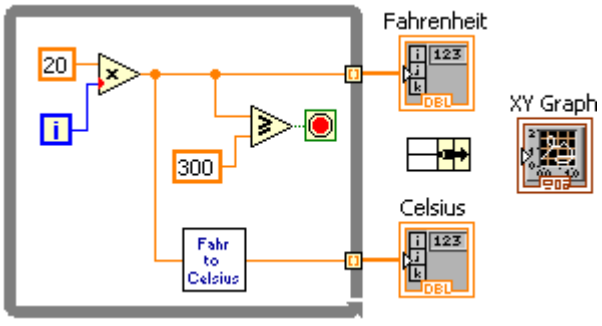


Figure 74: Bundle for XY Graph

Wire the **Fahrenheit** and **Celsius** results to the input **Bundle** terminals and the output **Bundle** terminal to the **XY Graph**.

Save the program and run it. The resulting graph is shown in the figure below.

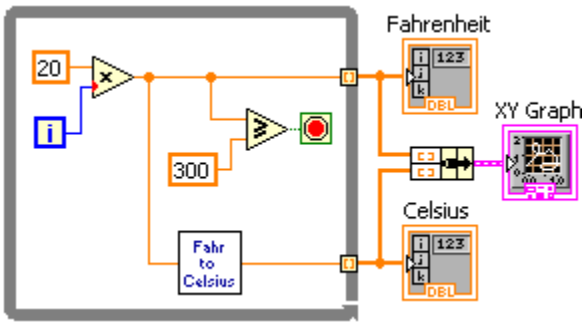


Figure 75: Wired XY Graph

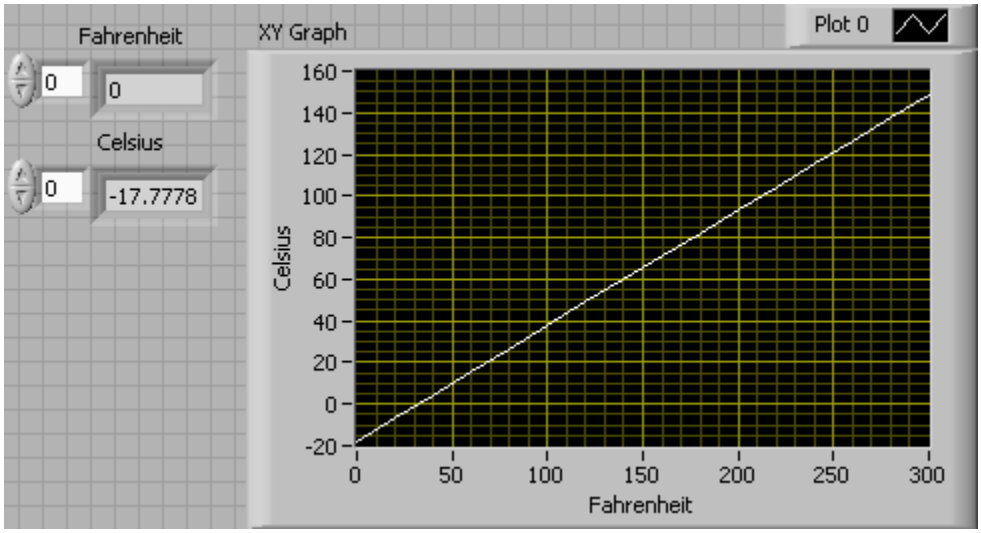


Figure 76: XY Graph Result

1.9 Interactivity

This G program shows how G allows programmers to develop interactive programs. Create the following G program and wire it as shown in the figure below.

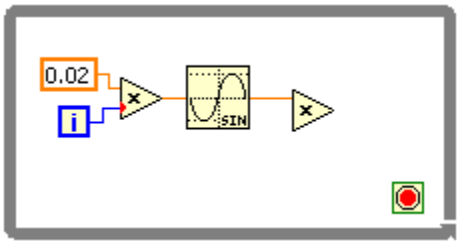


Figure 77: Creating Interactive Programs

In the GUI window, from the **Functions >> Modern >> Numeric** select the vertical pointer slide. From the **Functions >> Modern >> Graph** select **Waveform Chart**.

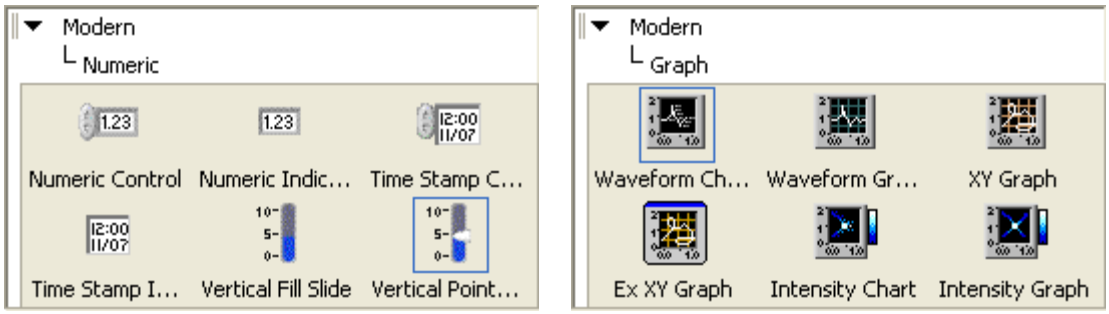


Figure 78: Vertical Pointer Slide and Waveform Chart

Re-label the vertical pointer slide as **Amplitude** and the waveform chart as **Sine Wave**. Re-arrange to GUI to look like the figure below.

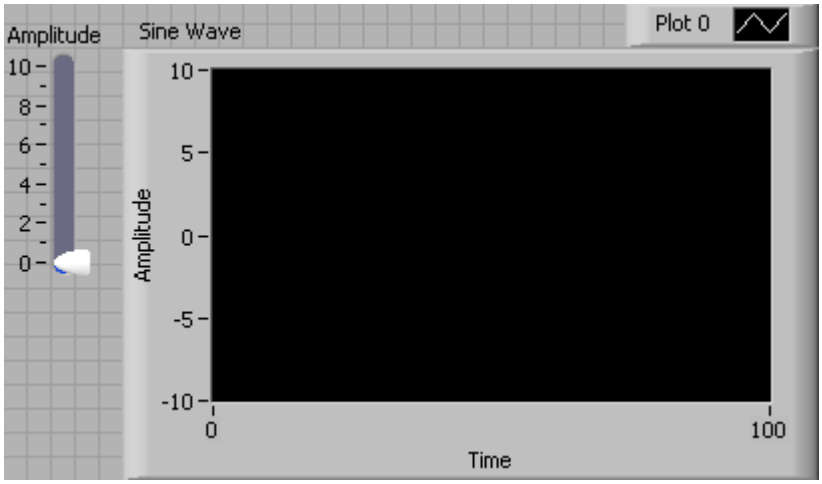


Figure 79: Slide & Waveform Chart in GUI Window

Right click on **Sine Wave** and select **Properties** from the pop-up menu.

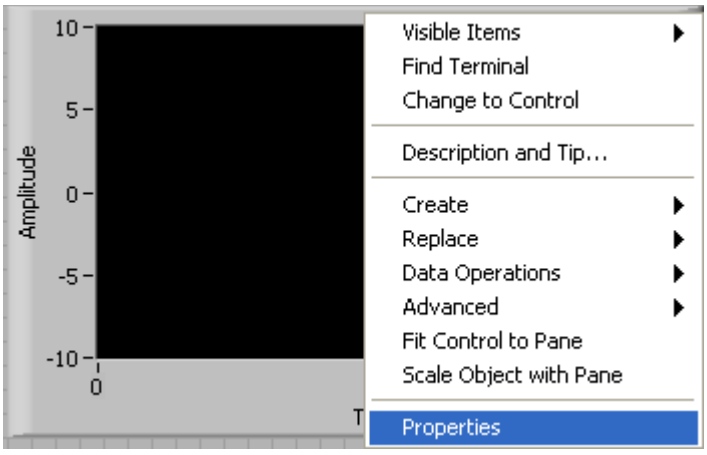


Figure 80: Selecting Chart Properties

Select the **Scales** tab and change **Maximum** to 1023. **Sine Wave** will display 1024 samples.

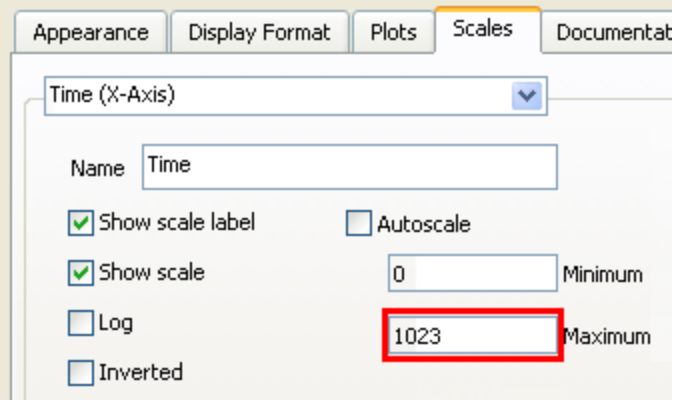


Figure 81: X-Axis Maximum

Click on the down arrow located to the right of **Time (X-Axis)** and select **Amplitude (Y-Axis)**.

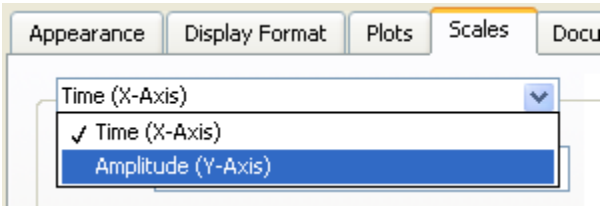


Figure 82: Selecting Y-Axis

De-select **Autoscale** and change the **Minimum** and **Maximum** values to **-10** and **10**. Click **OK**.

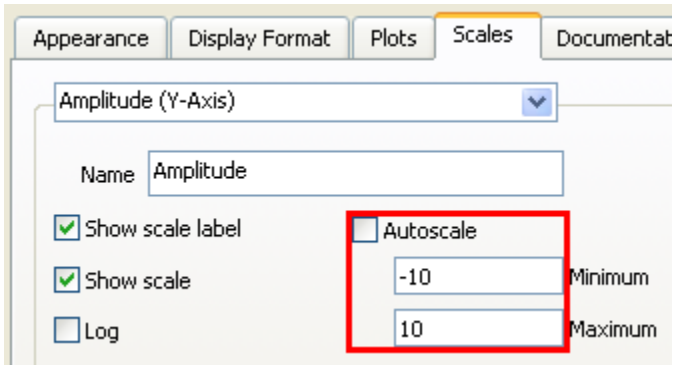


Figure 83: De-Selecting Autoscale

In the G programming window, re-arrange the **Amplitude** and **Sine Wave** terminals and finish the program as shown in Figure 84.

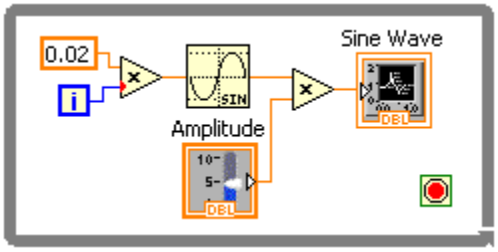


Figure 84: Interactive Sine Wave Diagram

Scroll the mouse pointer over the **Loop Control**…

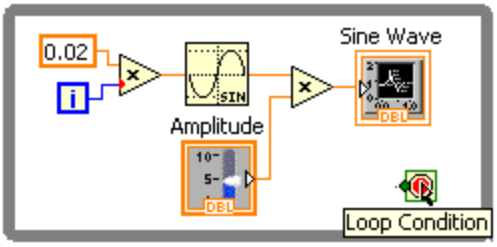


Figure 85: Loop Condition

And right click on the **Loop Control** and from the pop-up menu select **Create Control**.

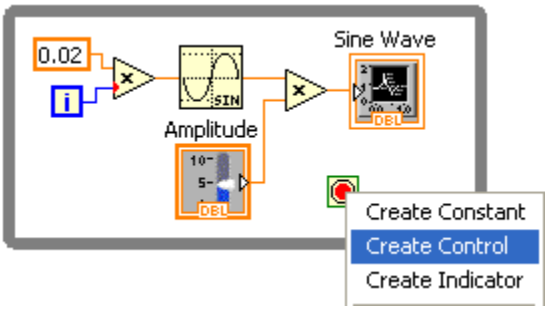


Figure 86: Create Loop Control

A **stop** terminal is created…

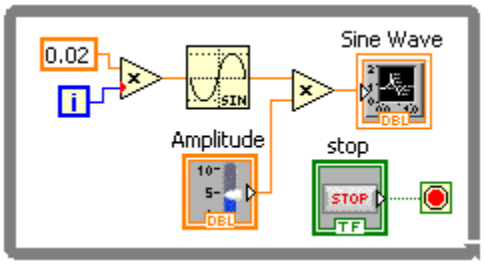


Figure 87: Interactive G Program

With the corresponding **stop** Boolean input control. Save the G program as **Interactivity.vi**.

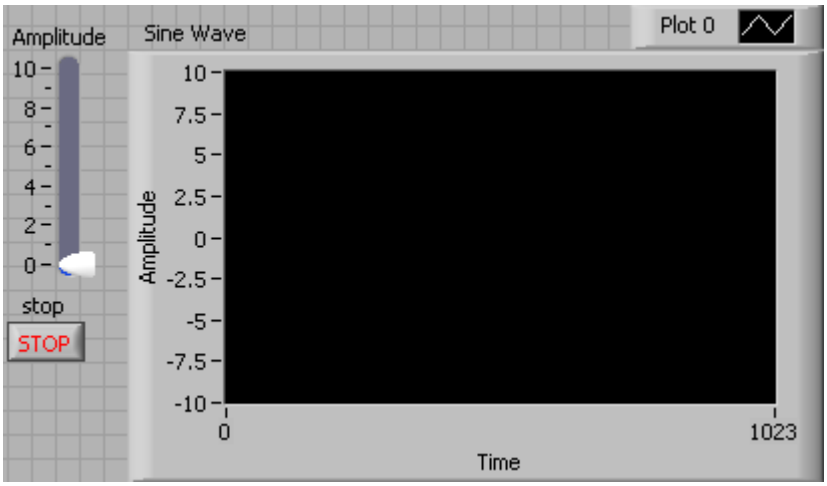


Figure 88: Interactive Program

Run the G program.

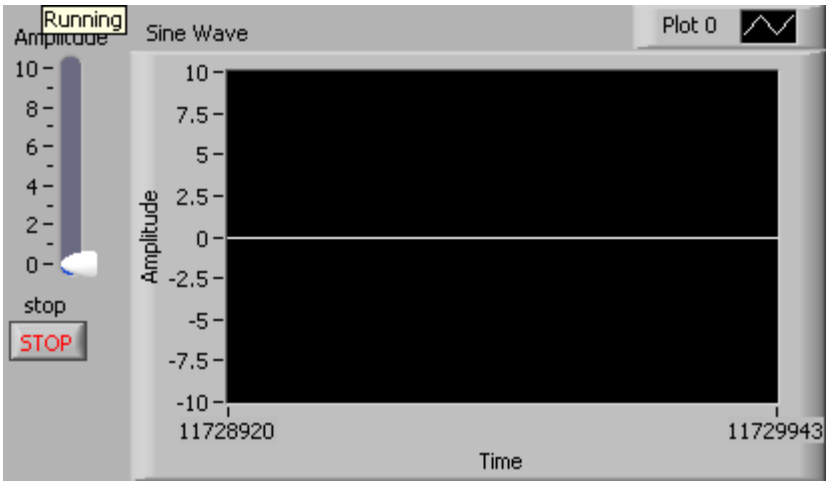


Figure 89: Interactive Program

While the program is running, change the **Amplitude** and watch the graph update to reflect the interactive changes.

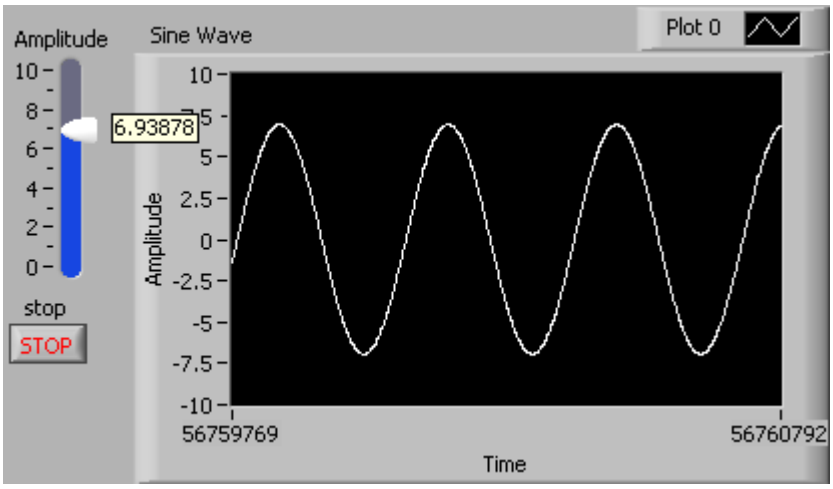


Figure 90: Interactive Program

To end the G program, simply click on the **stop** button.

Congratulations. You have successfully completed and executed your first interactive G program.

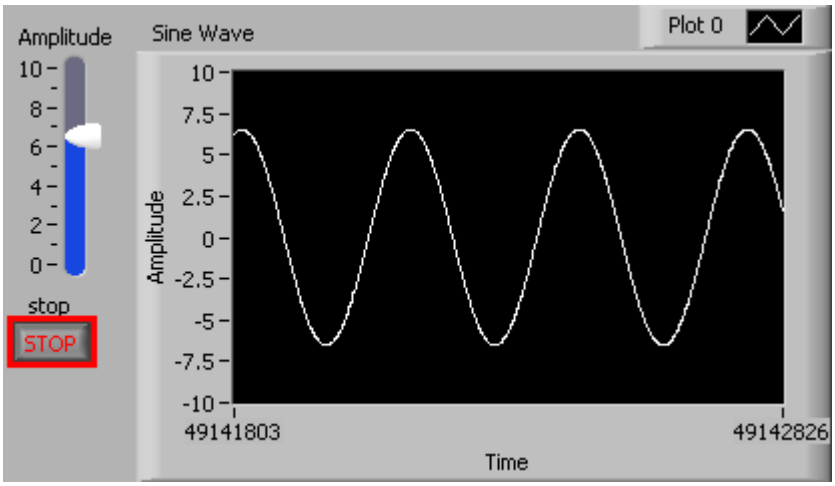


Figure 91: Interactive Program

1.10 Parallel Programming

Save a copy of **Interactivity.vi** as **Parallel Programming.vi**. Select the while loop as shown in the figure below.

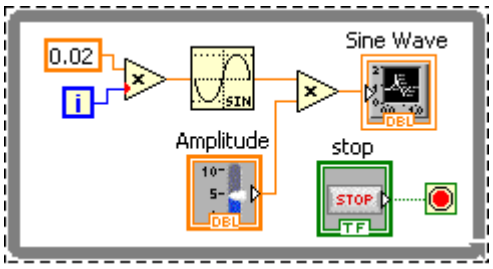


Figure 92: Select Diagram for Parallel Programming

From the menu select **Edit >> Copy**.

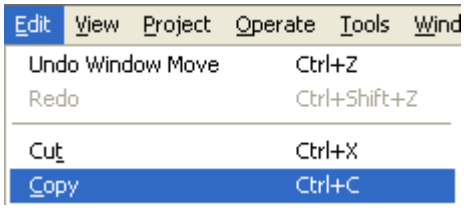


Figure 93: Copy Selected Diagram

Create a copy of the while loop and its contents by selecting **Edit >> Paste**. Organize the diagram as shown in the figure below.

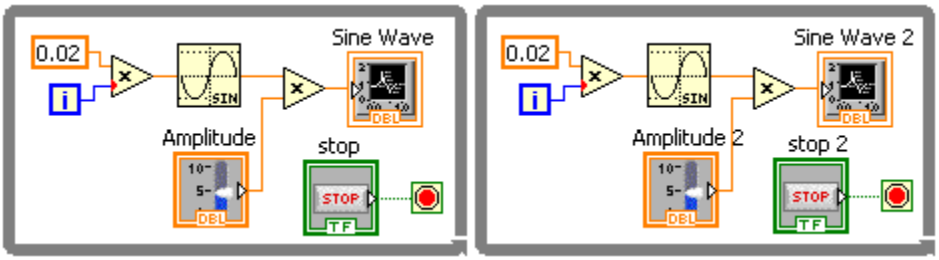


Figure 94: Paste Diagram

Go the GUI window and organize the input and output controls as shown in the figure below.

