香港開放教科書
Open Textbooks for Hong Kong

Free to use.
自由編輯運用
Free to change.
共享優質課本
Free to share.

Introduction to G Programming

香港公開大學
THE OPEN UNIVERSITY OF HONG KONG

Contents

Preface to an "Introduction to G Programming"

The Internet, personal devices and multicore computers have greatly changed and enhanced our lives by allowing us to access information and entertainment on-demand anytime, anywhere. While these technologies are great on their own merit, the reality is that in order to reap the benefits, someone has to program these devices to develop useful applications.

Historically, text-based high-level programming languages provided the first productive alternative to develop targeted applications. As more networked computing platforms enter the mainstream, the programming complexities of text based languages becomes a limiting factor, especially for domain experts who are typically not programming or computer science experts. The G programming language provides the next generation programming alternative allowing users to develop interactive parallel programs whether they have extensive programming experience or not. It's graphical syntax and constructs allow researchers, teachers, students and even children to program complex devices and systems in minutes rather than hours, days or even months.

G is a data flow graphical programming language. Originally designed to address test and measurement needs, its general purpose programming attributes has been applied in telecommunications, biomedical, aerospace, environmental and many other industries. In general, G is used in Science, Technology, Engineering and Math (STEM) projects and programs but is not limited to STEM.

The book was written to help the user learn the G programming syntax and begin developing G programs quickly and easily. Although familiarity with programming concepts could help learning G, the book assumes the user has had no previous exposure to any programming languages. Therefore, to avoid confusion, no pseudo-code or syntax comparisons are made with text-based programming languages. All examples in this book are working graphical examples and have been tested thoroughly. Chapter 1 is an introductory tutorial providing a reference for beginners and seasoned programmers alike. Subsequent chapters provide more details on the G syntax building up to the development of parallel programs that run on multicore platforms.

This book is not an introduction to programming, style guide, debugging or to development environments. It is strictly a concise G syntax. Additionally, the user must have access to National Instruments LabVIEW and be familiar with LabVIEW basics. Nonetheless, the user should be able to read along to learn and understand the benefits of G programming.

As one of the original LabVIEW development team members, developing G programs has been a pleasant and productive experience. It is the author's sincere hope that the user finds G programming and interesting endeavor as well.

Lalo Perez, Ph.D.

About the Author

Dr. Eduardo "Lalo" Perez is one of the original LabVIEW development team members responsible for the design, deployment and optimization of the Digital Signal Processing and Data Analysis Libraries still being used today.

Dr. Perez has nearly 30 years of engineering programming experience and nearly 20 years designing and implementing digital signal processing software architectures for the deployment of multimedia, communications and biomedical real-time applications. He has extensive experience in large enterprises -where he successfully deployed global multimedia networks and services for AT&T and Ernst & Young -as well as startups where he forged strategic alliances with Intel, Microsoft, Samsung and Texas Instruments to accelerate adoption of audio-visual services over IP. Dr. Perez' other accomplishments include: member of the first ever live video webcast team over ISDN in 1994, first commercial DSL live IP broadcast in 1999, provided nearly 3 years of webcasting services for Broadcast.com(now Yahoo! Broadcasting Services)with 100% success rate, team member at SBC Communications that launched SBC Internet Services and DSL services and provided guidance for deployment of early Internet multimedia communities.

Dr. Perez holds a variety of patents on high quality video encoding, compressive data acquisition, multirate media processing, remote computing and streaming delivery mechanisms.

A native from Mexico, he immigrated to the U.S. at the age of 18 to pursue higher education. He received his B.S., M.S., and Ph.D. in Electrical and Computer Engineering in the areas of Telecommunications, Image Processing, and Real-Time Digital Imaging Systems, respectively, all from the University of Texas at Austin.

Dr. Perez was a member of the 1976 Mexican Olympic Swimming team.

Chapter 1 Introduction to G Programming

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 Block Diagram programming window.

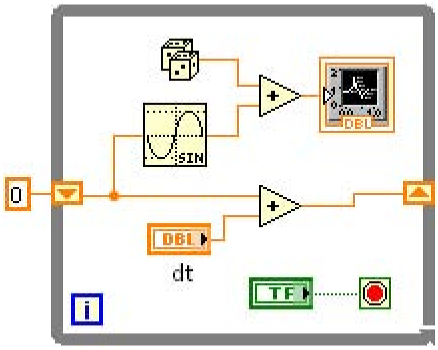


Figure 1.1 G Block Diagram

The program input data and results are manipulated and displayed in the Front Panel window.

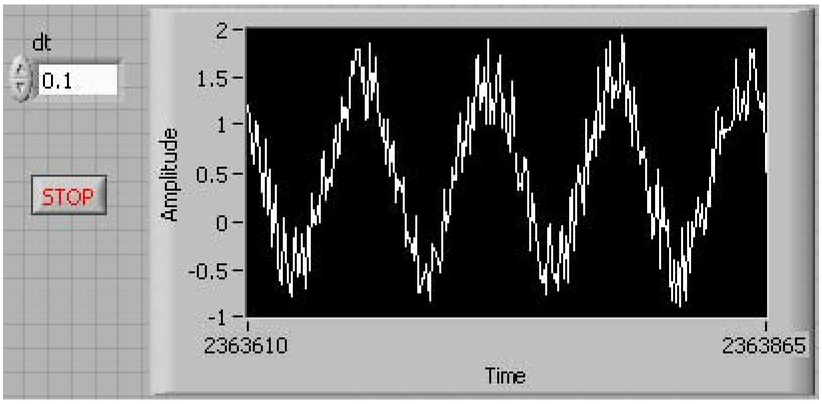


Figure 1.2 G User Interface

1.1 Hello Graphical World

The first program is to display the text "Hello graphical interactive parallel multicore world" in the Front Panel window.