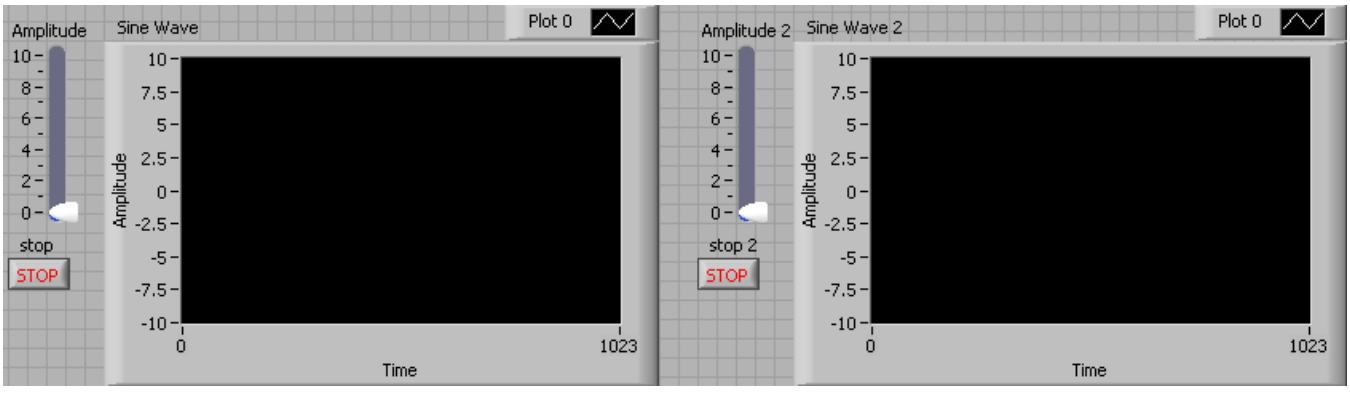


Figure 95: Parallel G Program

Congratulations!!! You have just completed your first parallel interactive program using G. Save the program, run it and interact with it. To end this program click on **stop** and **stop 2**.

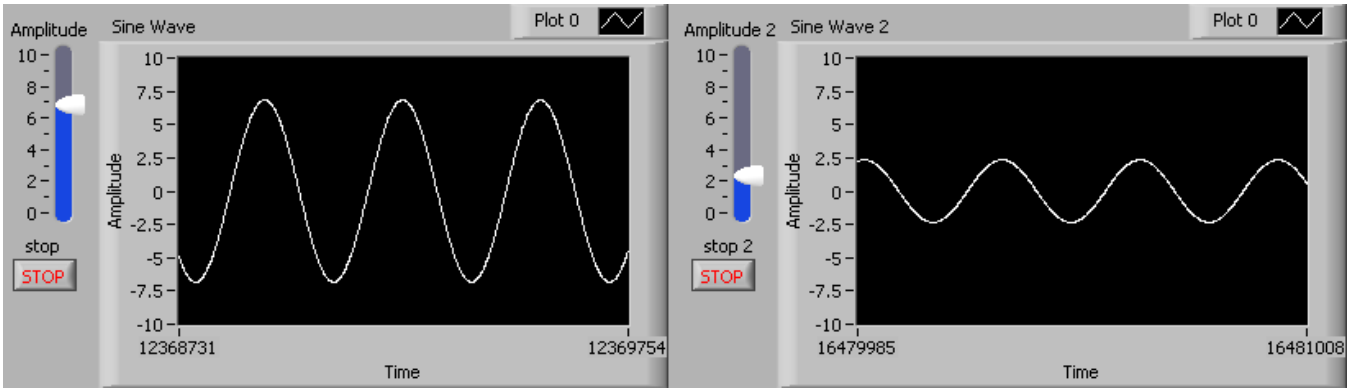


Figure 96: Parallel Interactive G Program

1.11 Multicore Programming

Save a copy of **Parallel Programming.vi** as **Multicore Programming.vi**. If you have a multicore computer, CONGRATULATIONS!!! You have just completed your first parallel interactive multicore G program.

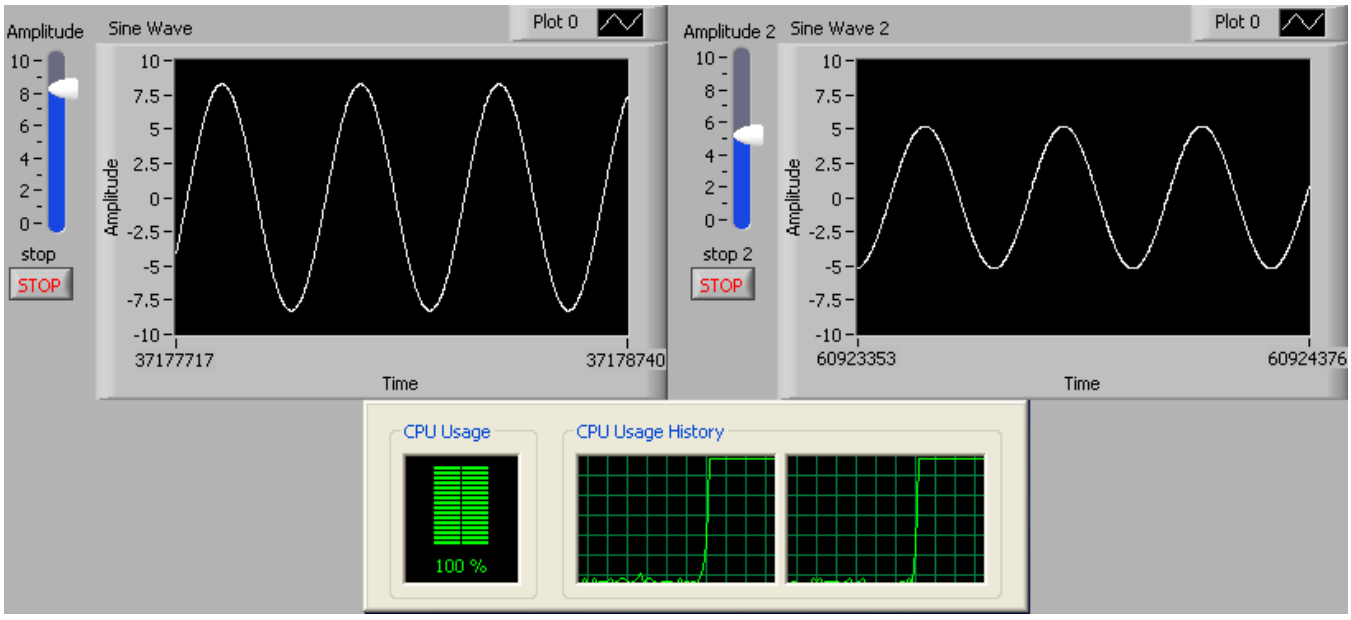


Figure 97: Interactive Multicore G Program

1.12 Polymorphism

This program shows the polymorphic properties of G. Create the G program shown below.

Notice that the **Subtract** and **Multiply** operations allow arrays to be wired in the G program.

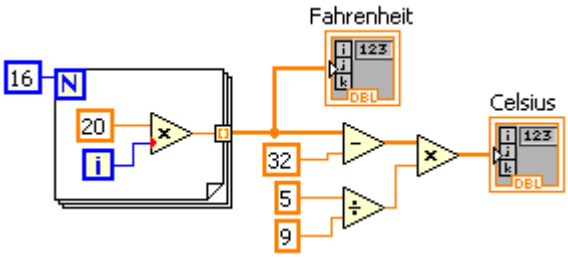
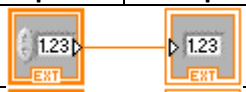
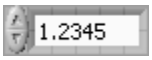
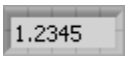
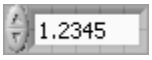
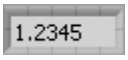
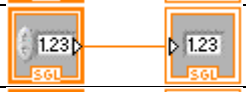
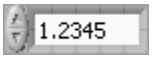
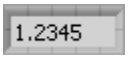
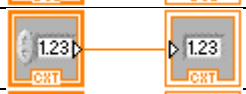
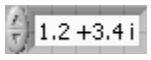
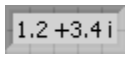
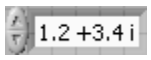
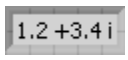
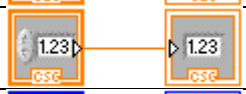
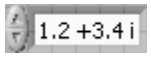
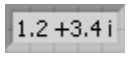
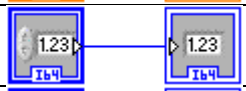
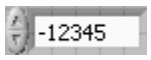
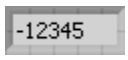
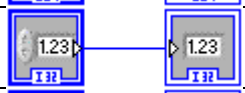
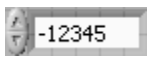
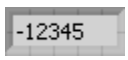
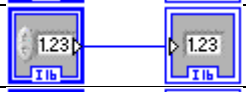
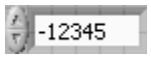
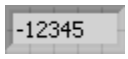
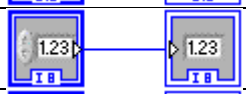
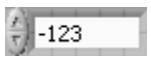
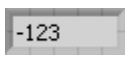
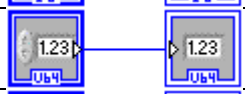
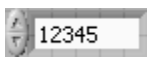
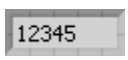
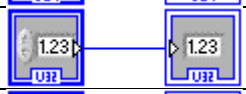
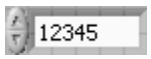
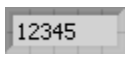
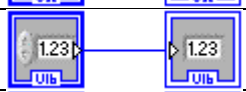
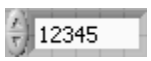
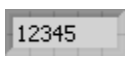
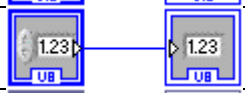


Figure 98: Polymorphic G Diagram

2 Data Types

Data Type	Block Diagram Terminal		Interactive User Interface		Comments
	Input	Output	Input	Output	
Extended					80-bit floating point numeric
Double					64-bit floating point numeric
Single					32-bit floating point numeric
Complex Extended					160-bit floating point numeric
Complex Double					128-bit floating point numeric
Complex Single					64-bit floating point numeric
Integer 64					64-bit signed integer numeric
Integer 32					32-bit signed integer numeric
Integer 16					16-bit signed integer numeric
Integer 8					8-bit signed integer numeric
Unsigned Integer 64					64-bt unsigned integer numeric
Unsigned Integer 32					32-bit unsigned integer numeric
Unsigned Integer 16					16-bit unsigned integer numeric
Unsigned Integer 8					8-bit unsigned integer numeric
Fixed Point					Mantissa.fractional fixed point numeric representation
Boolean					True or False.
String			String	String	Text

3 Operators

3.1 Numeric

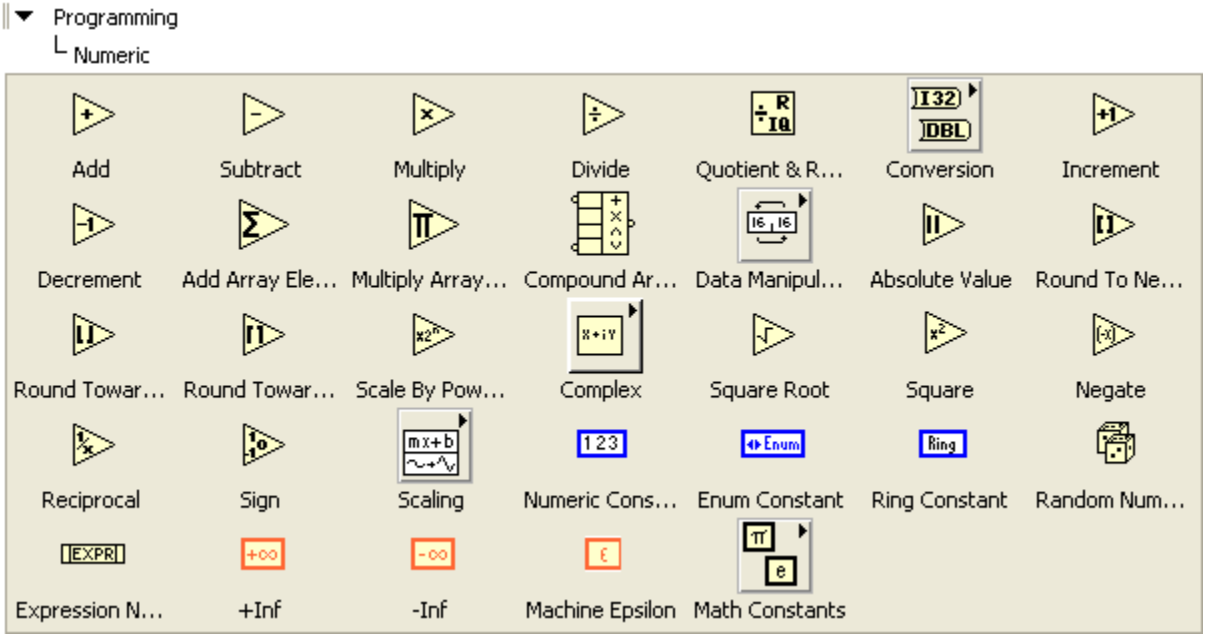


Figure 99: Numeric Operators

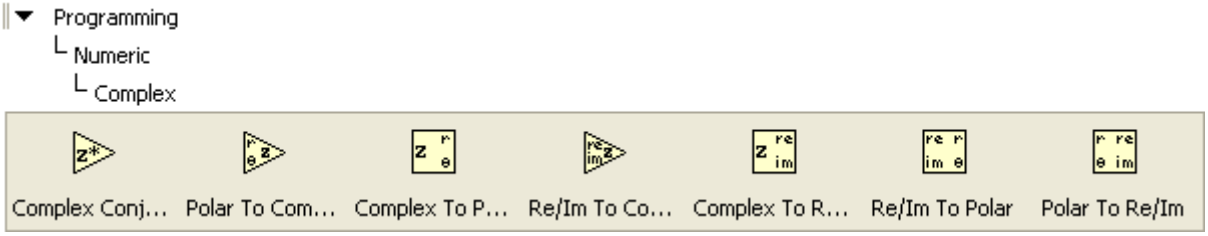


Figure 100: Complex Numeric Operations

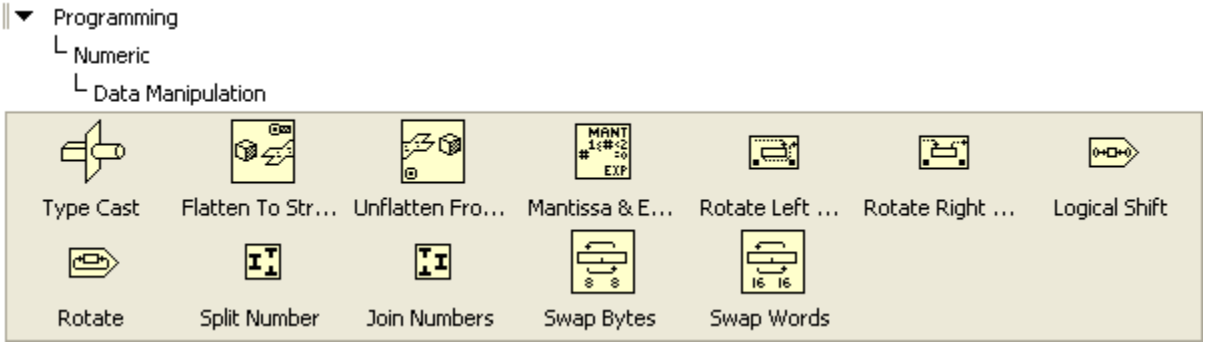


Figure 101: Numeric Data Manipulation Operators

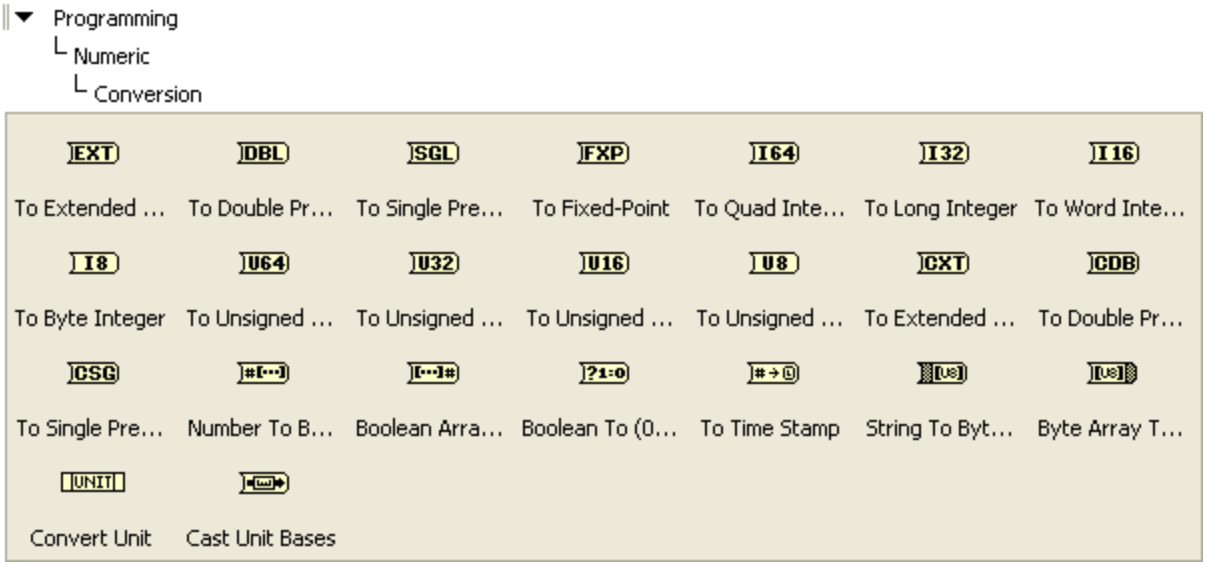


Figure 102: Numeric Conversion Operators

3.2 Boolean

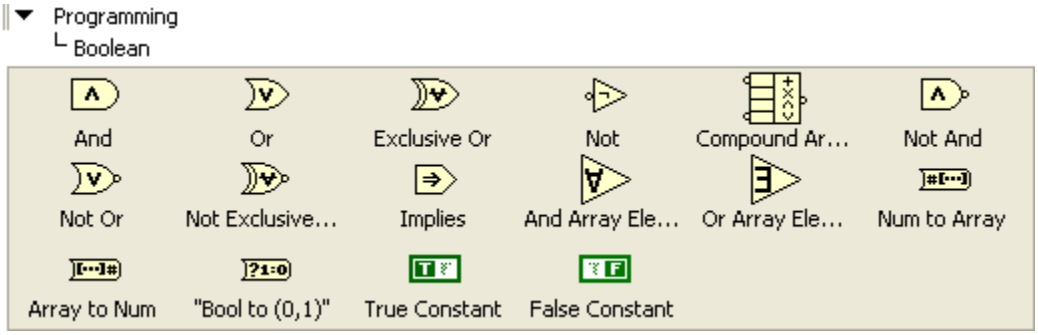


Figure 103: Boolean Operators

3.3 Comparison

Figure 104: Comparison Operators

3.4 String

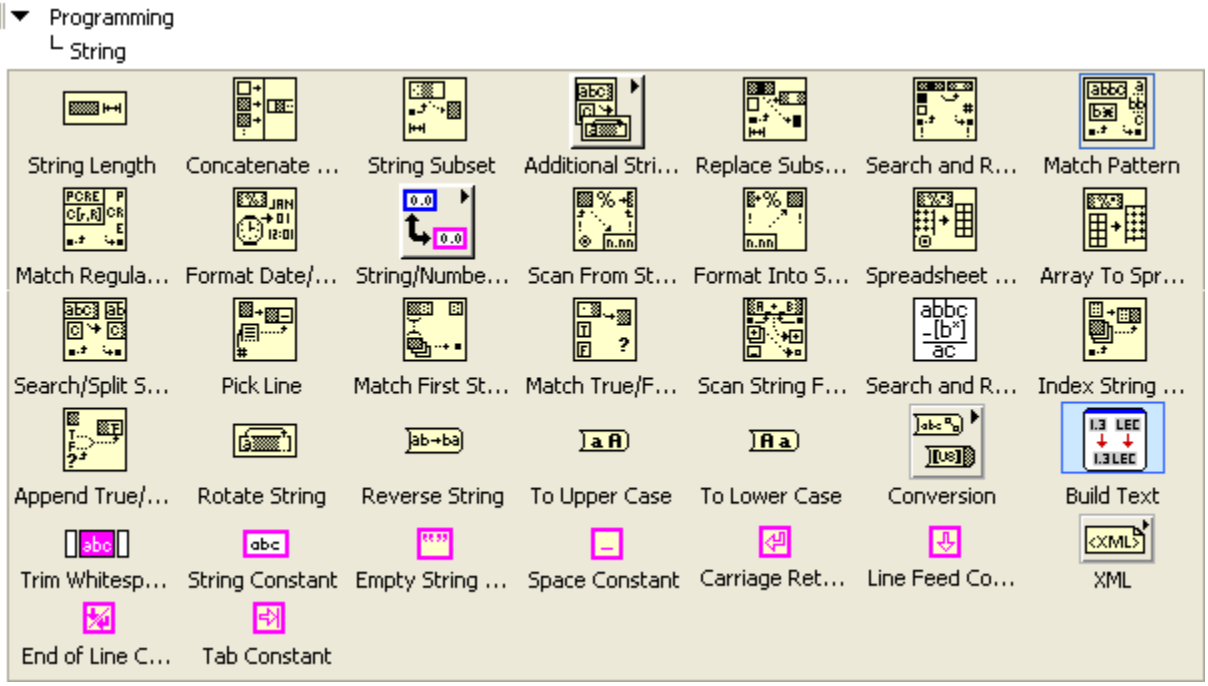


Figure 105: String Operators

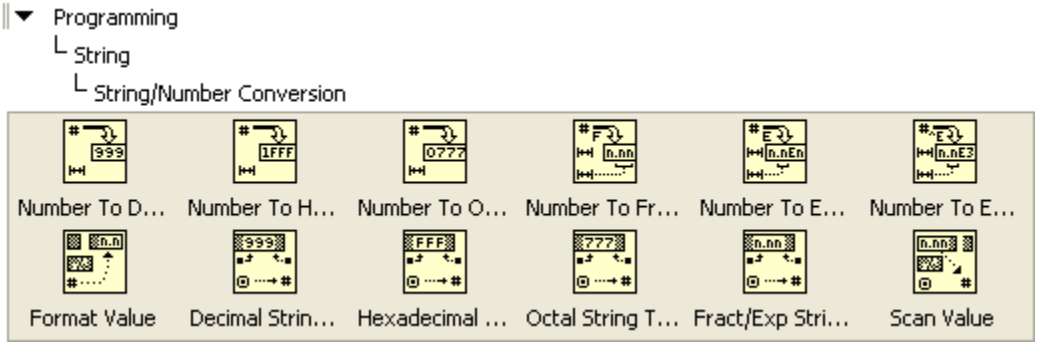


Figure 106: String/Number Operators

3.5 Math

3.5.1 Math Constants

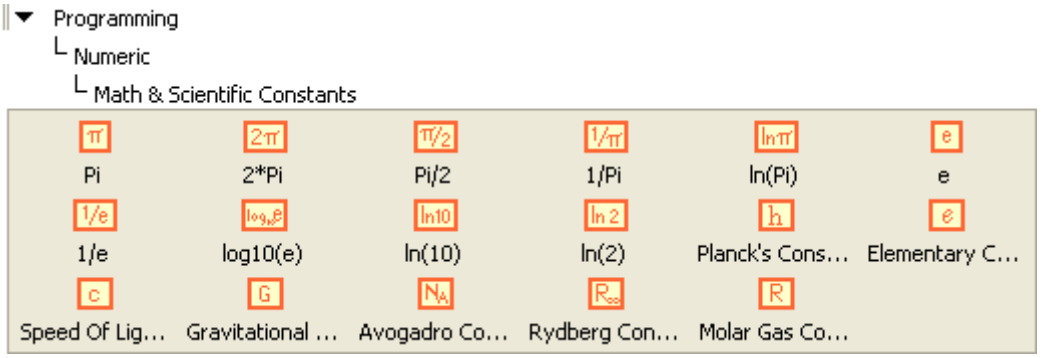


Figure 107: Mathematical Constants

3.5.2 Trigonometric Functions

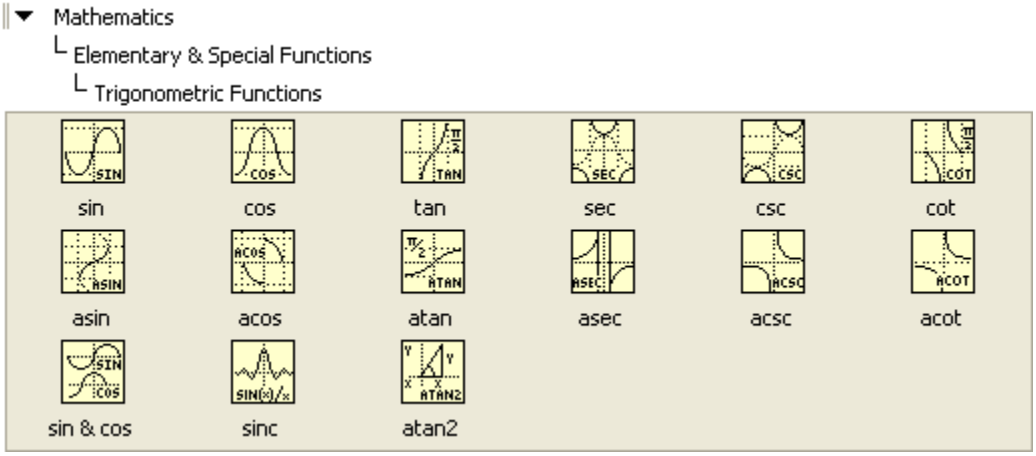


Figure 108: Trigonometric Functions

3.5.3 Exponential and Logarithmic Functions

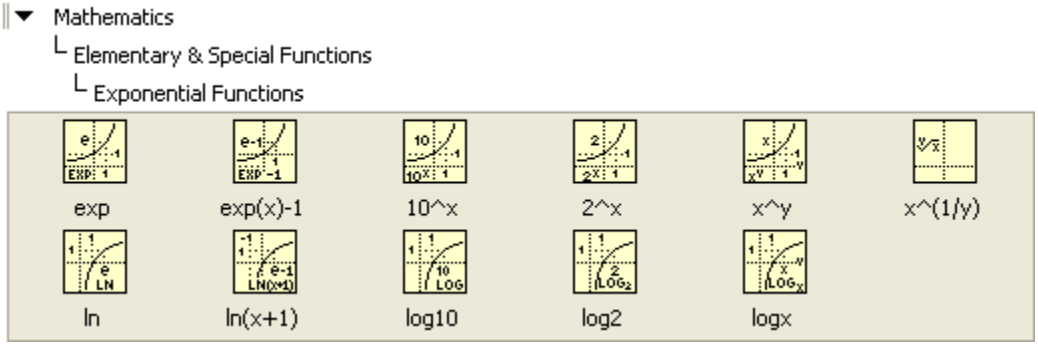


Figure 109: Exponential and Logarithmic Functions

3.5.4 Hyperbolic Functions

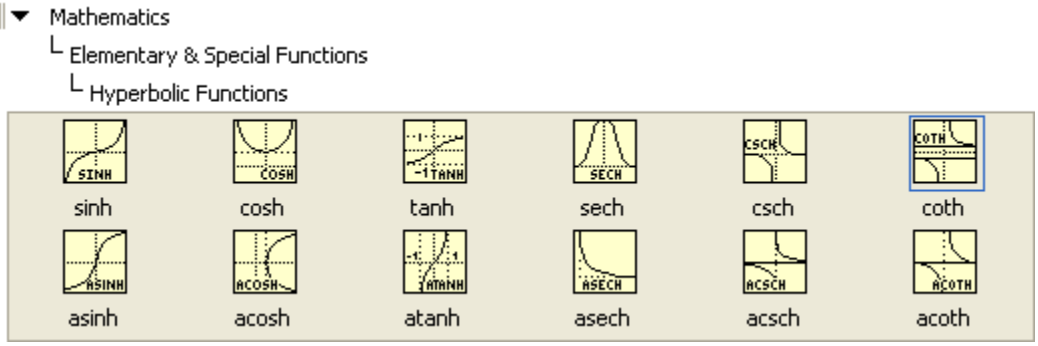


Figure 110: Hyperbolic Functions

4 Arrays and Clusters

To create an array in G, right click on the GUI window and select **Array** from the **Controls >> Modern >> Arrays, Matrix & Cluster** menu, and drop the array structure onto the GUI window to create an array.

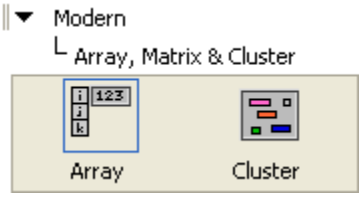


Figure 111: Array Structure

The array structure consists of an **index** or **element offset** (highlighted left portion of the array structure) and the array elements (right portion of the structure). When the array structure is placed on the GUI window, the data type of the array is undefined as indicated by the grayed out portion of the array.

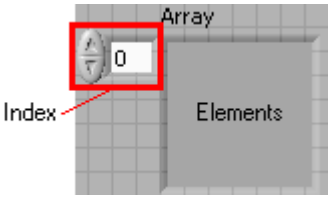


Figure 112: Index and Elements of an Array

To define the array data type, drag and drop any data type, such as numeric, Boolean, string or cluster structure, onto the **elements** portion of the array structure.

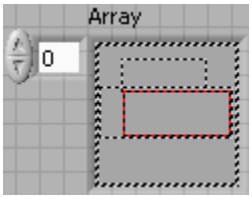


Figure 113: Creating Arrays

At this point, the newly defined array is an **Empty** or **Null Array** because no elements of the array have been defined. This is indicated by the grayed out data type within the **elements** array structure.

Figure 114: Empty Arrays

To define elements of an input array, select the element's **index** and enter the appropriate value. Figure 115 defines a numeric array with one element at **index** 0.

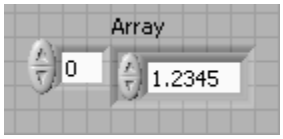


Figure 115: Defining Array Elements

G arrays are zero-based. The last element index of an **N** element array is **N-1**. Figure 116 and Figure 117 are those of a 10 element array.

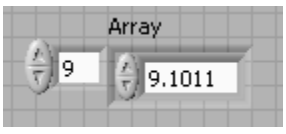


Figure 116: Last Array Element

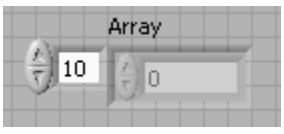


Figure 117: Undefined Nth Element

An output array is created similarly to an input array with the exception that an output data type needs to be dropped into the array structure.

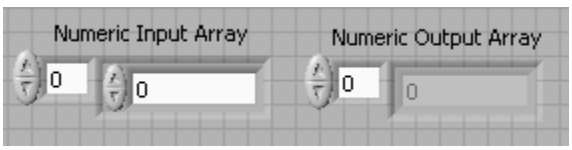


Figure 118: Creating Output Arrays

4.1 Multidimensional Arrays

To create multidimensional arrays, click on the array's **index** and select **Add Dimension** from the menu.

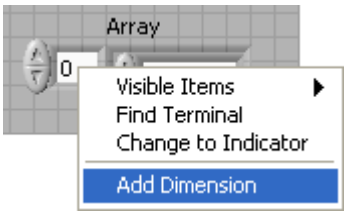


Figure 119: Creating Multidimensional Arrays

Figure 120 shows a 2-dimensional array.

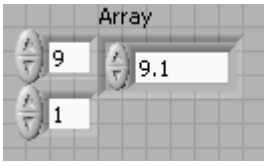


Figure 120: Multidimensional Array

4.2 Array Operators

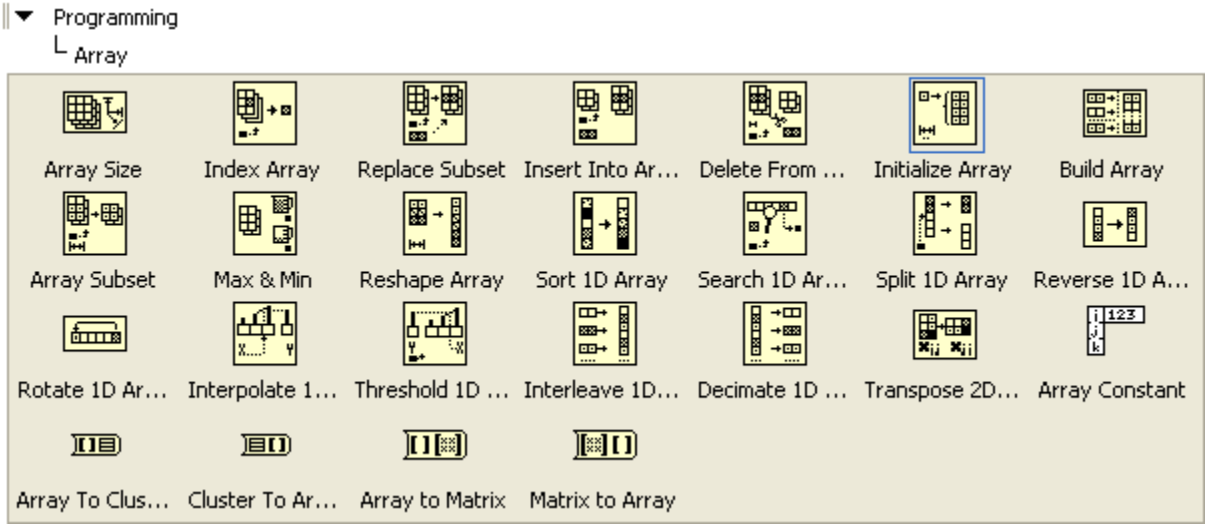


Figure 121: Array Operators

4.3 Clusters

Clusters allow users to create compound data types by aggregating various and different data types into a single unit.

Figure 122: Empty Cluster

Select the various data types and drag them onto the **cluster** structure. Figure 123 shows an **Error Cluster** consisting of a Boolean **Error**, a numeric **ID** and a string **Message** data types.

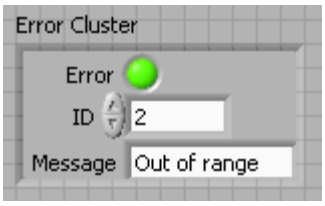


Figure 123: Cluster Example

4.4 Cluster Operators

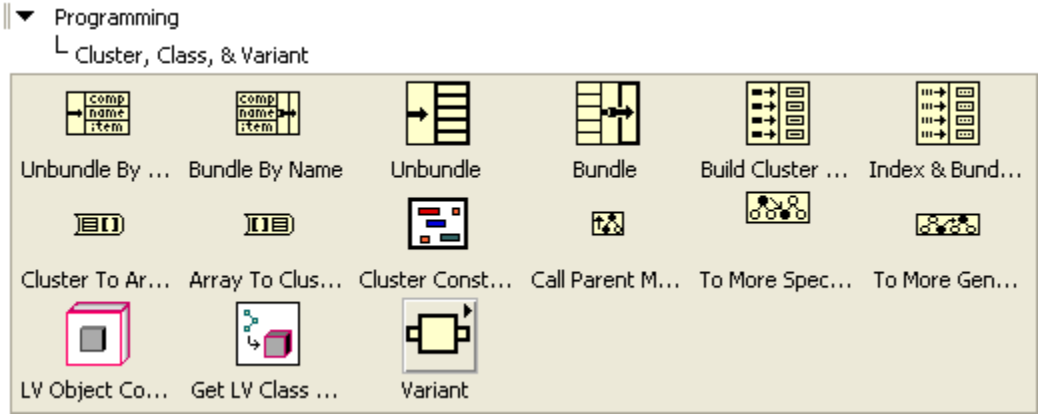


Figure 124: Cluster Operators

5 Data Flow Control

5.1 Case Structure

The case structure allows data to flow based on a integer, Boolean or string matching condition. The case executed is selected based on the data wired to the **Case Selector**.

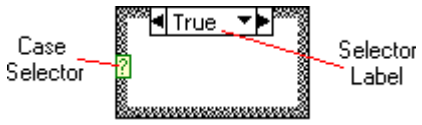


Figure 125: Case Structure

5.1.1 Boolean Selection

In the GUI window, select a Boolean control and an output string.

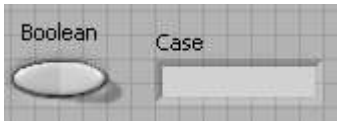


Figure 126: Case Selection User Interface

Arrange the diagram to look as in Figure 127.

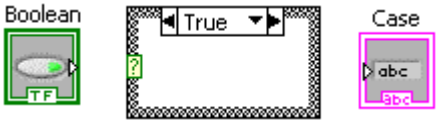


Figure 127: Case Selection G Diagram

In the **True** case, add a string constant containing **True Case**.

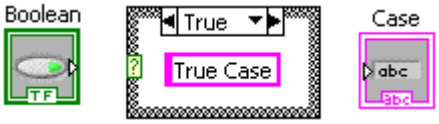


Figure 128: True Case Diagram

To select the **False** case, click on the selector label down arrow and select **False** from the pop-up menu. You can also cycle through the cases by clicking the next (right) or previous (left) arrows.

Figure 129: Selecting False Case

In the **False** case, add a string constant containing **False Case**.

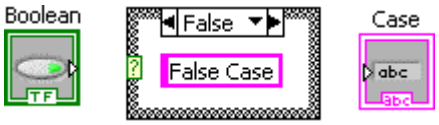


Figure 130: False Case Diagram

Wire the string constant in the **case structure** to the output string terminal.

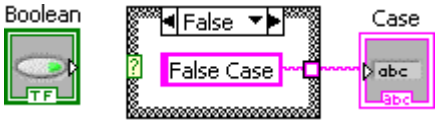


Figure 131: Wiring Case Structures

Select the **True** case and wire the string constant to the **case structure tunnel**. Complete the diagram as shown in Figure 132.

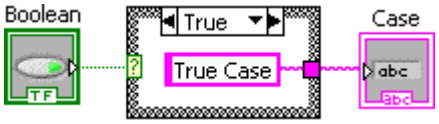


Figure 132: Completed Case Diagram

It is important to note that all instances in a **case structure** must be wired to enable data to flow from the **case structure**.

In the GUI window, toggle the Boolean input control and run the program.

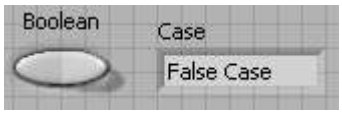


Figure 133: False Selection

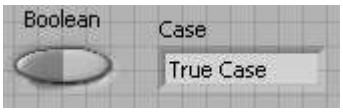


Figure 134: True Selection

5.1.2 Multicase Selection

Select an Integer 32 numeric input and an Integer 32 numeric output and label them **Selector** and **Case** respectively.

Figure 135: Multicase GUI

In the G programming window, create a **case structure**, select the **False** case and arrange the terminals as shown in Figure 136.

Figure 136: Multicase

Wire the **Selector** numeric control to the **case selector** on the case structure. The selector label reflects the diagram update.

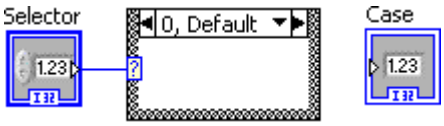


Figure 137: Multicase Selector

In the **0, Default** case, add a numeric constant and leave its value as 0.

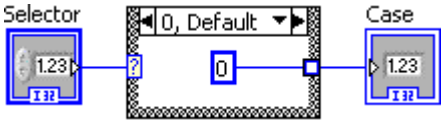


Figure 138: Default Case

Using the **selector label**, select case 1. Add a numeric constant, enter **1** and wire it to the case tunnel. The resulting diagram is shown in Figure 139.