Right click on the Block Diagram window and select **String Constant** from the **Functions** » **Programming** » **String** menu.

Drag and drop the **String Constant** onto the Block Diagram window as show in the following Figure 1.3.

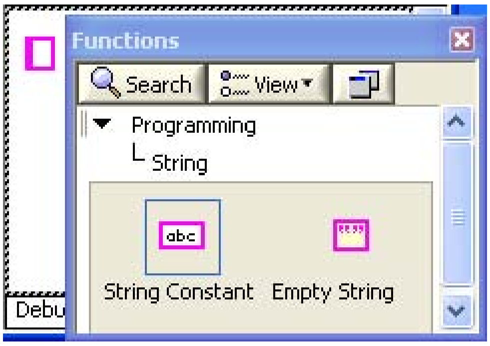


Figure 1.3 String Constant

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

Figure 1.4 "Hello...world" String Constant

Right click in the Front Panel window and select a **String Indicator** from the **Controls** »**Modern** » **String & Path** menu.

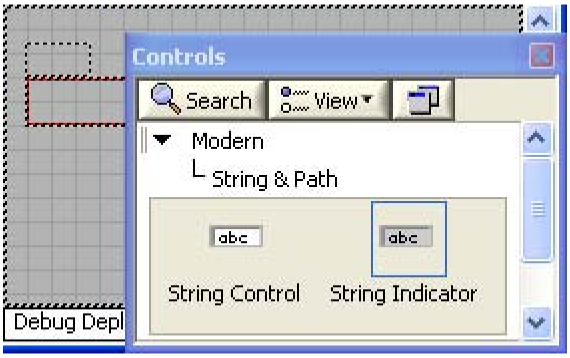


Figure 1.5 Select String Indicator

Drop it into the Front Panel window.

Figure 1.6 String Indicator

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

Figure 1.7 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 1.8 Wired G Block Diagram

Save your program as **Hello, World.vi**. Return to the Front Panel window. Click the run button ([U+27AF]). You have successfully completed and executed your first G program.

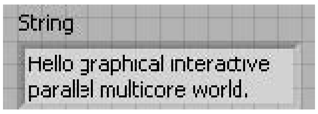


Figure 1.9 Hello, World G Program Executed

1.2 Arithmetic Expressions

The next program converts degrees from Fahrenheit to Celsius using the formula

$$C = \frac{5}{9}(F - 32)$$

In the Block Diagram window, select the subtract, multiply and divide from the **Functions** » **Mathematics** » **Numeric** menu

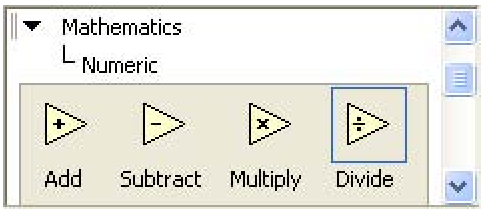


Figure 1.10 Numeric Operations

Wire the **subtract**, **multiply** and **divide** functions as shown in Figure 3.11.

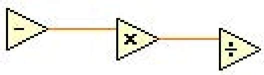


Figure 1.11 Subtract, Multiply and Divide

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

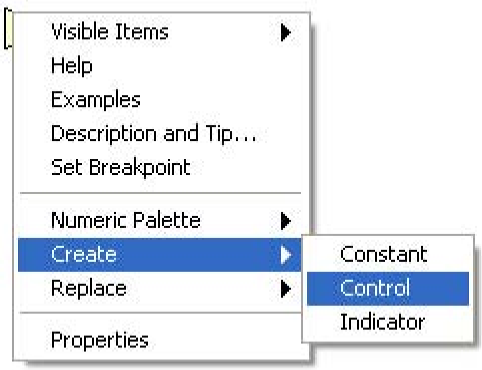


Figure 1.12 Create Control

Re-label**x** as **Fahrenheit** and wire the terminal as shown in Fahrenheit Input Control.

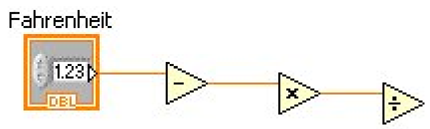


Figure 1.13 Fahrenheit Input Control

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

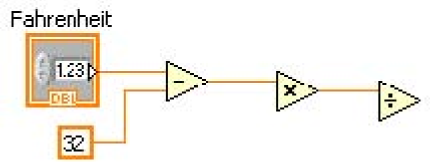


Figure 1.14 Fahrenheit Numeric Constant

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

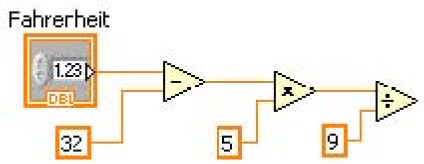


Figure 1.15 Fahrenheit Numeric Constants

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 fnal diagram is shown in Fahrenheit to Celsius G Diagram

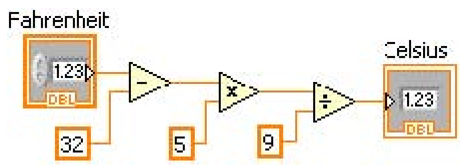


Figure 1.16 Fahrenheit to Celsius G Diagram

Switch to the Front Panel 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.

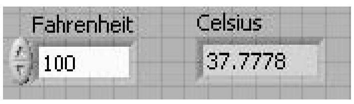


Figure 1.17 Fahrenheit to Celsius calculator

1.3 Functions

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

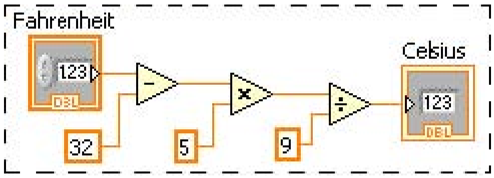


Figure 1.18 Select G Block Diagram

The selected diagram is highlighted as shown in Selected G Block Diagram

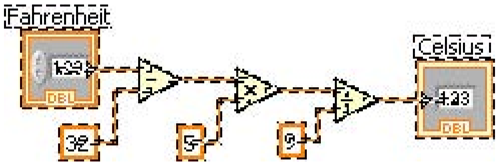


Figure 1.19 Selected G Block Diagram

From the **Edit** menu select **Create SubVI** to create a G function. The resulting diagram is shown in Creating a Function .