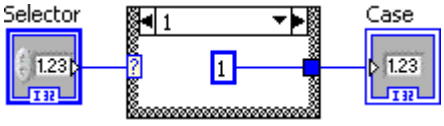


Figure 139: Case 1

Right click anywhere in the **case structure** and select **Add Case After** from the pop-up menu.

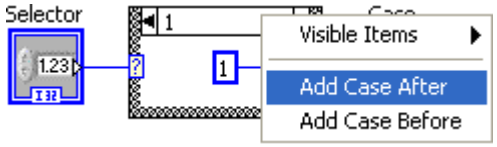


Figure 140: Adding Cases

Case 2 is added after case 1. Add a numeric constant, enter **2** and wire it to the case structure tunnel.

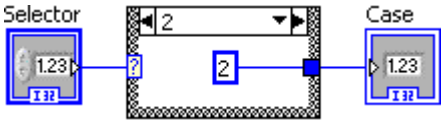


Figure 141: Case 2

Figure 142 shows the results of running this simple case selection programs for **Selector** set to 0, 1, 2 and 3 respectively.

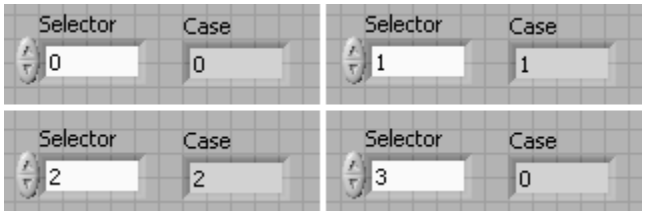


Figure 142: Multicase Selection Program

5.2 For Loop

The **For Loop** structure repeatedly executes the diagram within the structure. The **Loop Count** specifies the number of times the loop contents must be executed and the **Loop Iteration** indicates which iteration is currently being executed.

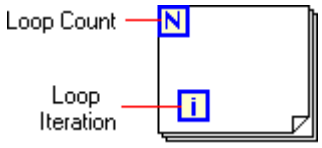


Figure 143: For Loop Structure

The **Loop Count** and **Loop Iteration** are of Integer 32 data types. If the **Loop Count** is set to **N**, then the **Loop Iteration** value range is from **0** to **N-1**. This is illustrated in Figure 144 and Figure 145.

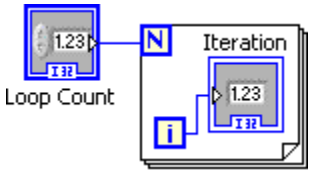


Figure 144: Loop Count

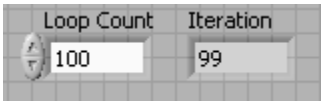


Figure 145: Final Loop Iteration

5.2.1 Shift Registers

Shift Registers allow the preservation of intermediate results between sequences of iterations.

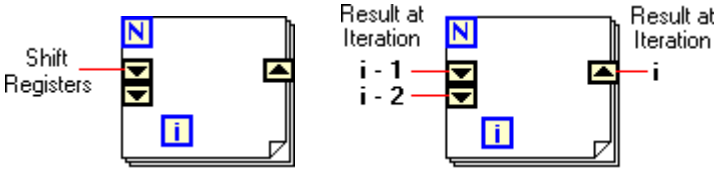


Figure 146: Shift Registers

To add a **Shift Register**, right click on the **For Loop** structure and select **Add Shift Register** from the pop-up menu.

Figure 147: Adding Shift Registers

To add elements to the **shift register**, right click on the **shift register** and select **Add Element** from the pop-up menu.

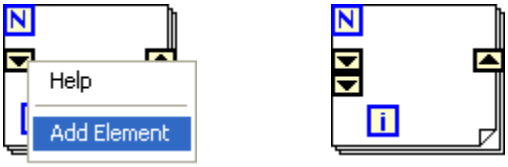


Figure 148: Adding Shift Register Elements

To illustrate the use of the **shift registers**, the following example computes the Fibonacci number Fib(n).

$$Fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ Fib(n-1) + Fib(n-2), & n > 1 \end{cases}$$

In the GUI window, select an integer 32 numeric input and output controls and labeled them **n** and **Fib(n)** respectively. Arrange the diagram as shown in Figure 149.

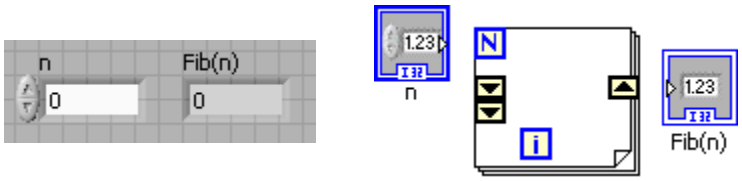


Figure 149: Shift Register Example

Add a **0** and **1** numeric constants to initialize the elements of the shift register and wire them to the **i-1** and **i-2** elements respectively. Add the **add** operator in the for loop and complete the program wiring as shown in Figure 150.

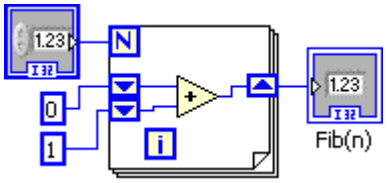


Figure 150: Fibonacci G Program

For **n** = 0, the for loop iterates 0 times and passes 0 to **Fib(n)**, therefore Fib(0)=0. For **n** = 1, the for loop the values in **i-1** and **i-2** shift register elements are added (0+1) and saved in the **i** shift register element (1). Since the loop iterates once only, the resulting value is passed to **Fib(n)**, therefore Fib(1)=1. For **n** = 2, the first iteration produces the value of 1. Prior to the next and final iteration, the values are shifted in the register as follows:

- The value in the **i-2** shift register element is discarded
- The value in the **i-1** shift register element is shifted to the **i-2** shift register element
- The value in the **i** shift register element is shifted to the **i-1** shift register element

To start the 2nd and final iteration, the **i-1** shift register element contains 1 and the **i-2** shift register element contains 0. These are added to produce 1, which is passed to **Fib(n)** and, therefore, Fib(2)=1. This process is repeated for values of **n** > 2.

Save this program as **Fibonacci.vi**. Figure 151 shows the result of Fib(8).

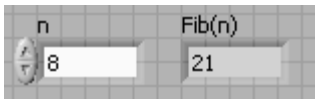


Figure 151: Fib(8) = 21

5.2.2 Auto-Indexing

Auto-indexing allows input array elements to be operated on and output array elements to be aggregated automatically in a for loop. It is not required to wire the **Loop Counter**. The for loop automatically reduces the array dimensionality by one.

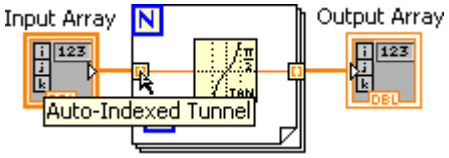


Figure 152: For Loop Auto-Indexing

5.2.3 Disabling Auto-Indexing

It is sometimes necessary to disable auto-indexing. In this example, the **For Loop** is used to scan the elements of the array taking advantage of the auto-indexing feature. However, the result is a single number. Wiring the result through the **For Loop** with auto-indexing enabled results in a broken data type wire.

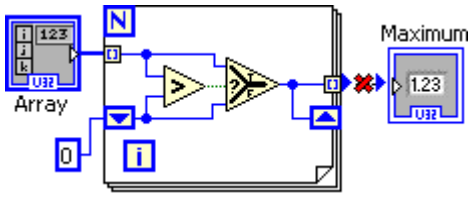


Figure 153: Broken Auto-Indexing

To disable auto-indexing, right click on the target **Auto-Indexed Tunnel** and select **Disable Indexing** from the pop-up menu.

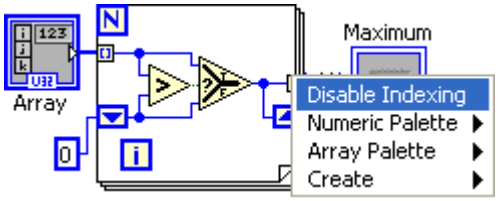


Figure 154: Disabling Auto-Indexing

The final diagram with the **Auto-Indexed Tunnel** disabled is shown in Figure 155.

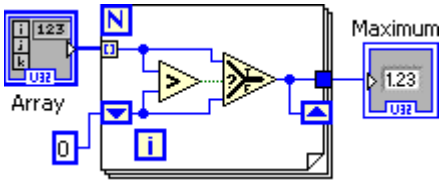


Figure 155: Disabled Auto-Indexing

5.3 While Loop

The **While Loop** conditionally iterates executing the statements within the structure. The **Loop Condition** establishes whether the loop iterates or terminates. The **Loop Iteration** is a zero-based iteration execution reference similar to the For Loop.

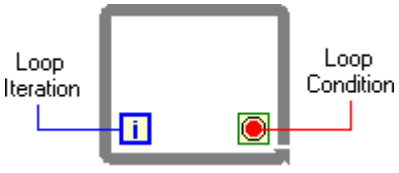


Figure 156: While Loop Structure

5.3.1 Loop Condition

5.3.1.1 Stop if True

The default **loop condition** is to continue if the Boolean condition is **False** (or stop if **True**). The **while loop** in Figure 157 will iterate while **Iterations** is less than **Loop Iteration** is **False** or, equivalently, will stop iterating when **Iterations** is less than the value in **Loop Iteration.**

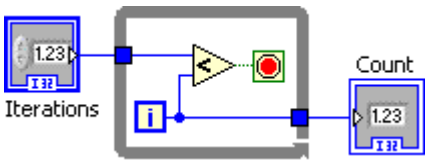


Figure 157: Stop If True

5.3.1.2 Continue if True

At times it is more convenient to let the **while** loop iterate while the condition is True. To change the **loop condition**, right click on the **loop condition** icon and select **Continue if True** from the pop-up menu.

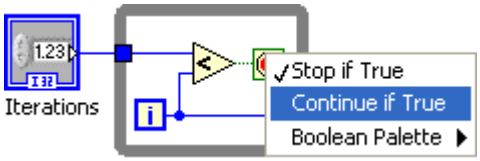


Figure 158: Changing Loop Condition

Figure 159 shows the **Loop Condition** set to **Continue if True**.

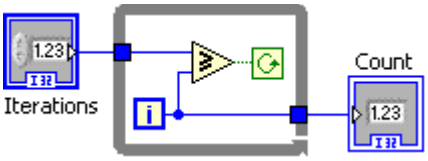


Figure 159: Continue If True

5.3.2 Shift Registers

Programmatically, while loop **shift registers** are identical to for loop **shift registers**. Refer to tion **5.2.1 Shift Registers** for the discussion. However, an example is provided to illustrate the use of shift registers in while loops.

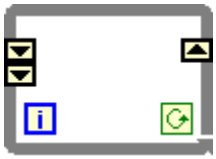


Figure 160: While Loop Shift Registers

In the following example, Euler's number **e** is computed to the specified accuracy using the infinite series

$$e = \sum_{n=0}^{n=\infty} \frac{1}{n!} = 1 + \sum_{n=1}^{n=\infty} \frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots = 2.7182818284$$

Notice that two shift registers keep track of the factorial and the sum. Also notice the dot in the multiplication. This is because the **loop iteration** is an integer 32 data type and the input from one of the shift registers is double precision numeric. The dot represents that the integer 32 data type has been coerced into a double precision number.

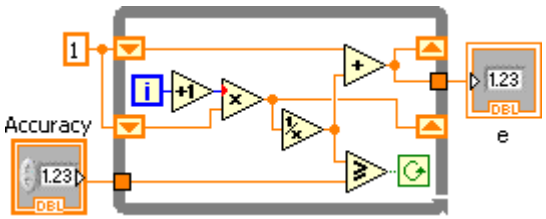


Figure 161: Computing e

Save the program as **e.vi**. The result of running this program is shown in Figure 162.

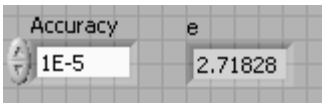


Figure 162: Computed e to 5 Digits

5.3.3 Enabling Auto-Indexing

By default, while loops are auto-indexed disabled. In order for while loops to process and generate arrays, the loop tunnel must be enabled to auto-indexed arrays.

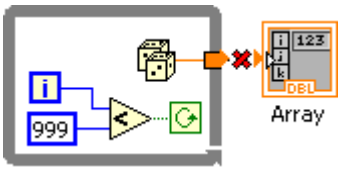


Figure 163: Disabled Auto-Indexing

To enable auto-indexing, right click on the **loop tunnel** and select **Enable Indexing** from the pop-up menu.

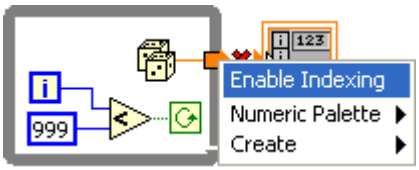


Figure 164: Enabling Auto-Indexing

In this example the while loop appropriately generates a 1,000 element numeric array with random numbers.

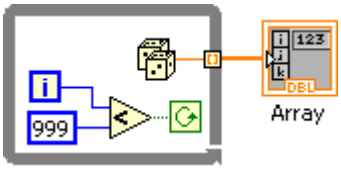


Figure 165: Auto-Indexing Enabled

5.4 Sequence

Although G was designed to easily develop interactive, parallel programs, it is sometimes necessary to execute diagrams in sequential order. The **sequence structure** allows G diagrams to execute sequentially.

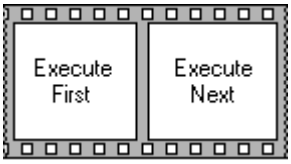


Figure 166: Sequence Structure

The following examples time in milliseconds (ms) the execution of a G diagram. The sequence of events is get a start time stamp, execute the diagram, get stop time stamp and take the difference between the stop and start times to determine the execution time.

5.4.1 Flat Sequence

Flat Sequences always execute left to right. A **Flat Sequence** structure starts with a single frame and allows a user to visualize the diagram sequences.

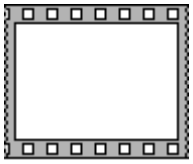


Figure 167: Sequence Frame

To add frames to a sequence, right click on the sequence structure and select either **Add Frame After** or **Add Frame Before** from the pop-up menu according to the program's needs.

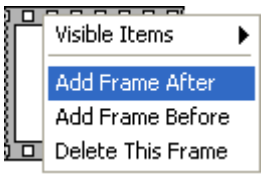


Figure 168: Adding Sequence Frames

Add two more frames to the sequence structure to get a three frame sequence as shown in Figure 169.

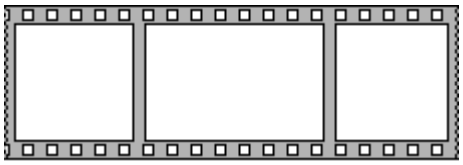


Figure 169: Three Frame Sequence

From the **Functions >> Programming >> Timing** menu select **Tick Count (ms)** function.

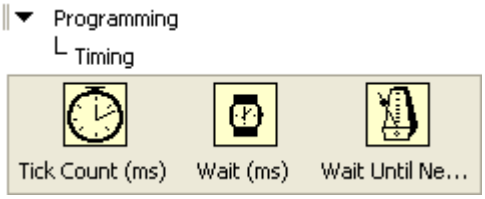


Figure 170: Tick Count Function

Drop the **Tick Count (ms)** function in the first (left most) frame of this sequence. Make a copy of the **Tick Count** function and place it on the third (right most) frame as shown in Figure 171.

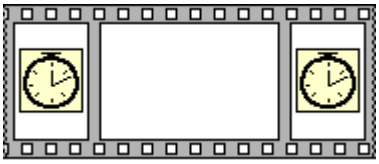


Figure 171: Start and Stop Tick Counts

Add a **For Loop** that iterates 5,000 times to the second frame. Add a **subtract** operator, an signed integer 32 output and complete the program as shown in Figure 172. The execution of this program shows the time in milliseconds it took for the 2nd sequential frame to execute.

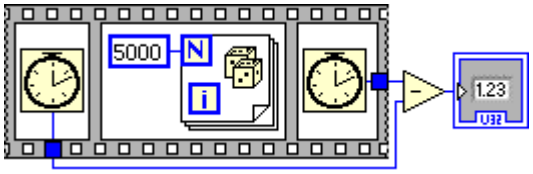


Figure 172: Timing G Program

5.4.2 Stacked Sequence

A **Stacked Sequence** provides a more compact presentation of program sequences. It is programmatically identical to the **Flat Sequence** with the exception that a **Sequence Local** enables data to flow to subsequent frames. Additionally, as frames are added, a **Sequence Selector** provides access to the desired frame (see Figure 173).

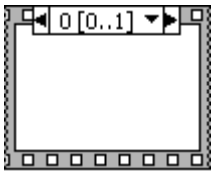


Figure 173: Stacked Sequence

For this timing example, start with a **Stacked Sequence** and add 3 more frames. The sequence frames are labeled 0, 1, 2 and 3 and will execute in that order.

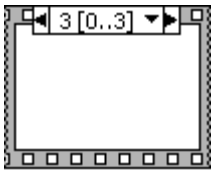


Figure 174: Four Frame Stacked Sequence

Go to the first frame (frame 0) and add a **Tick Count (ms)** function. Right click on the sequence structure and select **Add Sequence Local** from the pop-up menu.

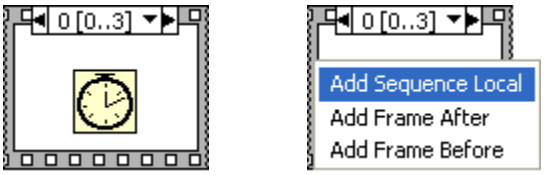


Figure 175: Adding Sequence Locals

The **Sequence Local** is shown as an undefined tunnel. Wire the **Tick Count (ms)** function to the **Sequence Local** to define the tunnel data type and data flow. Data can now flow from frame 0 to the other frames as needed.

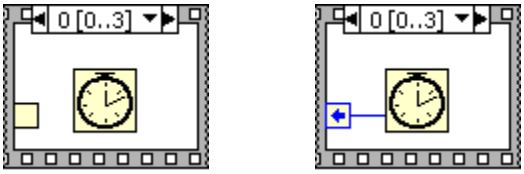


Figure 176: Sequence Local

Go to the next frame sequence (frame 1) and enter the program to be timed.

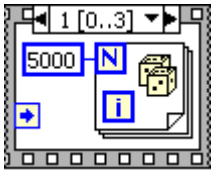


Figure 177: Frame to Time

Go to the third frame of the sequence (frame 2), add a **Tick Count (ms)** function, add another **Sequence Local** and wire the **Tick Count (ms)** to the new **Sequence Local**. The wired sequence frame is shown in Figure 178.

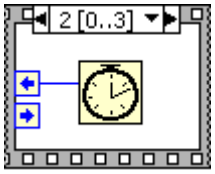


Figure 178: Stop Time Stamp

Go to the last frame (frame 3) and add a **Subtract** function. Wire the **Sequence Locals** from frame 2 and frame 0 to the **Subtract** function as shown in Figure 179. To complete the diagram, wire the output of the **Subtract** function to the unsigned integer 32 output.

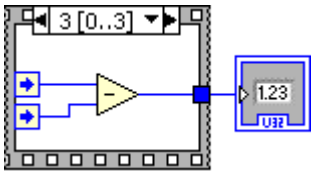


Figure 179: Stacked Timing G Program

It is important to note that the programs in Figure 172 and Figure 179 are programmatically identical.

6 Functions

Any G program can become a function. Three operations must be done:

1. Edit connecting input and/or output terminals
2. Edit the icon (optional but recommended)
3. Save the G program

For this example open the **Fibonacci.vi**.

6.1 Connectors

On the GUI window, right click on the icon located on the right upper corner of the window and select **Show Connector**.

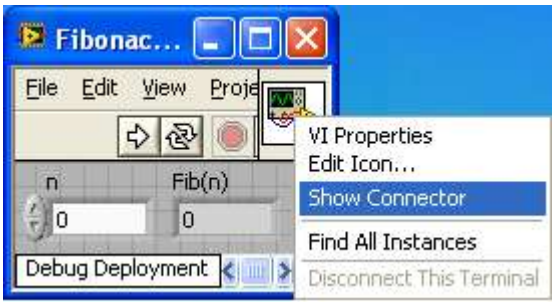


Figure 180: Show Connector Pane

This brings up the connector pane as shown in Figure 181.

Figure 181: Connector Pane

Right click on the connector pane and select **terns**. A menu with connector patterns is sented from which you can select the appropriate pattern. For this example select the pattern highlighted in Figure 182.

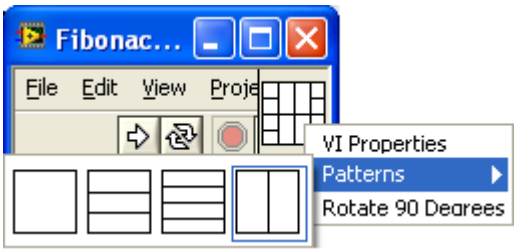


Figure 182: Select Connector Pattern

Click on the connector terminal followed by a click on the input or output control to which the terminal is to be associated. In Figure 183, the left connector terminal is associated with the numeric input control **n**.

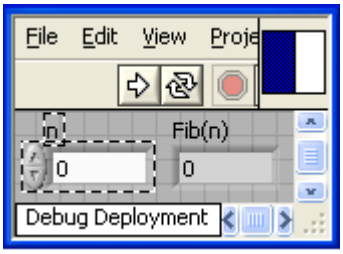


Figure 183: Associating Terminals

Repeat for all the input and output controls that are to be associated to the terminals. For the **bonacci.vi**, Figure 184 shows the right connector terminal associated with the **Fib(n)** output terminal.

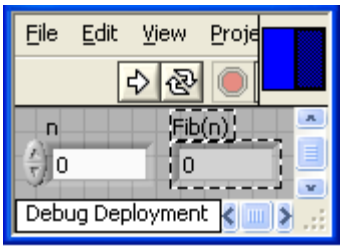


Figure 184: Connected Terminals

6.2 Icon Editor

Right click on the connector pane and select **Edit Icon…** from the pop-up menu. This will bring the icon editor (Figure 186).

Edit the icon for black and white, 16-color and 256-color displays and click **OK** when completed.

Save the G program to complete the function.

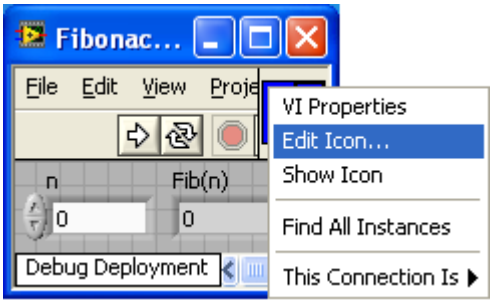


Figure 185: Selecting Icon Editor

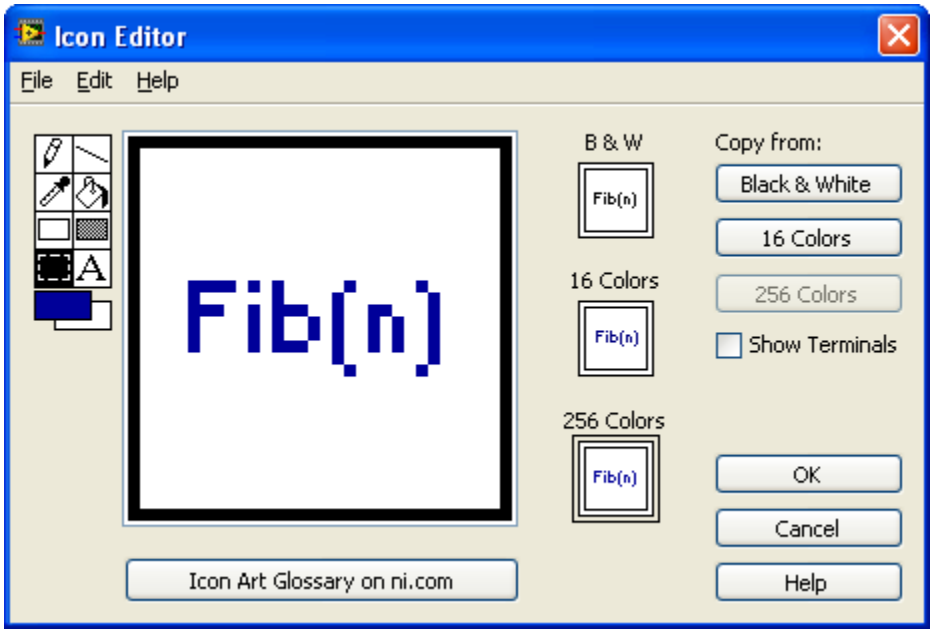


Figure 186: Icon Editor

6.3 Invoking Functions

To invoke functions, right click on the G programming window and select **Select a VI…** from the pop-up menu. This will bring a file dialog box. Find the desired function to be part of the program and click **OK**.

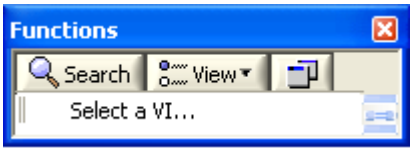


Figure 187: Invoking Functions

In the example shown in Figure 188, the Fibonacci series of the first 20 Fibonacci numbers is stored in an array. The numbers are computed by invoking the **Fibonacci.vi** function.

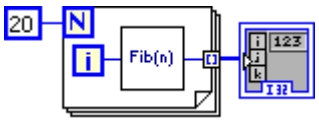


Figure 188: Fibonacci Series

7 Graphs

7.1 Waveform Chart

Waveform Charts provide a historical graphical representation of numeric data.

The following example will build a simple G program that will allow you to chart a sine wave as it is being generated on a point-by-point basis using the equation:

$$y_i = \sin(0.2 \times i)$$

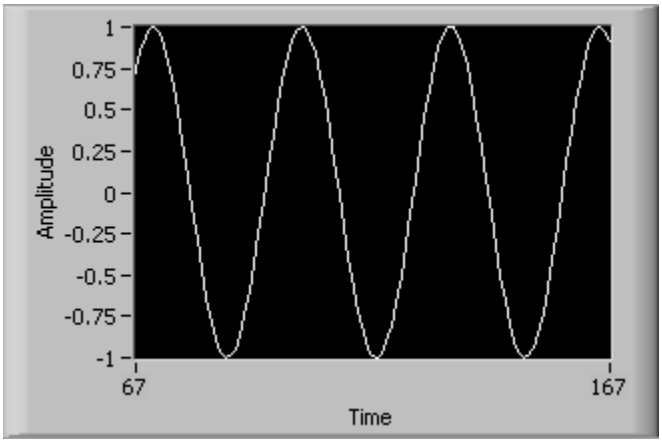


Figure 189: Waveform Chart

Start with a while loop and add into it a **Multiply** and **Sine** functions, a numeric constant with value 0.2 and a Boolean control to stop the loop when its value is **True**. Arrange the diagram to look as in Figure 190.

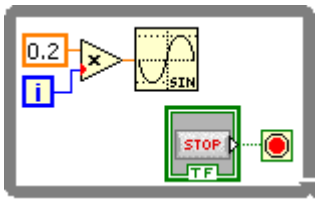


Figure 190: While Loop For Waveform Chart

To select a **waveform chart**, right click on the GUI window and select **Waveform Chart** from the **Controls >> Modern >> Graph** menu.

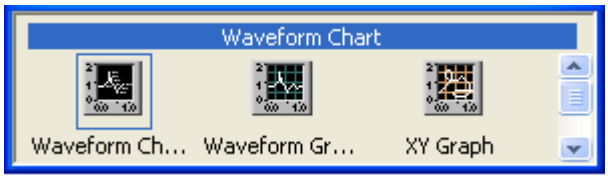


Figure 191: Selecting Waveform Chart

This places the **Waveform Chart** in the GUI window.

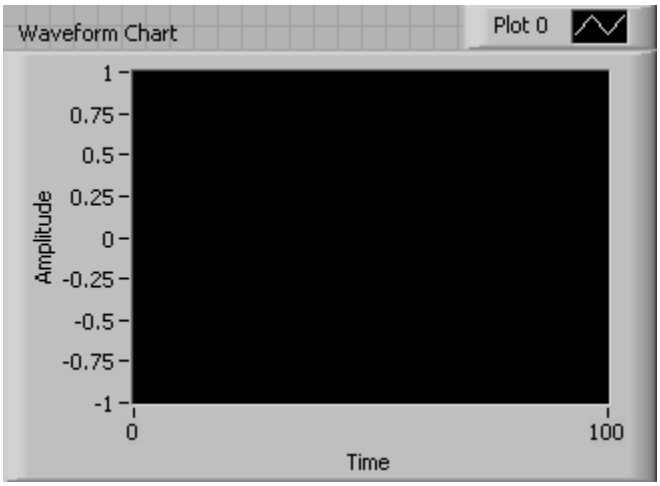


Figure 192: Waveform Chart in GUI Window

In the G programming window, make sure that the **Waveform Chart** terminal is inside the while loop. Wire the output of the **Sine** function to this terminal.

Notice that **Waveform Chart** terminal is that of a numeric output.

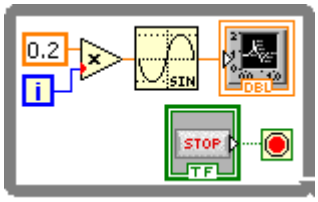


Figure 193: Waveform Chart Terminal

Most modern computers will run this program too fast. Thus, before this program is executed, a delay of 125 milliseconds will be inserted in the while loop. This will allow users to see how the **Waveform Chart** operates as data samples are plotted in the chard.

From the **Functions >> Programming >> Timing** select **Wait Until Next ms Multiple**. This will put the while loop to sleep for the indicated number of milliseconds.

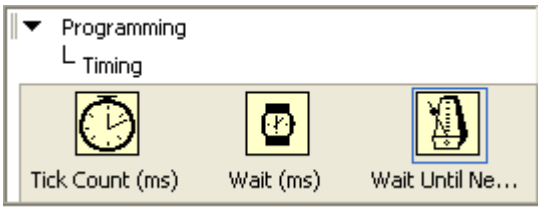


Figure 194: Wait Until Next ms Multiple

Drop the **Wait Until Next ms Multiple** function inside the loop and wire a constant to it with the value 125. This will delay the loop for 125 milliseconds. The final **Waveform Chart** program is shown in Figure 195.

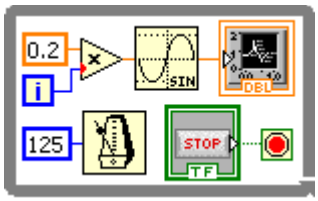


Figure 195: Waveform Chart Program

The default graphing mode of the **Waveform Chart** is **autoscaling**. You will notice the autoscaling property when the program first begins to run and the y-axis, labeled **Amplitude**, updates automatically as new numerical values are aggregated and displayed on the chart.

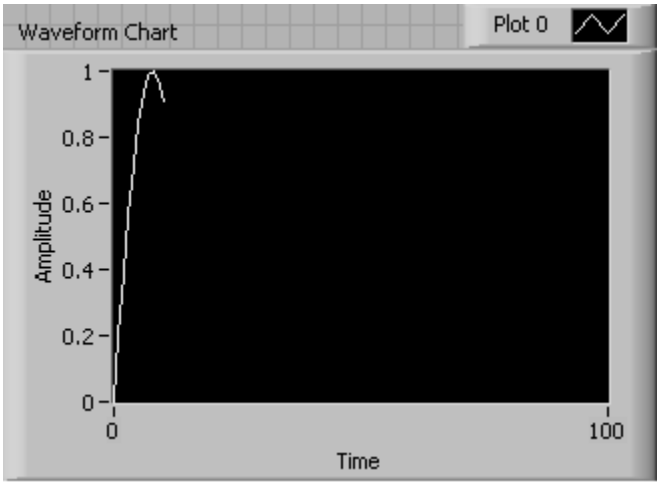


Figure 196: Waveform Chart Autoscaling

As the program continues to run, the graph continues to build as per the values associated with the x-axis, labeled **Time**, which correspond to the index value of the equations.

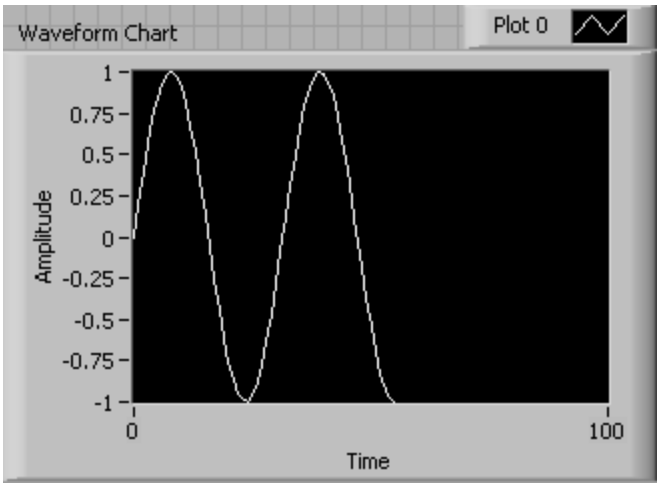


Figure 197: Accumulating Values for the Waveform Chart

As the program continues to run, the autoscaling property also applies to the x-axis. Noticed the updated x-axis. For this example, the x-axis will continue updating so as long as the program is running. This gives the appearance of a scrolling strip chart.

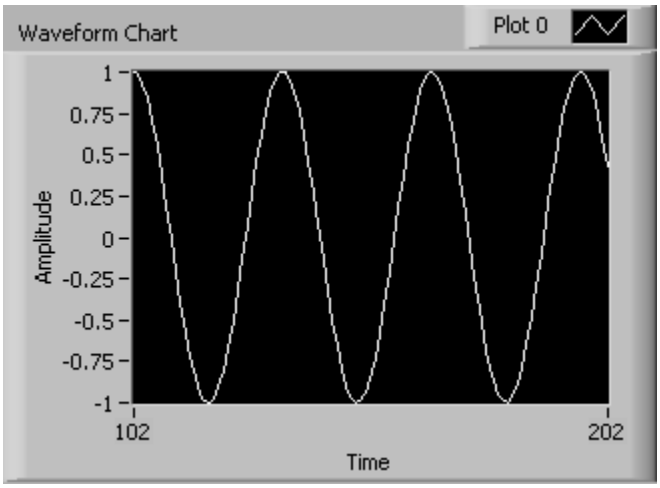


Figure 198: Scrolling X-Axis

Stopping and restarting the G program retains the numeric history and continues to aggregate the values for display.

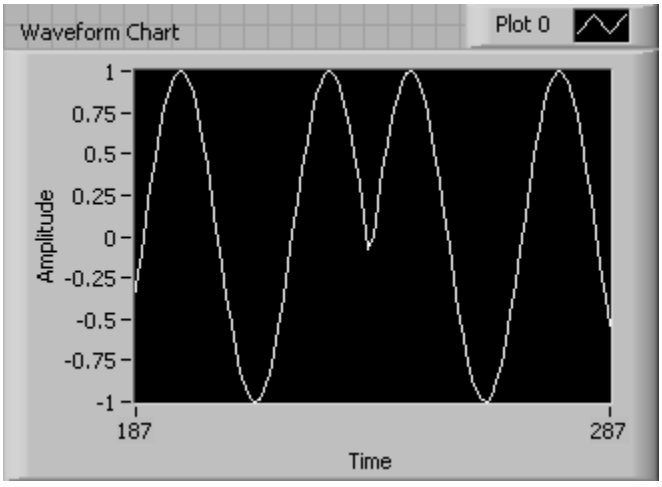


Figure 199: Graph History Retained Between Runs

The **Waveform Chart** options can be easily updated by right clicking on the **Waveform Chart** and selecting the appropriate option to update from the pop-up menu.

Selecting **Properties** from this pop-up menu brings up the **Waveform Chart** dialog window.

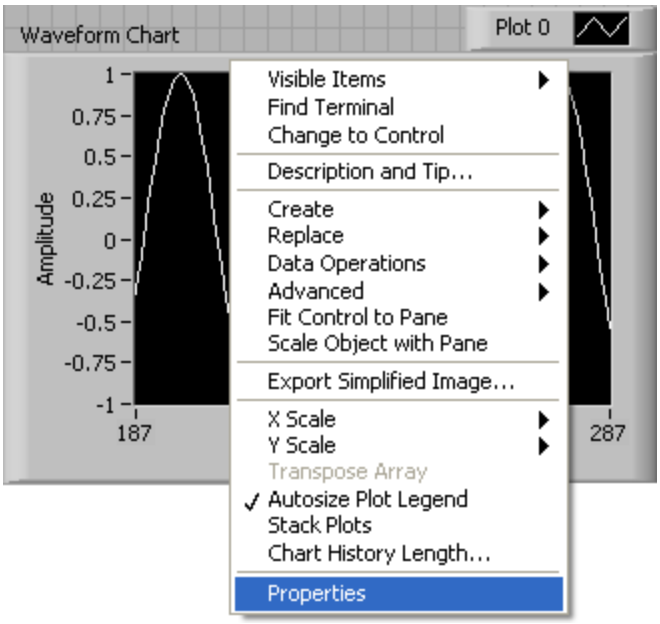


Figure 200: Waveform Chart Pop-Up Menu

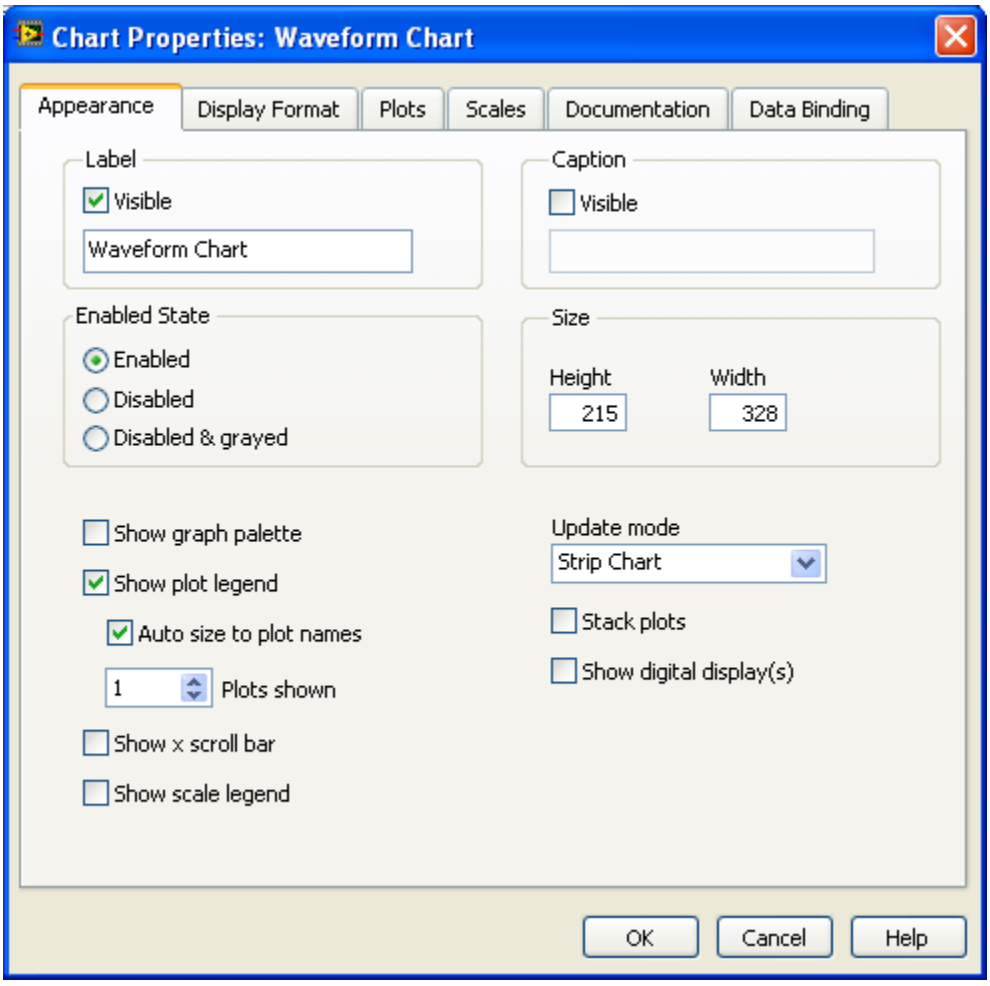


Figure 201: Waveform Chart Options Dialog Box

7.2 Waveform Graph

The **Waveform Graph** allows numeric arrays to be displayed graphically in the GUI window.

Similar to the previous example, we will build a simple G program that will allow you to graph a sine wave using the equation:

$$y_i = \sin(0.2 \times i)$$

for i=0, 1, 2, … , 99.