Figure 1.20 Creating a Function

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

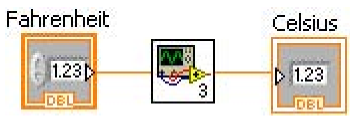


Figure 1.21 Diagram with Function

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**...

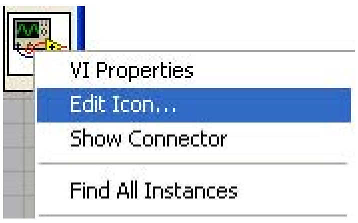


Figure 1.22 Edit Icon

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

Figure 1.23 Icon Editor

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

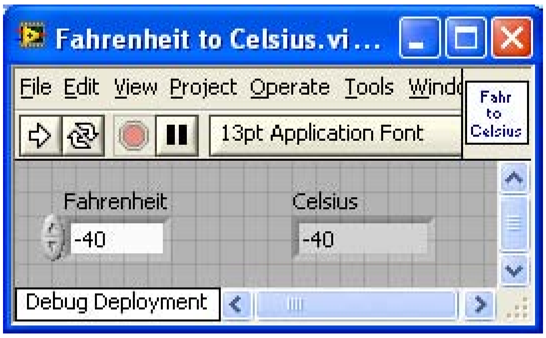


Figure 1.24 Edited Icon

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

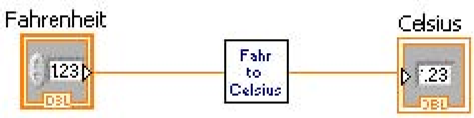


Figure 1.25 Function Calling

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(x/y}*=O is true therefore

Leap Year = {*Rem* (Year/4) = 0 And Not (*Rem* (Year/100) = 0)} Or *Rem* (Year/400) = 0 (3.1)

where And, Or and Not are Boolean operators.

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 Block Diagram window. Go to the **Functions** » **Programming** » **Numeric** menu in the Block Diagram window.

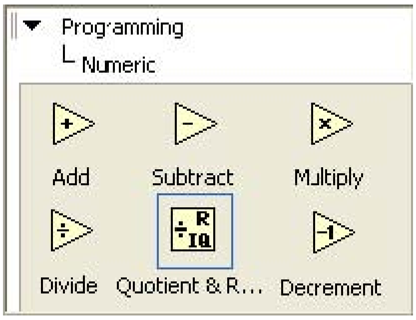


Figure 1.26 Quotient & Remainder Function

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.

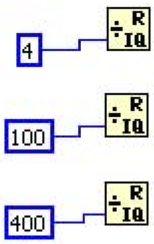


Figure 1.27 Leap Year Numeric Constants

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.

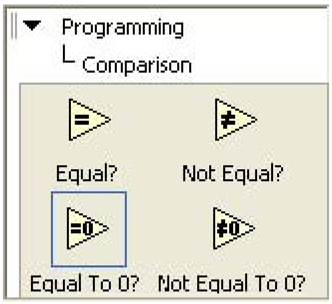


Figure 1.28 Comparison Functions

Organize the comparison operations as show in the diagram.

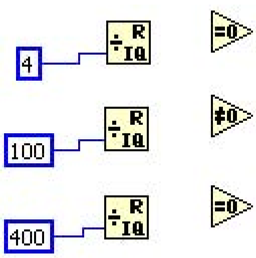


Figure 1.29 Diagram

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

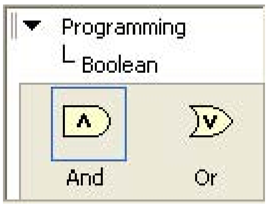


Figure 1.30 Boolean Operators

Place the Boolean operators as shown in Q&R, Comparison & Boolean Functions.

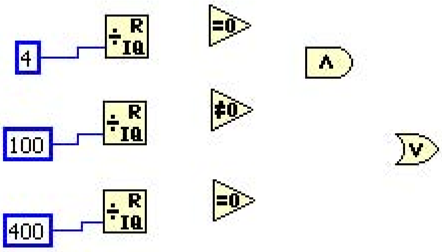


Figure 1.31 Q&R, Comparison & Boolean Functions

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

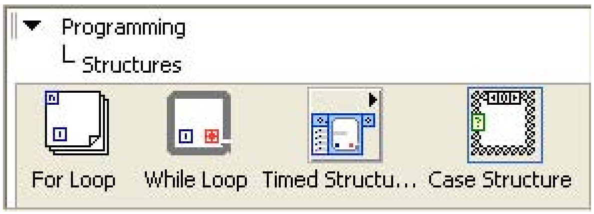


Figure 1.32 Case Structure

Click and drag on the Block Diagram window to create the **Case Structure**.

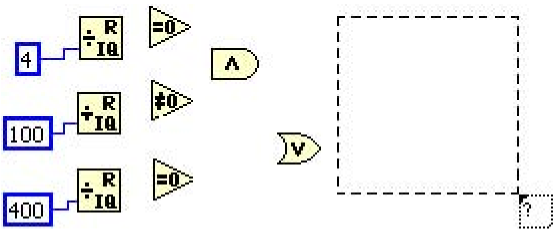


Figure 1.33 Creating a Case Structure

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

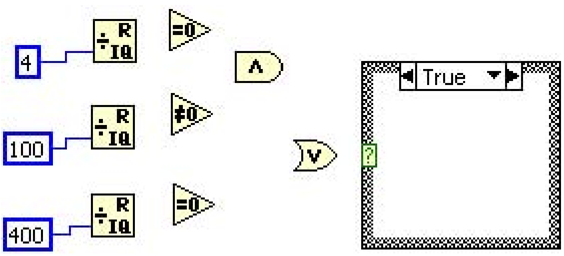


Figure 1.34 Created Case Structure

Drop a string constant and type "Is a Leap Year".

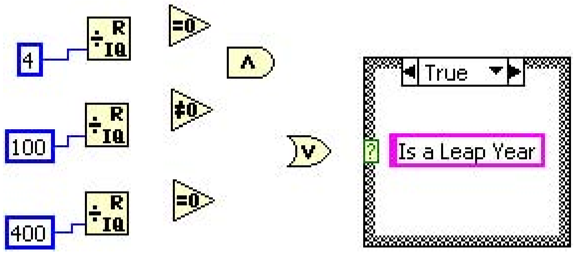


Figure 1.35 True Case Editing

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

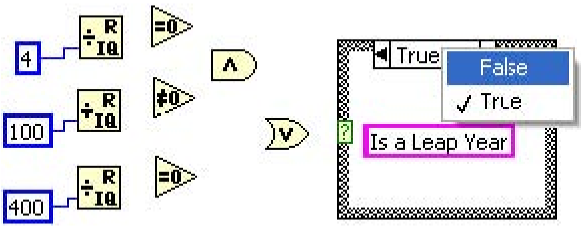


Figure 1.36 Selecting the False Case

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

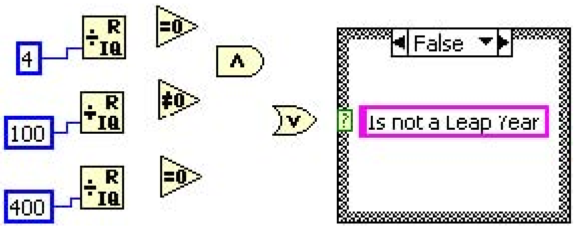


Figure 1.37 False Case Editing

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

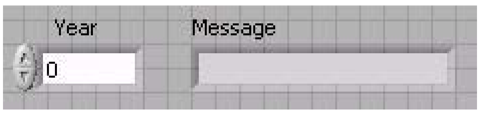


Figure 1.38 Leap Year GUI

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

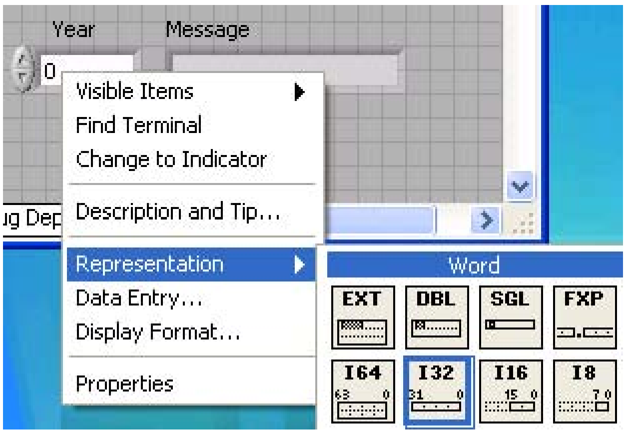


Figure 1.39 32-Bit Integer Numeric

Arrange the **Year** and **Message** terminals in the Block Diagram window as shown in the figure.

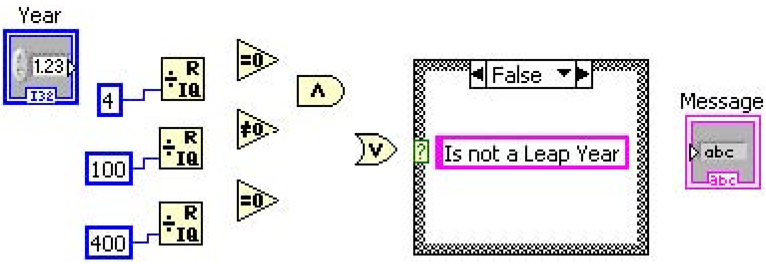


Figure 1.40 Unwired Leap Year Diagram

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

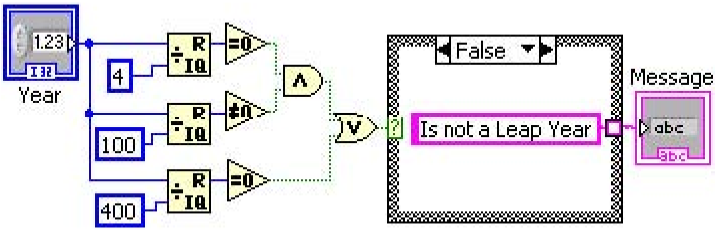


Figure 1.41 Leap Year False Case

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

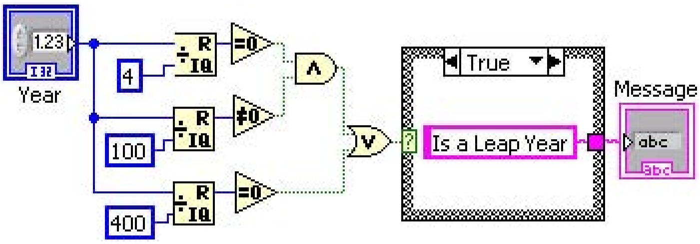


Figure 1.42 Leap Year True Case

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 1.43 Leap Year Program

1.5 Arrays

Right click on the Front Panel window and select**Array** from the **Controls** » **Modern** » A**rrays, Matrix & Cluster** menu, and drop an array onto the Front Panel 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 Front Panel window, the data type of the array is undefned as indicated by the grayed out portion of the array.