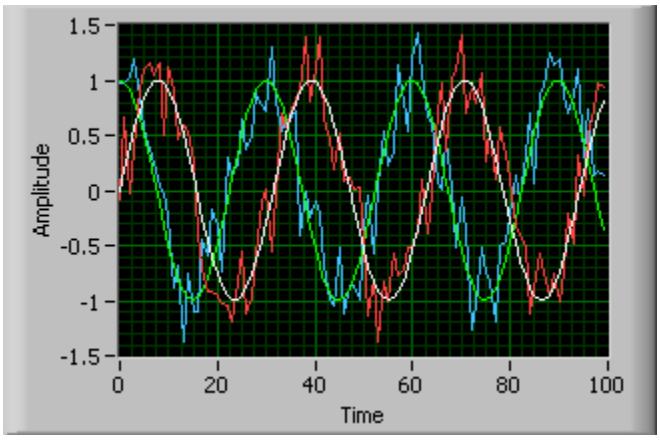


Figure 202: Waveform Graph

7.2.1 Single Plot

Start by building the following program shown in Figure 203.

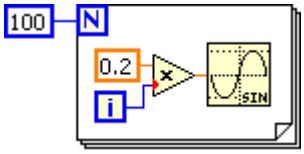


Figure 203: For Loop Sine Wave

Right click on the GUI window, select **Waveform Graph** from the **Modern >> Graph** pop-up menu, and drop it on the GUI window.

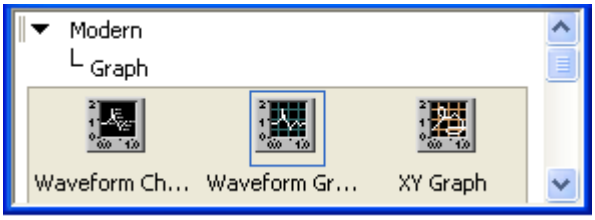


Figure 204: Select Waveform Graph

In the G programming window you will see the **Waveform Graph** terminal. Wire the **Sine** function output to the **Waveform Graph** terminal through the **For Loop**.

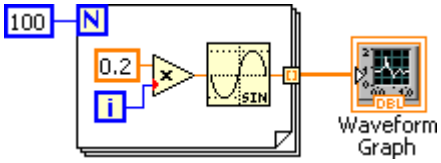


Figure 205: Waveform Graph Diagram

Run the program. The resulting graph is shown in Figure 206.

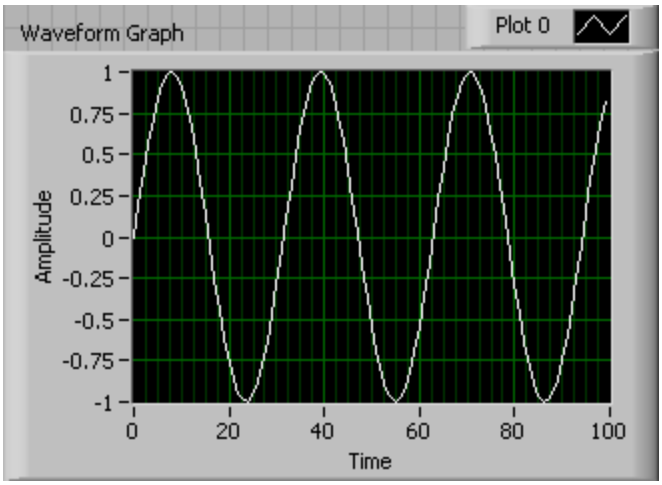


Figure 206: Sine Wave Graph

7.2.2 Multiplots

In this example a sine wave and a noisy sine wave will be plotted. Modify the previous example to add noise to the sine operation as shown in Figure 207.

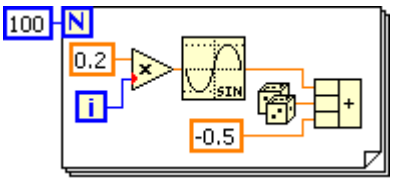


Figure 207: Sine and Noisy Sine Waveforms

Add a **Build Array** operator and wire the output of the **Sine** function and the multi-add operator containing the sine value plus some random noise between -0.5 and 0.5 to the **Build Array** operator. Wire the output of the **Build Array** operator to the **Waveform Graph** terminal.

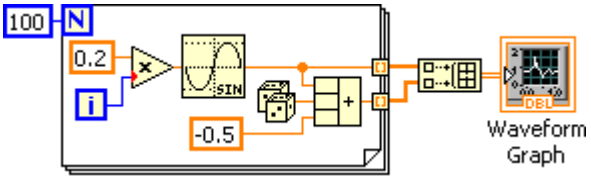


Figure 208: Bundle Arrays for Multiplotting

You can continue adding 1D arrays to be multiplotted into a single **Waveform Graph**.

Run the program. The multiplot result is shown in Figure 209.

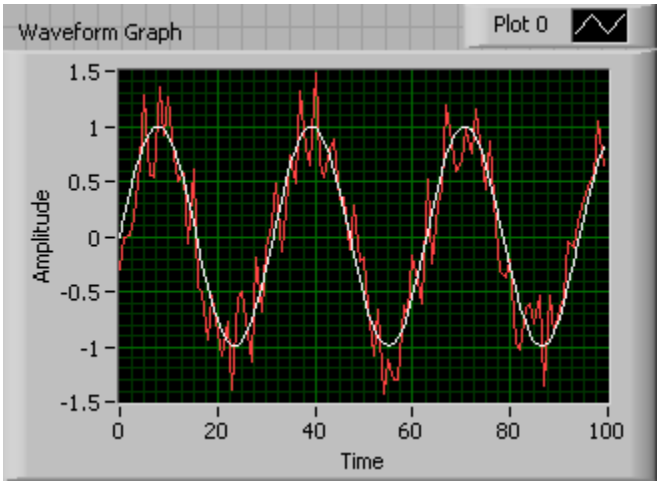


Figure 209: Multiplot

7.3 XY Graph

The **XY Graph** plots x vs. y numeric values contained in arrays.

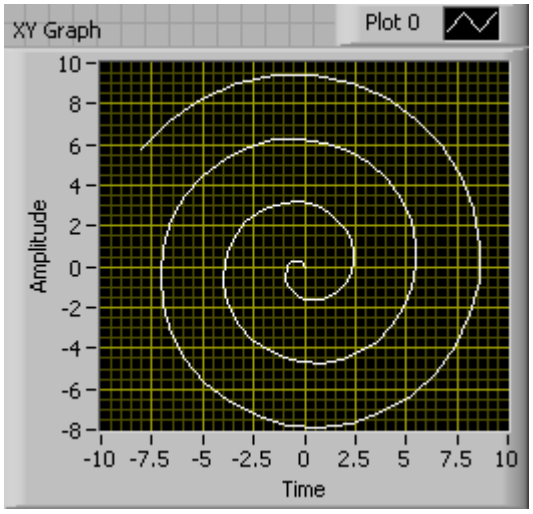


Figure 210: XY Graph

The example shown in Figure 211 generates the spiral shown in Figure 210.

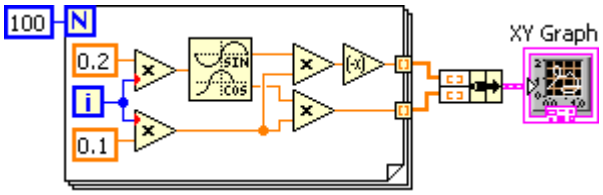


Figure 211: Spiral G Program

8 Interactive Programming

The heart of interactive programming in G is the **while loop**. Any input control within the **while loop** can be modified from the GUI window at run time to provide seamless interaction with the G program.

Figure 212: Creating Interactive Programs

In the GUI window, from the **Functions >> Modern >> Numeric** select the vertical pointer slide. From the **Functions >> Modern >> Graph** select **Waveform Chart**.

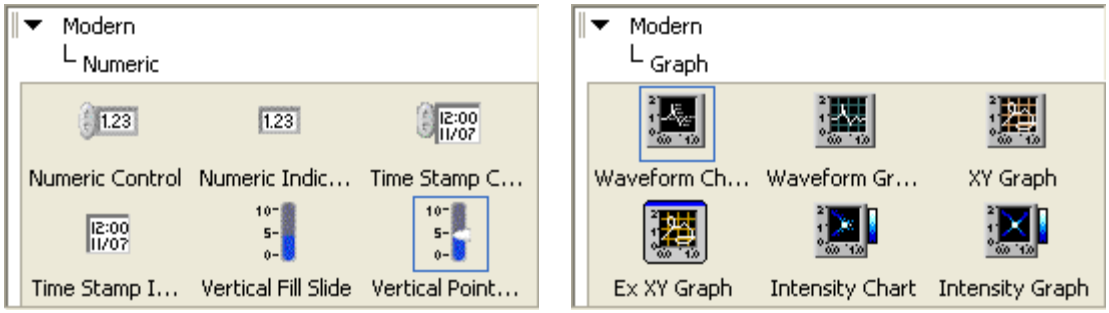


Figure 213: Vertical Pointer Slide and Waveform Chart

Re-label the vertical pointer slide as **Amplitude** and the waveform chart as **Sine Wave**. Re-arrange to GUI to look like the figure below.

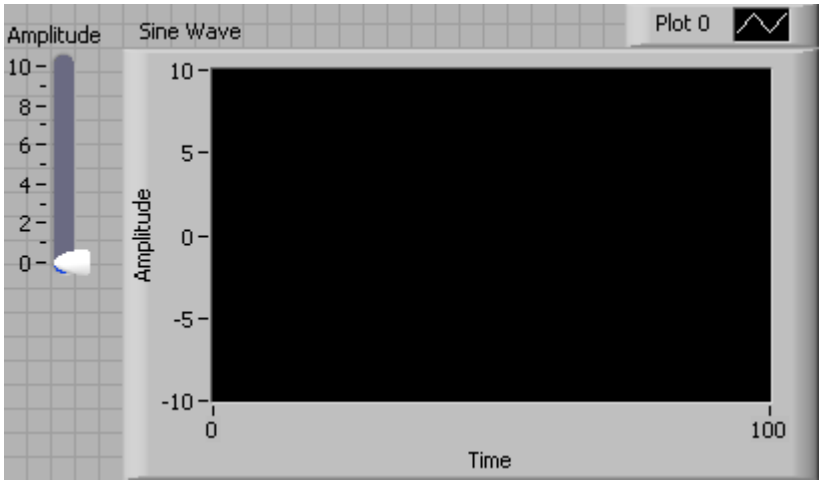


Figure 214: Slide & Waveform Chart in GUI Window

Right click on **Sine Wave** and select **Properties** from the pop-up menu.

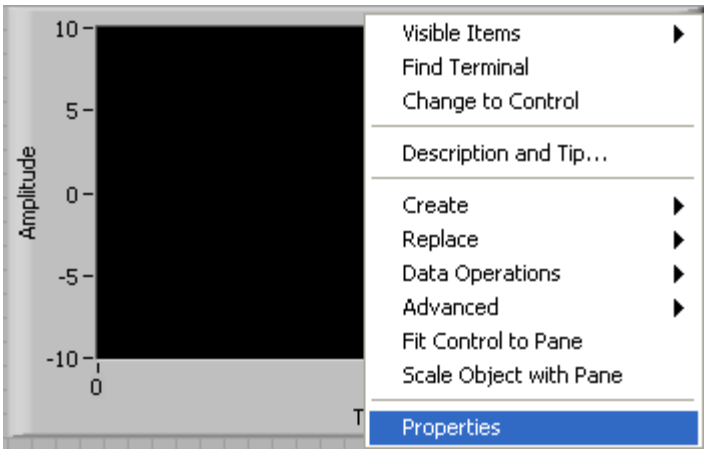


Figure 215: Selecting Chart Properties

Select the **Scales** tab and change **Maximum** to 1023. **Sine Wave** will display 1024 samples.

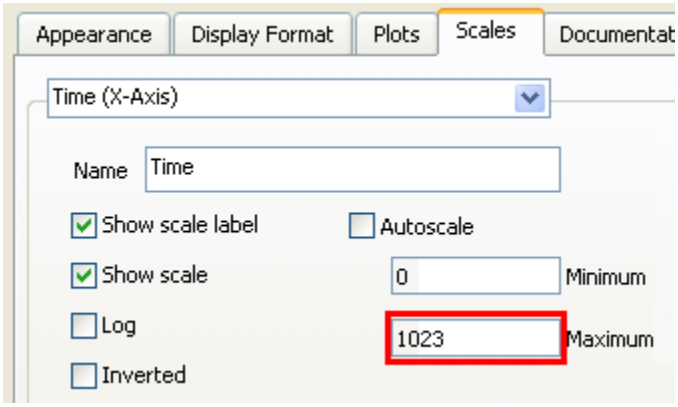


Figure 216: X-Axis Maximum

Click on the down arrow located to the right of **Time (X-Axis)** and select **Amplitude (Y-Axis)**.

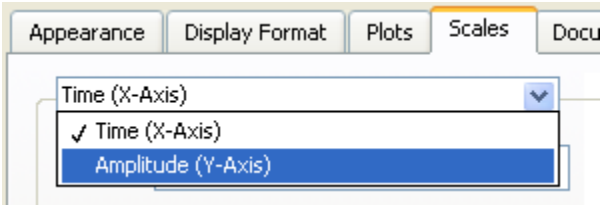


Figure 217: Selecting Y-Axis

De-select **Autoscale** and change the **Minimum** and **Maximum** values to **-10** and **10**. Click **OK**.

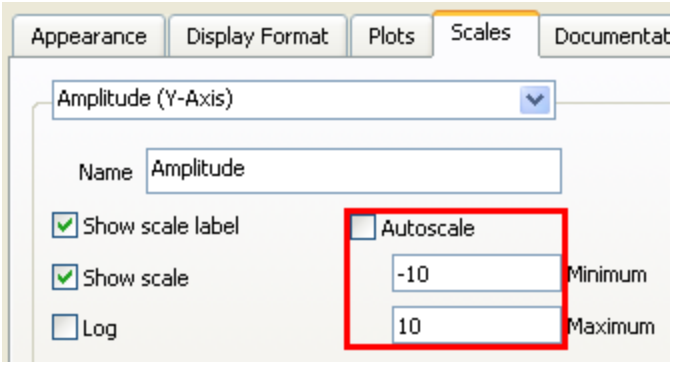


Figure 218: De-Selecting Autoscale

In the G programming window, re-arrange the **Amplitude** and **Sine Wave** terminals and finish the program as shown in Figure 84.

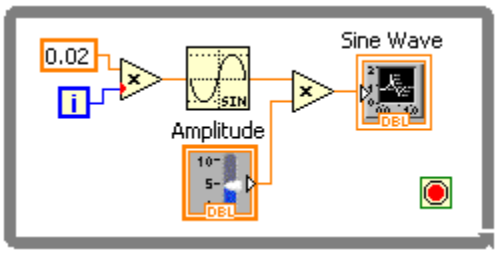


Figure 219: Interactive Sine Wave Diagram

Scroll the mouse pointer over the **Loop Control**…

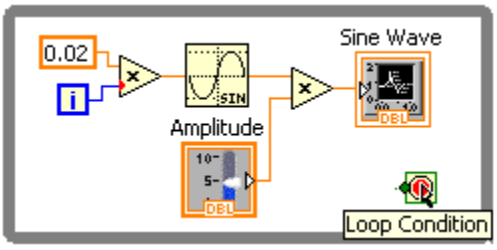


Figure 220: Loop Condition

And right click on the **Loop Control** and from the pop-up menu select **Create Control**.

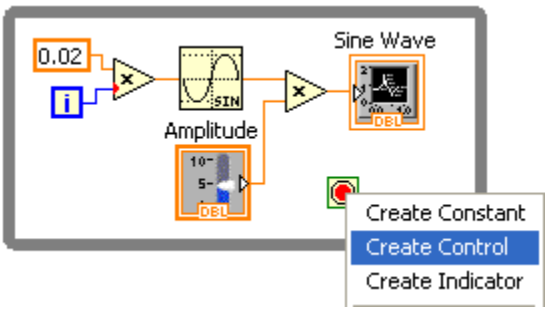


Figure 221: Create Loop Control

A **stop** terminal is created…

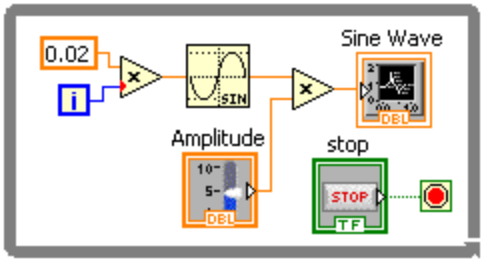


Figure 222: Interactive G Program

With the corresponding **stop** Boolean input control. Save the G program as **Interactivity.vi**.

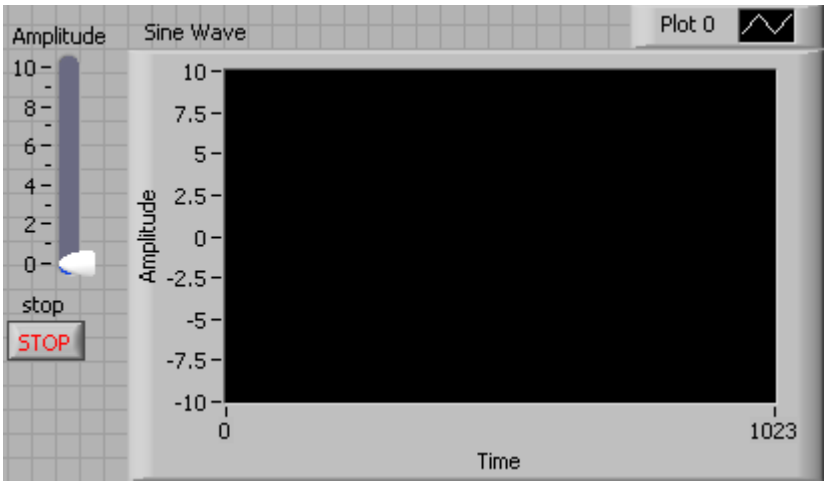


Figure 223: Interactive Program

Run the G program.

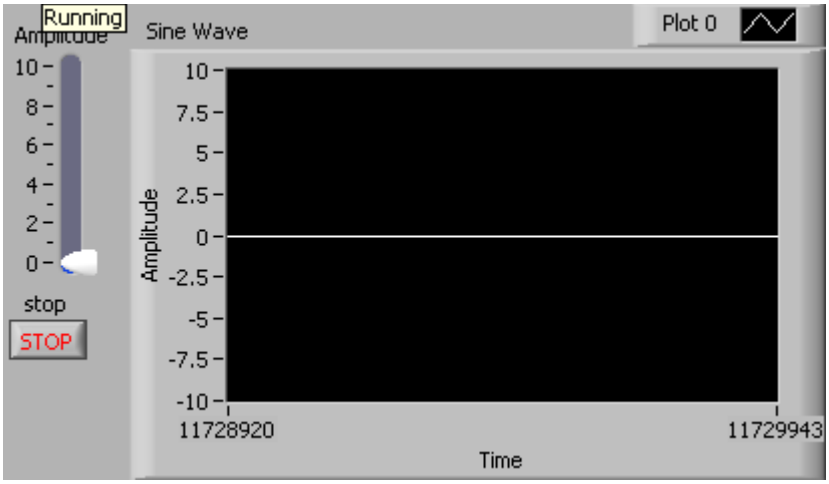


Figure 224: Interactive Program

While the program is running, change the **Amplitude** and watch the graph update to reflect the interactive changes.

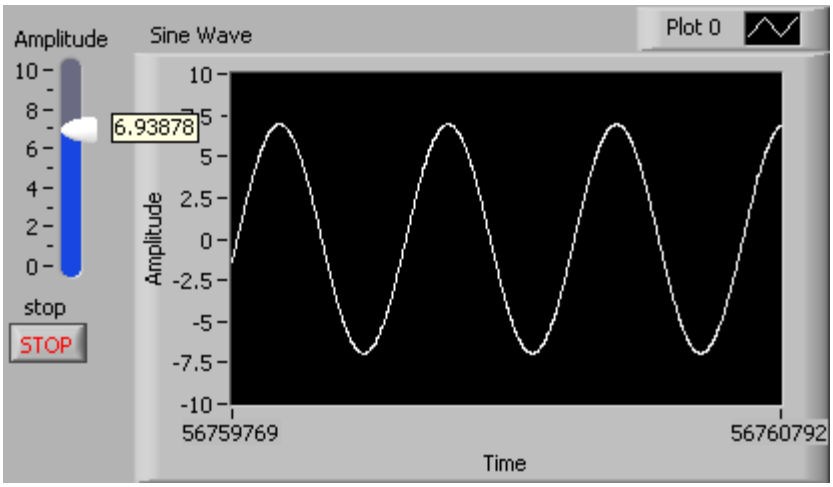


Figure 225: Interactive Program

To end the G program, simply click on the **stop** button.