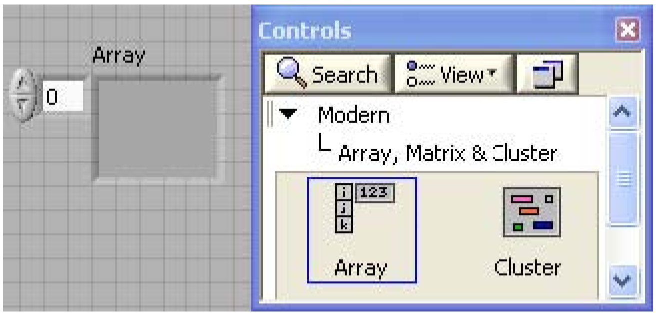


Figure 1.44 Arrays

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.

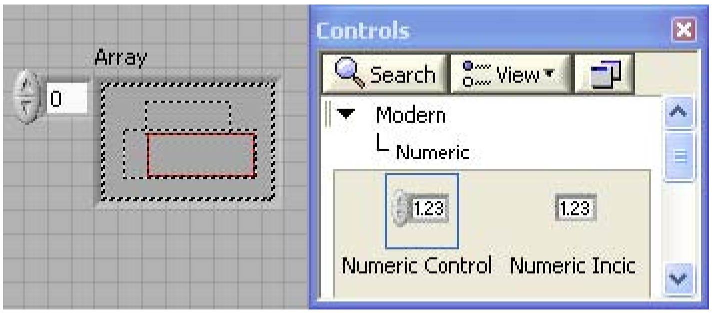


Figure 1.45 Creating a Numeric Array

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

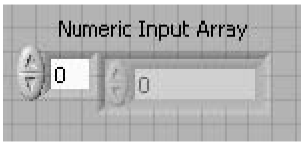


Figure 1.46 Empty Numeric Array

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

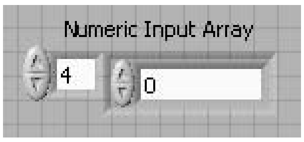


Figure 1.47 Defining Numeric Array Elements

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.

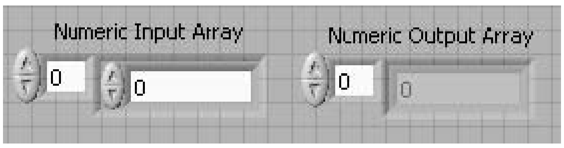


Figure 1.48 Creating Output Numeric Arrays

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 ofsets 0 through 10 as shown in Numeric Input and Output Arrays.

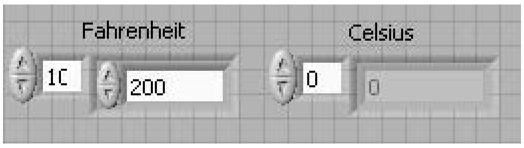


Figure 1.49 Numeric Input and Output Arrays

Right click in the Block Diagram window, navigate to **Programming** » **Structures** and click on **For Loop**.

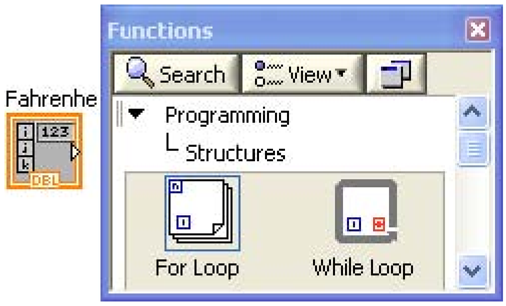


Figure 1.50 For Loop Structure

Click and drag to create the For Loop as shown in Creating For Loops and For Loop.

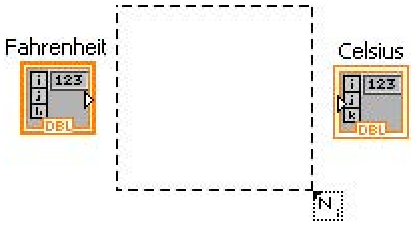


Figure 1.51 Creating For Loops

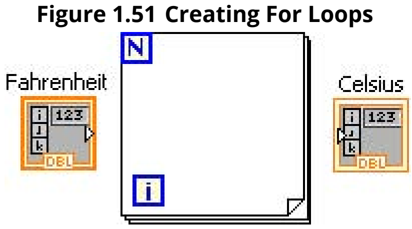


Figure 1.52 For Loop

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**.

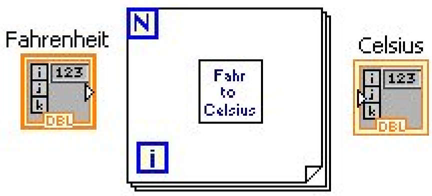


Figure 1.53 Function in Diagram

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.

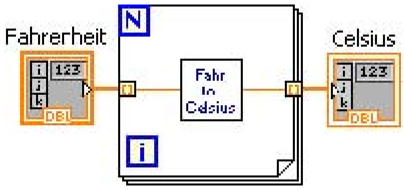


Figure 1.54 Wired Function in Diagram

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.

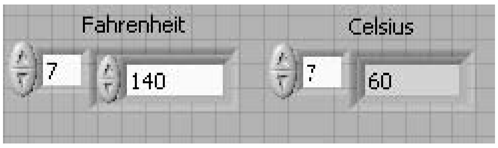


Figure 1.55 Fahrenheit to Celsius Arrays

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, 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 Block Diagram window, select the **While Loop** structure by clicking on it from the **Functions** » **Programming** » **Structures** menu.

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

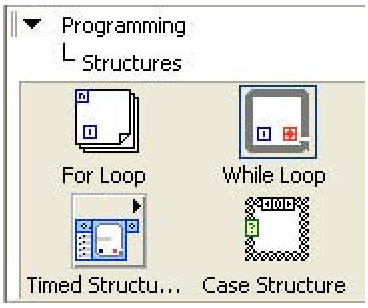


Figure 1.56 While Loop Structure

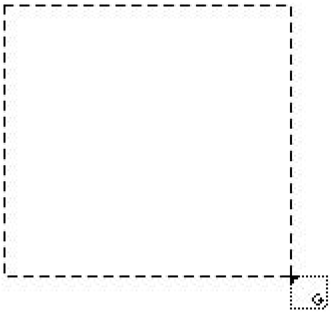


Figure 1.57 Creating a While Loop

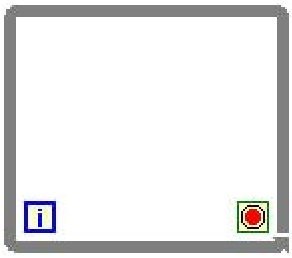


Figure 1.58 While Loop

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

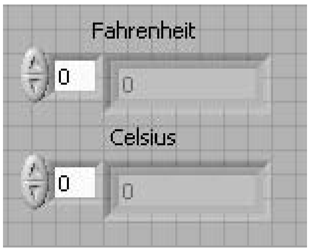


Figure 1.59 Numeric Output Arrays

Re-arrange the diagram as in While Loop Diagram .

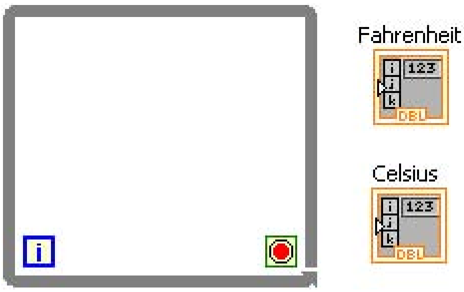


Figure 1.60 While Loop Diagram

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 Generating Fahrenheit Values

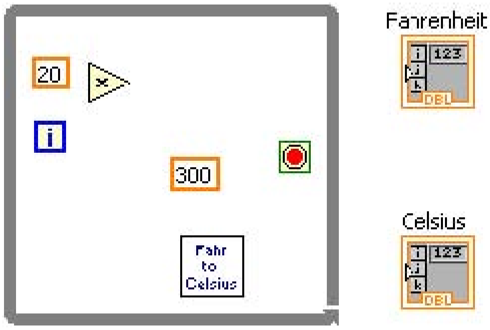


Figure 1.61 Generating Fahrenheit Values

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

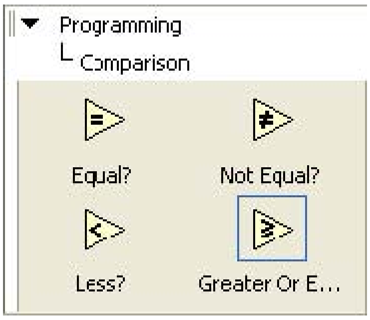


Figure 1.62 Greater or Equal Function Description

Wire the **While Loop** components as shown in Generating Fahrenheit Values & Stop Condition.

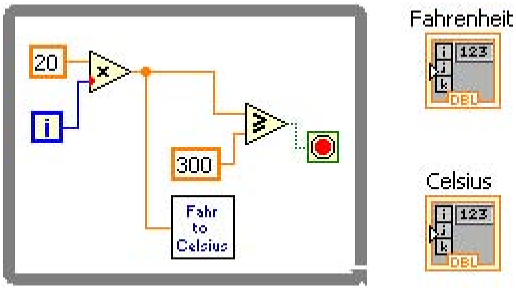


Figure 1.63 Generating Fahrenheit Values & Stop Condition

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 Broken Wires).

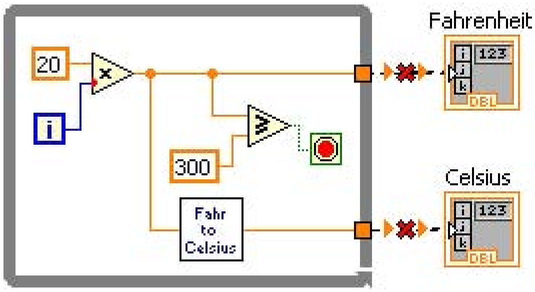


Figure 1.64 Broken Wires

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

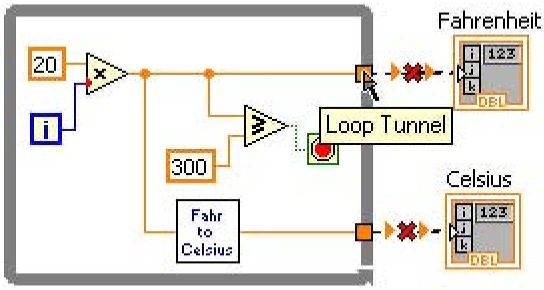


Figure 1.65 Loop Tunnel

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

Figure 1.66 [/topic/body/fig/title/title {"- topic/title "}) Enable Loop Indexing (title]

This enables values to accumulate and store the results into an array. Repeat for the **Celsius** array.

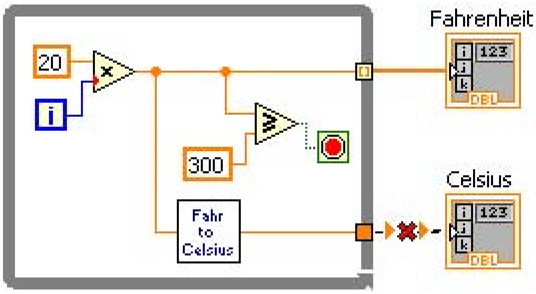


Figure 1.67 Broken Wire Repaired

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.