Congratulations. You have successfully completed and executed your first interactive G program.

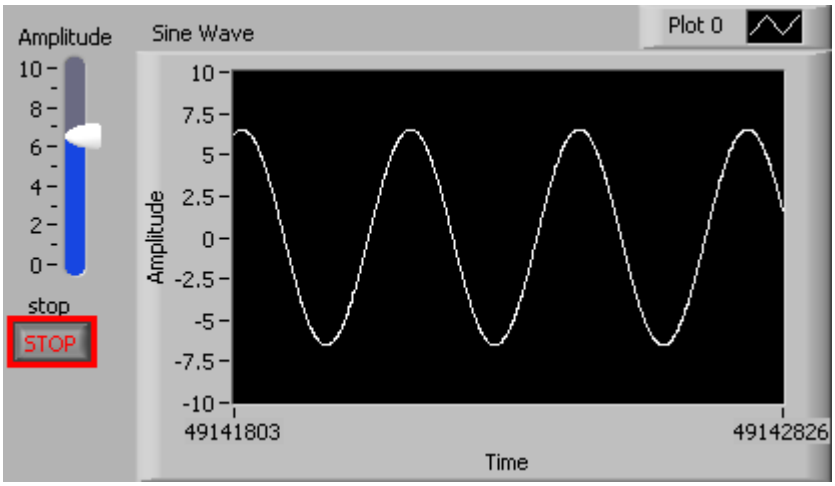


Figure 226: Interactive Program

9 Parallel Programming

In 1985, by design, G was developed to address and simplify parallel programming. If you have gone through the examples in this book, you have already developed various parallel programs.

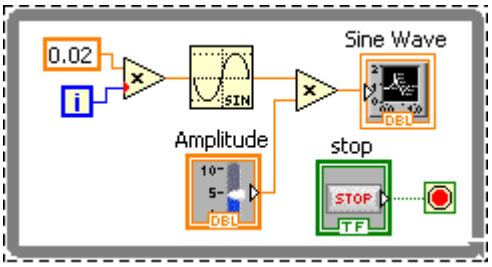


Figure 227: Select Diagram for Parallel Programming

In the following example, we will develop a simple program where interactivity and parallelism are part of the program.

From the menu select **Edit >> Copy**.

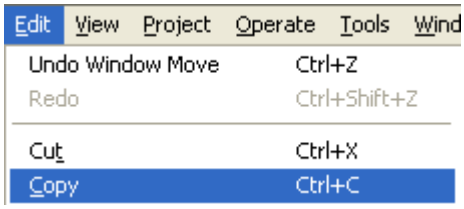


Figure 228: Copy Selected Diagram

Create a copy of the while loop and its contents by selecting **Edit >> Paste**. Organize the diagram as shown in the figure below.

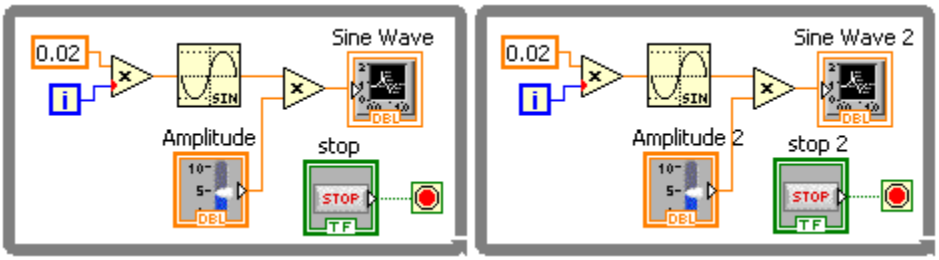


Figure 229: Paste Diagram

Go the GUI window and organize the input and output controls as shown in the figure below.

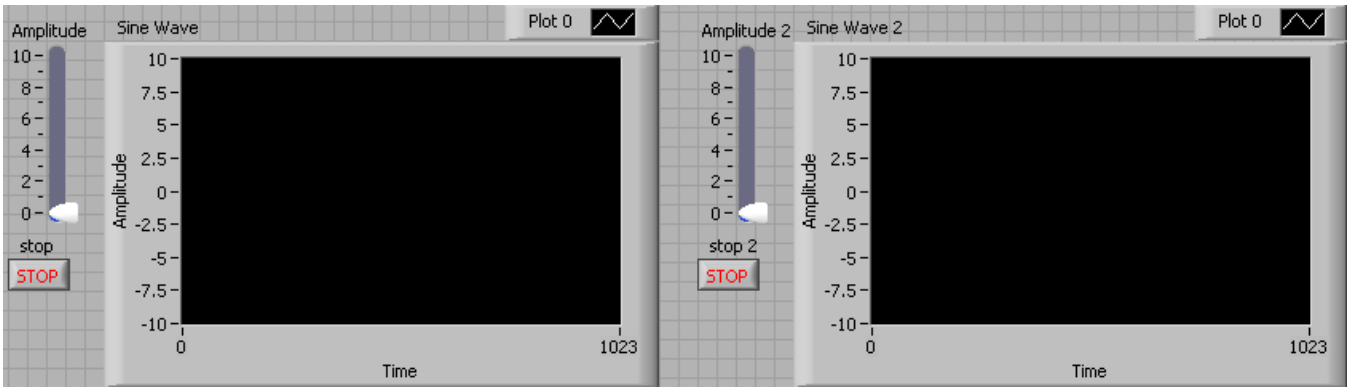


Figure 230: Parallel G Program

You have just completed your first parallel interactive program using G. Save the program, run it and interact with it. To end this program click on **stop** and **stop 2**.

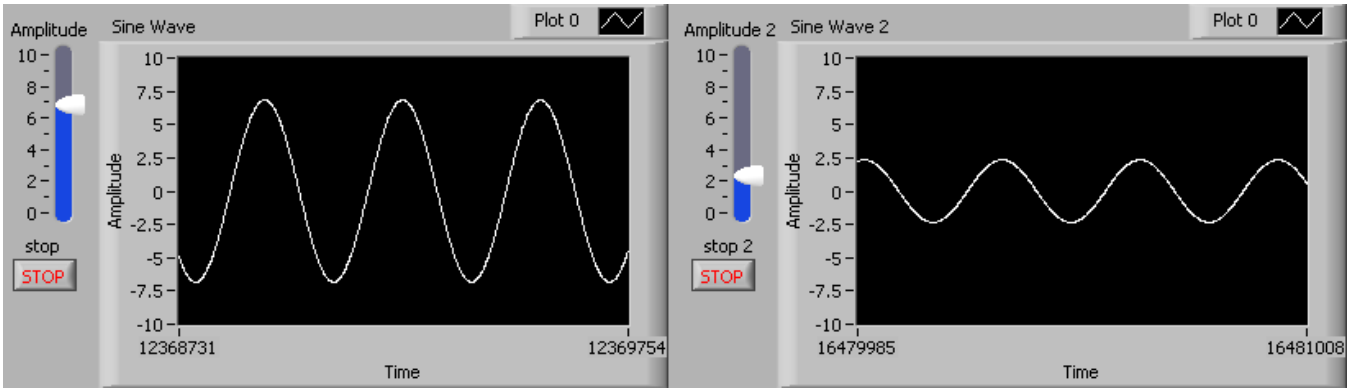


Figure 231: Parallel Interactive G Program

10 Multicore Programming

If you have written parallel programs in G and have a multicore computer, CONGRATULATIONS!!! You have been successfully developing interactive parallel programs that execute in multicore PC processors.

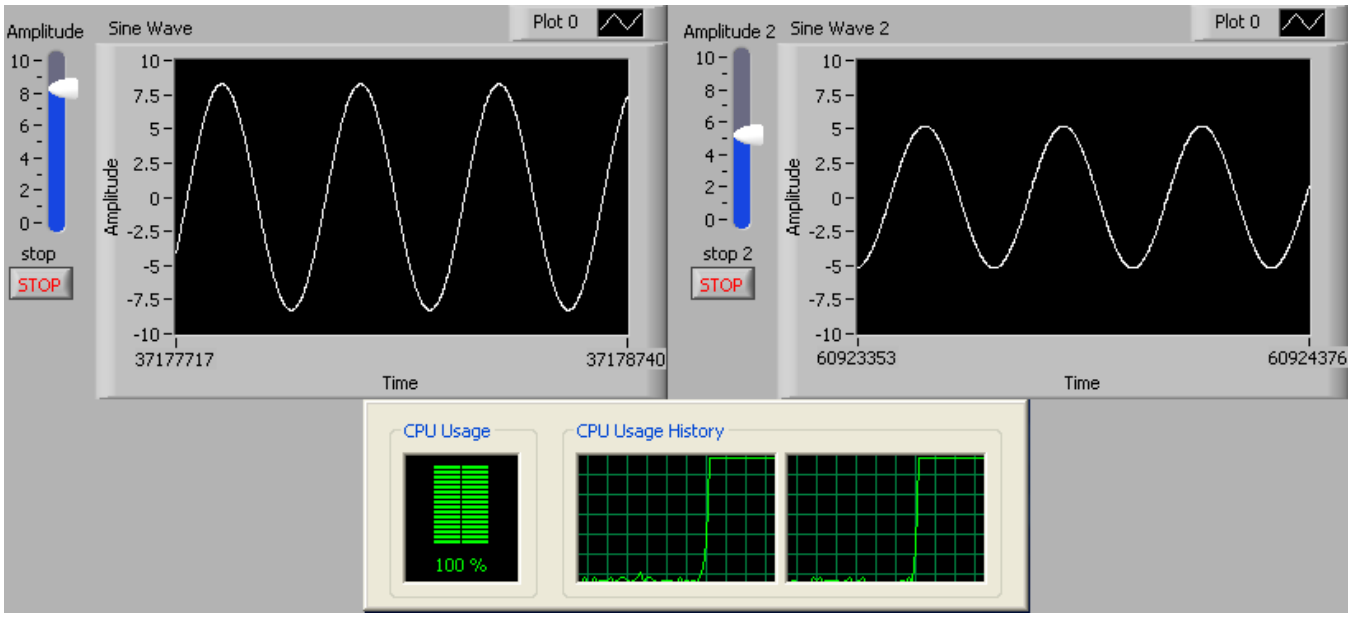


Figure 232: Interactive Multicore G Program

The following sections discuss some multicore programming techniques to improve the performance of G programs.

10.1 Data Parallelism

Matrix multiplication is a compute intensive operation that can leverage data parallelism. Figure 233 shows a G program with 8 sequential frames to demonstrate the performance improvement via data parallelism.

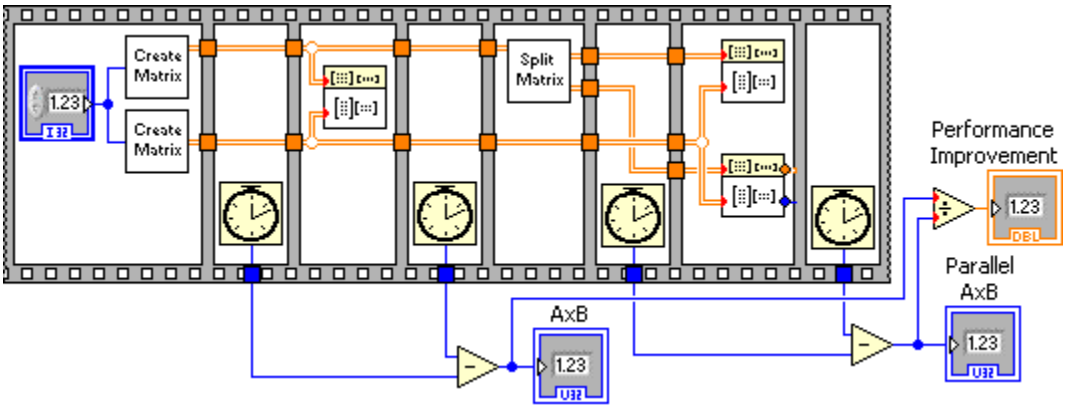


Figure 233: Data Parallelism

The **Create Matrix** function generates a square matrix based of size indicated by **Size** containing random numbers between 0 and 1. The **Create Matrix** function is shown in Figure 234.

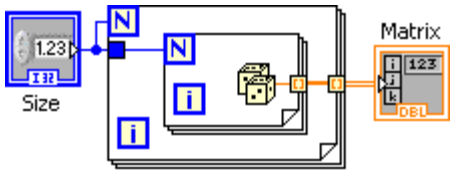


Figure 234: Creating a Square Matrix

The **Split Matrix** function determines the number of rows in the matrix and shifts right the resulting number of rows by one (integer divide by 2). This value is used to split the input matrix into the top half and bottom half matrices. The **Split Matrix** function is shown in Figure 235.

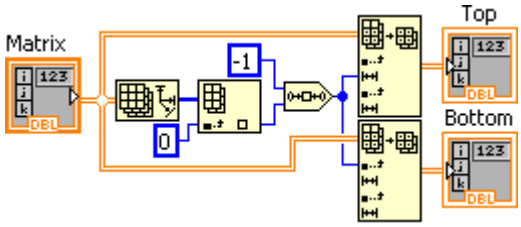


Figure 235: Split Matrix into Top & Bottom

Sequence Frame	Operation Description
First Frame	Generates two square matrices initialized with random numbers
Second Frame	Records start time for single core matrix multiply
Third Frame	Performs single core matrix multiply
Fourth Frame	Records stop time of single core matrix multiply
Fifth Frame	Splits the matrix into top and bottom matrices
Sixth Frame	Records start time for multicore matrix multiply
Seventh Frame	Performs multicore matrix multiply
Eighth Frame	Records stop time of multicore matrix multiply

The rest of the calculations determine the execution time in milliseconds of the single core and multi-core matrix multiply operations and the performance improvement of using data parallelism in a multi-core computer.

The program was executed in a dual core 1.83 GHz laptop. The results are shown in Figure 236. By leveraging data parallelism, the same operation has nearly a **2x** performance improvement. Similar performance benefits can be obtained with higher multicore processors

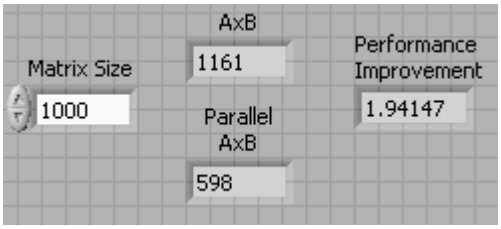


Figure 236: Data Parallelism Performance Improvement

10.2 Task Pipelining

A variety of applications require tasks to be programmed sequentially and continually iterate on these tasks. Most notably are telecommunications applications require simultaneous transmit and receive. In the following example, a simple telecommunications example illustrates how these sequential tasks can be pipelined to leverage multicore environments.

Consider the following simple modulation - dulation example where a noisy signal is modulated transmitted and demodulated. A typical diagram is shown in Figure 237.

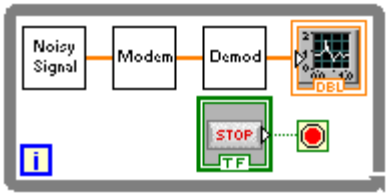


Figure 237: Sequential Tasks

Adding a shift register to the loop allows tasks to be pipelined and be executed in parallel in rate cores should they be available. Task pipelining is shown in Figure 238.

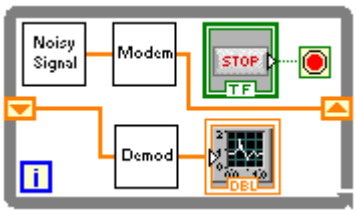


Figure 238: Pipelined Tasks

The program below times the sequential task and the pipelined tasks to establish its performance improvement when executed in multicore computers.

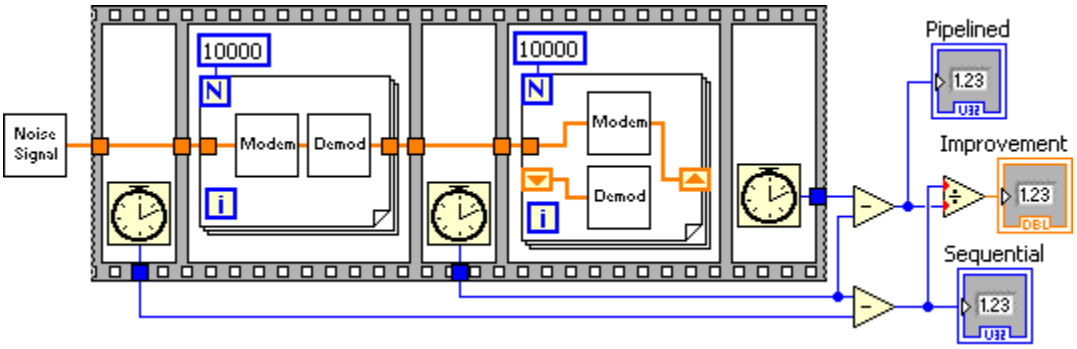


Figure 239: Task Pipelining Program Example

Figure 240 shows the results of running the above G program in a dual core 1.8 GHz laptop. Pipelining shows nearly **2x** performance improvement.

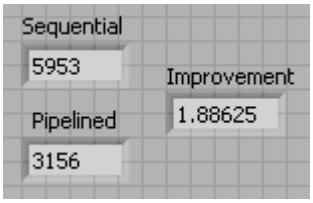


Figure 240: Pipelining Performance Improvement

10.3 Pipelining Using Feedback Nodes

Feedback Nodes provide a storage mechanism between loop iterations. They are programmatically identical to the **Shift Registers**. **Feedback Nodes** consist of an **Initializer Terminal** and the **Feedback Node** itself (see Figure 241).

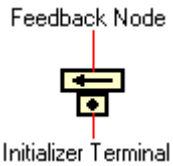


Figure 241: Feedback Node

To add a **Feedback Node**, right click on the G programming window and select **Feedback Node** from the **Functions >> Programming >> Structures** pop-up menu. The direction of the **Feedback Node** can be changed by right clicking on the node and selecting **Change Direction**.

Figure 242: Feedback Node Direction

The diagram shown in Figure 243 is programmatically identical to the diagram in Figure 238.

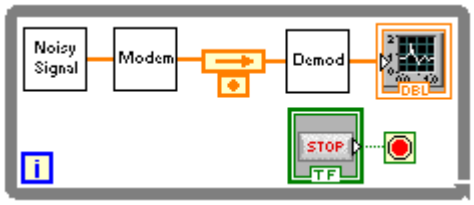


Figure 243: Pipelining with Feedback Node

Similarly, the diagram in Figure 244 is programmatically identical to that in Figure 239.

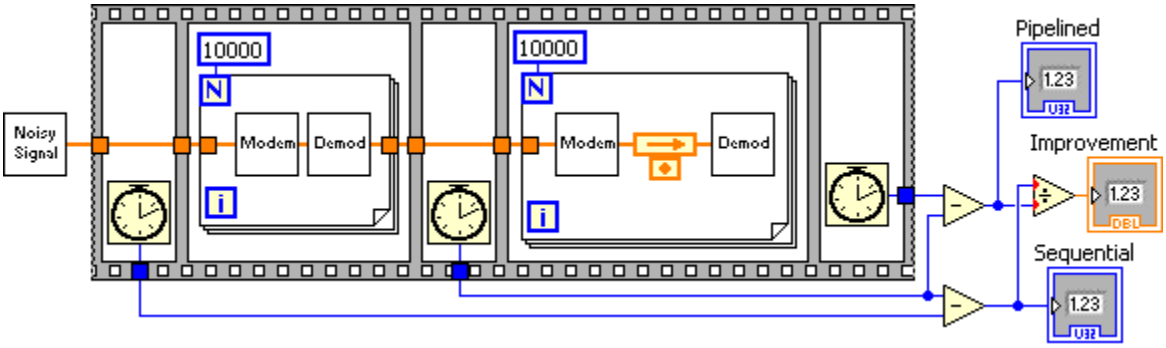


Figure 244: Pipelining Tasks with Feedback Nodes

11 Input and Output

11.1 Writing to File

Consider the function in Figure 245 where a set of numbers in a one-dimensional array represents the resulting noisy signal is to be written to a file. This section will outline the steps required to create files.