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

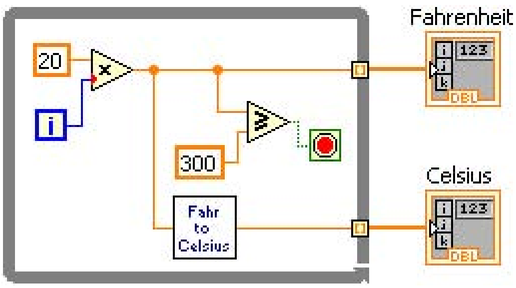


Figure 1.68 Fahrenheit to Celsius While Loop

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}

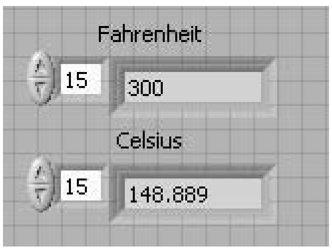


Figure 1.69 Fahrenheit and Celsius Arrays

1.8 Graphs

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

Figure 1.70 XY Graph Selection

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

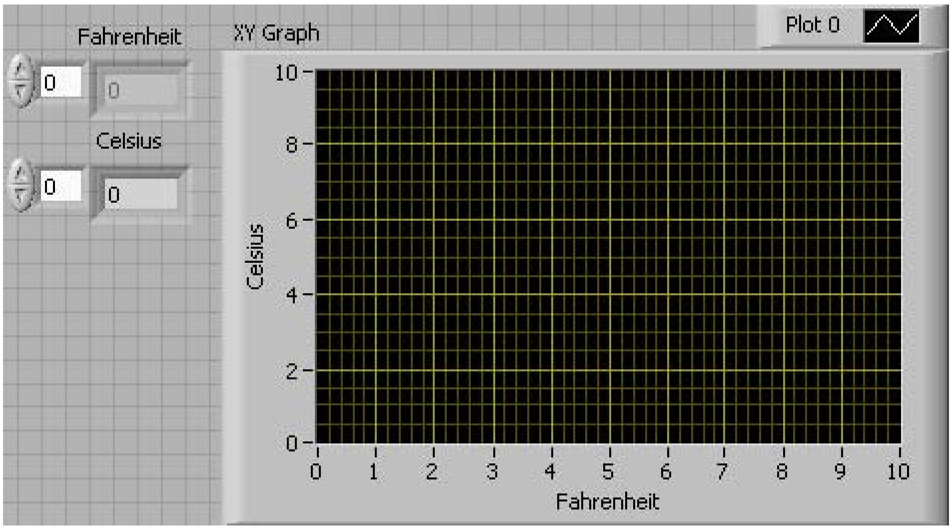


Figure 1.71 XY Graph in Front Panel window

The Block Diagram window contains the**XY Graph** terminal.

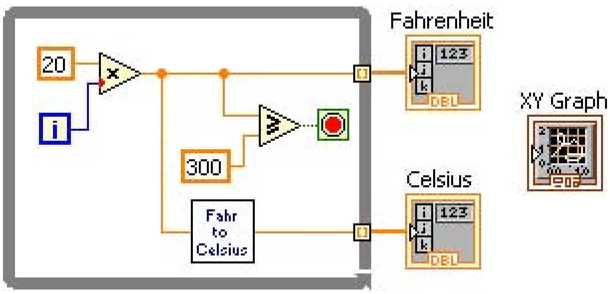


Figure 1.72 XY Graph Terminal in Diagram

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

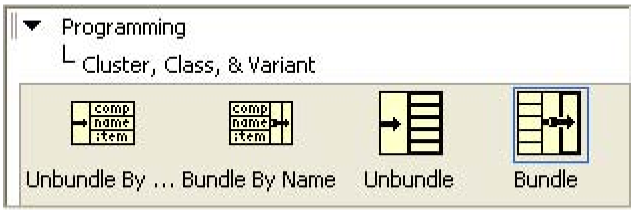


Figure 1.73 Bundle Operator

Drop it on the diagram as shown in Bundle for XY Graph.

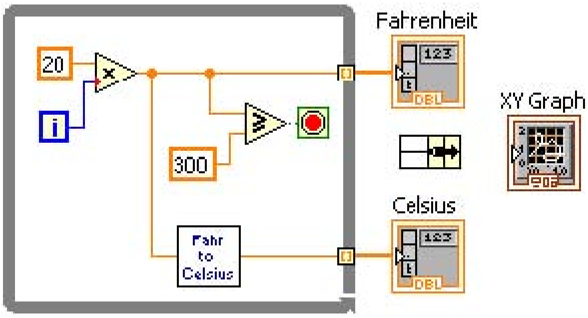


Figure 1.74 Bundle for XY Graph

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 fgure below.