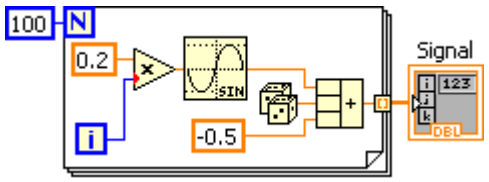


Figure 245: Noisy Signal Function

Create a new G program, right click in the G programming window and select **File Dialog** from the **Functions >> Programming >> File I/O >> Advanced Functions** menu. Drag and drop the **File Dialog** function onto the G programming window.

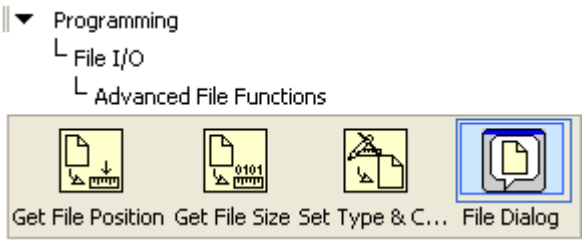


Figure 246: File Dialog

The **Configure File Dialog** dialog box automatically appears to configure the function. Accept the default configuration shown in Figure 247 to create a single file by clicking the **OK** button.

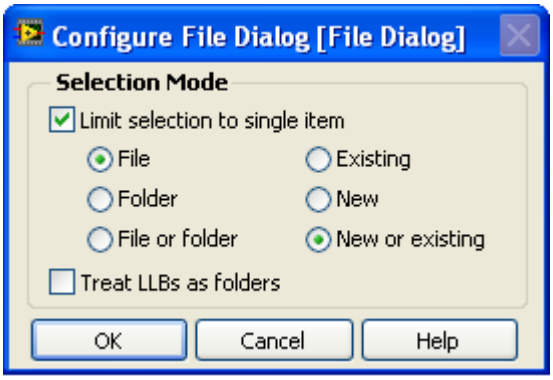


Figure 247: Configure File Dialog

The resulting diagram after closing the configuration dialog box is shown in Figure 248. Optionally, right click on **File Dialog** and select **View As Icon** from the pop-up menu. This will save some real estate in the G programming window.

Figure 248: G File Dialog

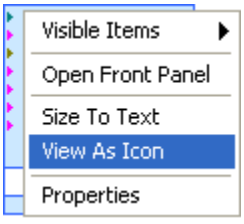


Figure 249: View As Icon

From the **Functions >> Programming >> File I/O** menu select **Open/Create File**, **Write Binary File** and **Close File** functions.

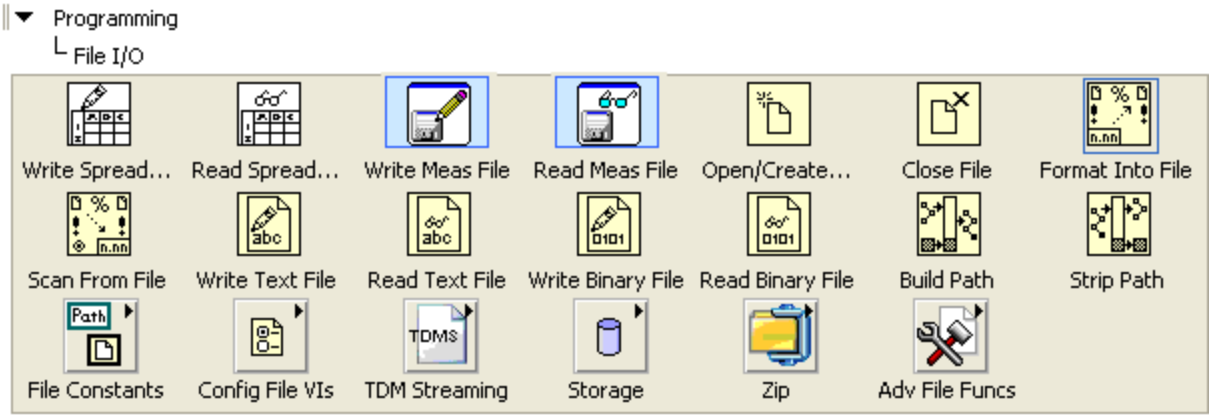


Figure 250: File Input and Output Operators

Arrange the File I/O operations as shown in Figure 251.

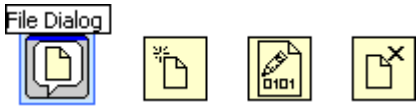


Figure 251: Open, Write and Close File Diagram

Right click on the **operation (0:open)** terminal of the **Open/Create File** function (highlighted in Figure 252).

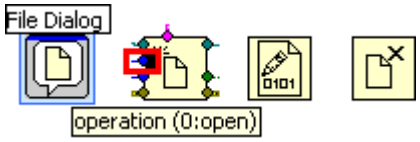


Figure 252: File Create Operation

Select **Create >> Constant** from the pop-up menu.

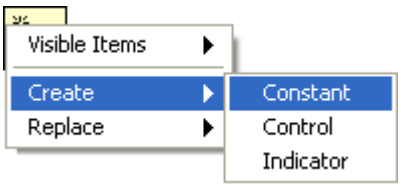


Figure 253: Create Operation Constant

Arrange the diagram to look as in Figure 254.

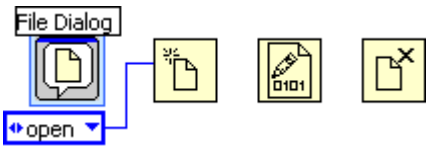


Figure 254: Operation Constant

Click on the down arrow in the **operation** constant just created and select **open or create** from the pop-up menu.

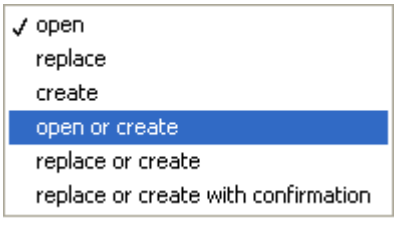


Figure 255: Open or Create File Operation

The resulting updated **operation** constant value is shown in Figure 256.

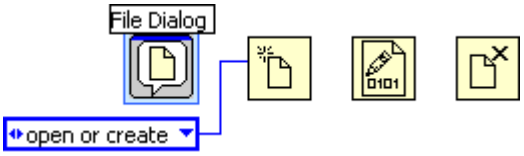


Figure 256: Create File to Write

Repeat the process to create a constant for the **access (0:read/write)** terminal (highlighted in Figure 257).

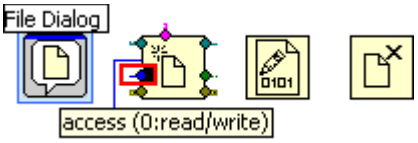


Figure 257: File Access Mode

Set the constant to **write-only**. Re-arrange the block diagram to look like the diagram shown in Figure 258. At this point, the file is set to create a new file for writing.

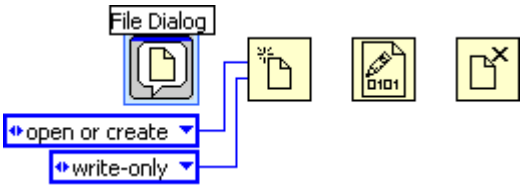


Figure 258: Write-Only Mode

Get the **Noisy Signal** function and wire its output data to the **Data** terminal of the **Write to Binary File** function.

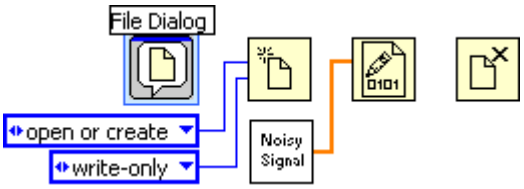


Figure 259: Writing Binary Data

Complete the diagram by connecting the **Open**, **Write** and **Close** file operations as shown in Figure 260.

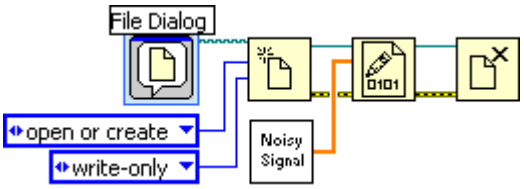


Figure 260: Writing to File G Program

When this G program is executed, the standard file dialog box appears. Name the file to be written **signal.dat**.

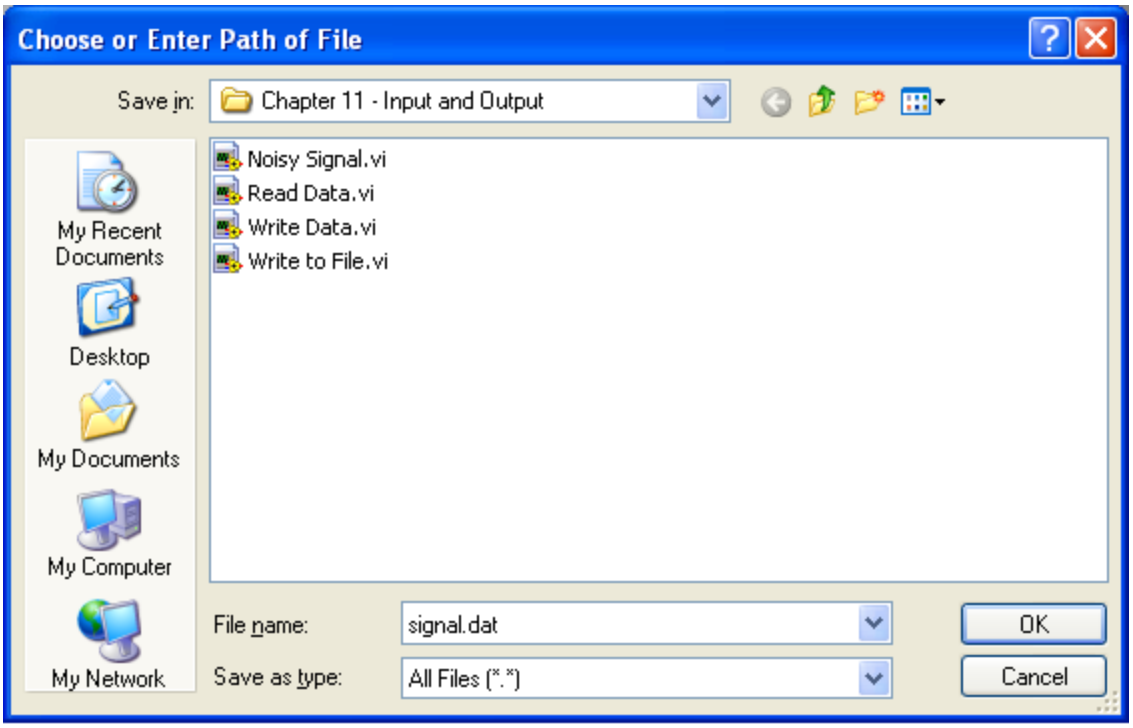


Figure 261: Create File Dialog Box

Once the program completes executing, the **signal.dat** file is created and located in the location indicated by the path selected.

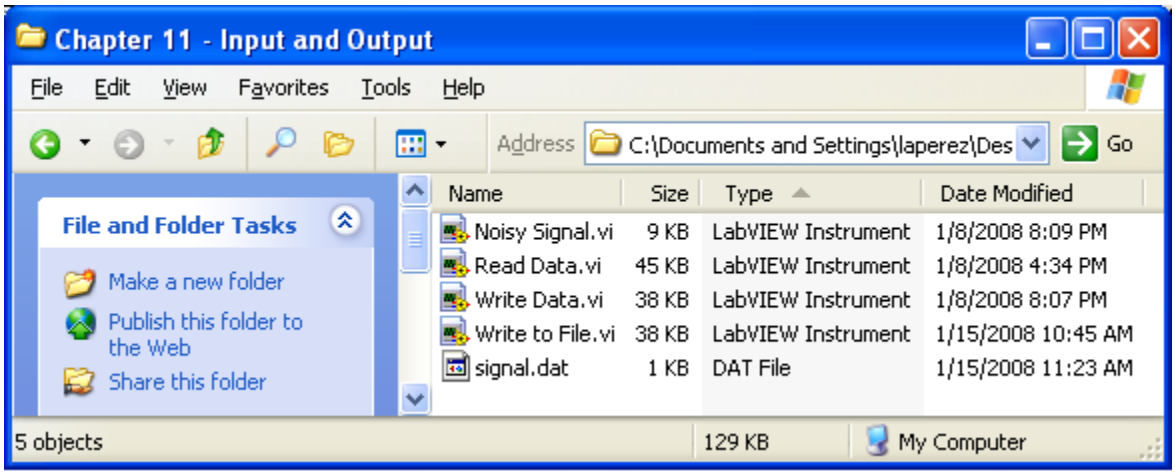


Figure 262: Signal.dat File

11.2 Reading From Files

The **signal.dat** file created in the previous example will be used to read data from a file. As in the previous example, select the **File Dialog**, **Open/Create File**, **Read from Binary File** and **Close File** functions.

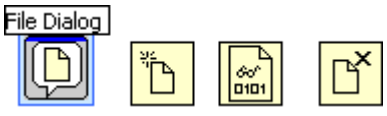


Figure 263: Operators to Read Files

Create constants by right clicking on the **operation (0:open)** and **access (0:read/write)** terminals of the **Open/Create File** operation. Set the constants to **open** and **read-only** respectively (see Figure 264).

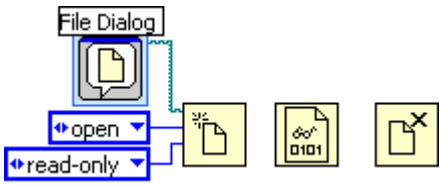


Figure 264: Set to Open and Read

Similar to creating arrays, drop an array constant in the G diagram, drop a numeric constant onto the array constant and set the data type representation to **double**. Wire this array constant to the **data type** terminal of the **Read from Binary File** function as shown in Figure 265.

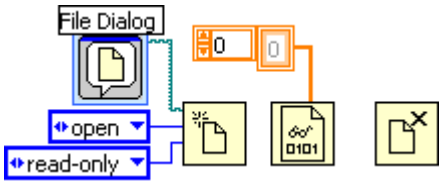


Figure 265: Data Type to Read

In the GUI window, drop a **Waveform Graph**.

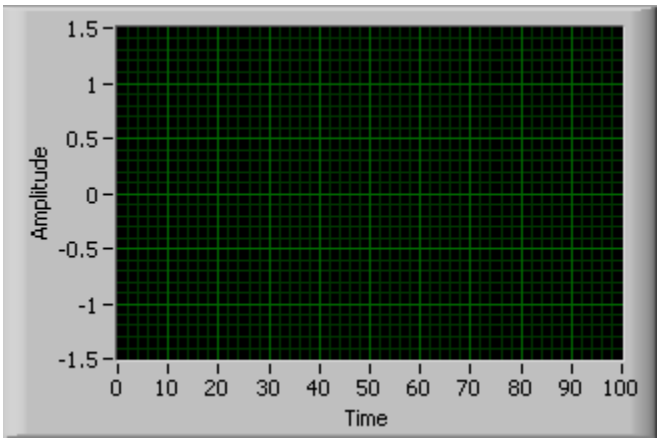


Figure 266: Graph for Data To Be Read

With the data type specified, wire the **data** terminal of the **Read from Binary File** function to the **Waveform Graph** terminal as shown in Figure 267.

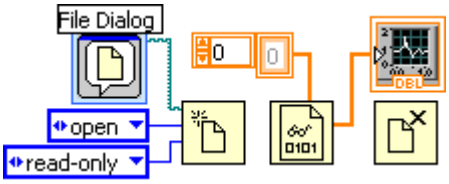


Figure 267: Data to Be Read

Complete the program by wiring **refnum** and **error** terminals of the **Open/Create File**, **Read from Binary File** and **Close File** functions as shown in Figure 268.

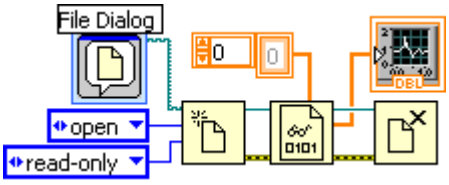


Figure 268: Read Binary Data G Program

When this program is executed, a file dialog box appears. Select the **signal.dat** file and click **OK**.

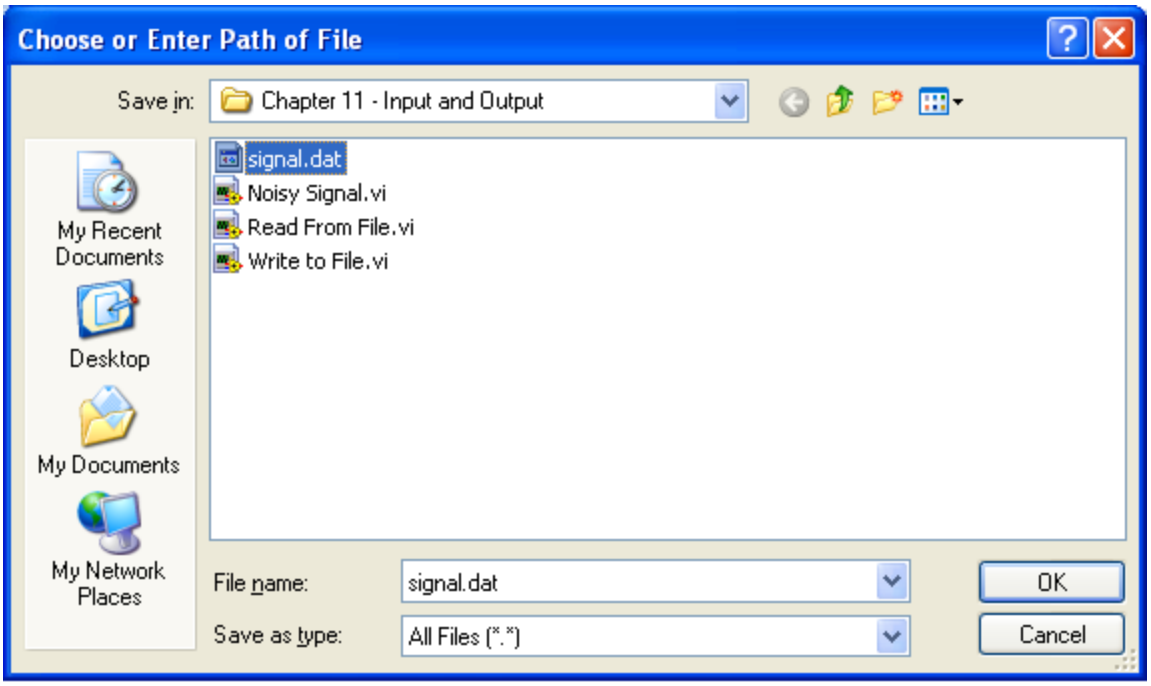


Figure 269: Select Binary File to Read From

The binary data in **signal.dat** is read and plotted in a **Waveform Graph**. The result is shown in Figure 270.

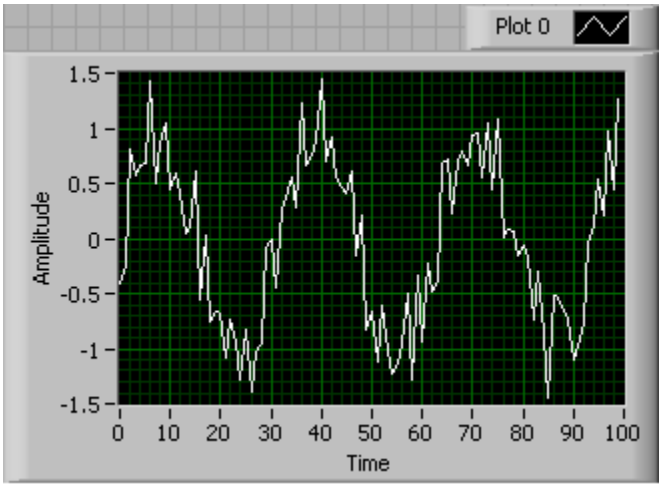


Figure 270: Read Data Graphed

Table of Figures