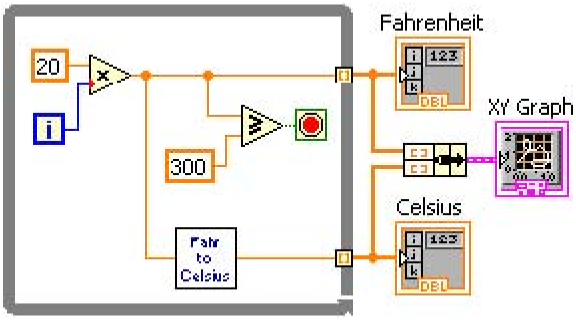


Figure 1.75 Wired XY Graph

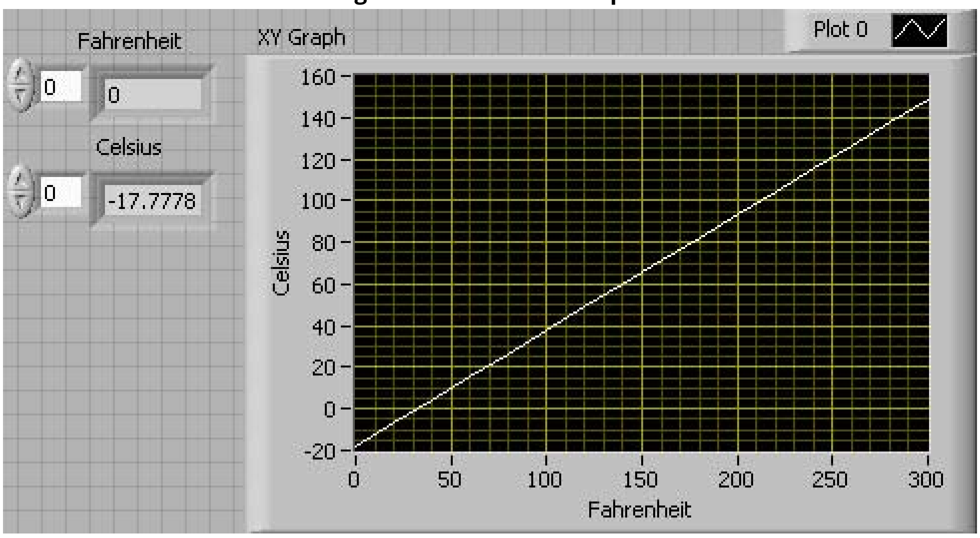


Figure 1.76 XY Graph Result

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.

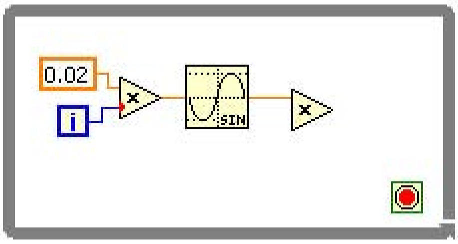


Figure 1.77 Creating Interactive Programs

In the Front Panel window, from the **Functions** » **Modern** » **Numeric** select the vertical pointer slide. From the **Functions** » **Modern** » **Graph** select **Waveform Chart**.

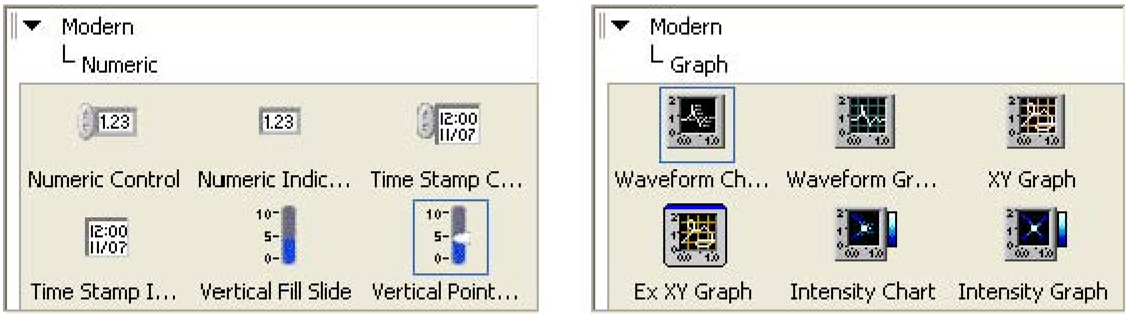


Figure 1.78 Vertical Pointer Slide and Waveform Chart

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.

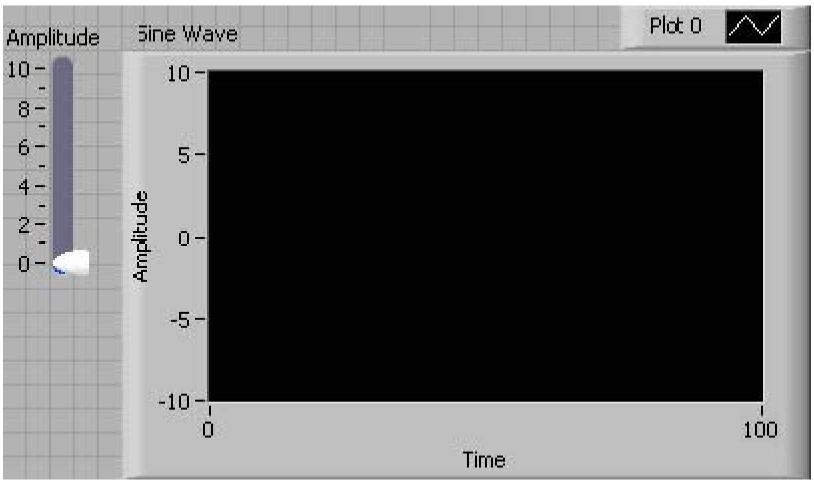


Figure 1.79 Slide & Waveform Chart in Front Panel window

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

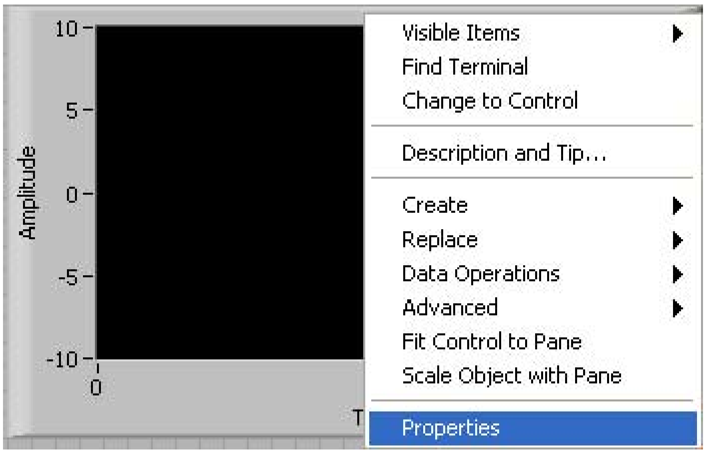


Figure 1.80 Selecting Chart Properties

Select the **Scales** tab and change **Maximum** to 1023. **Sine Wave** will display 1024 samples. Click on the down arrow located to the right of Time (XAxis) and select Amplitude (YAxis).

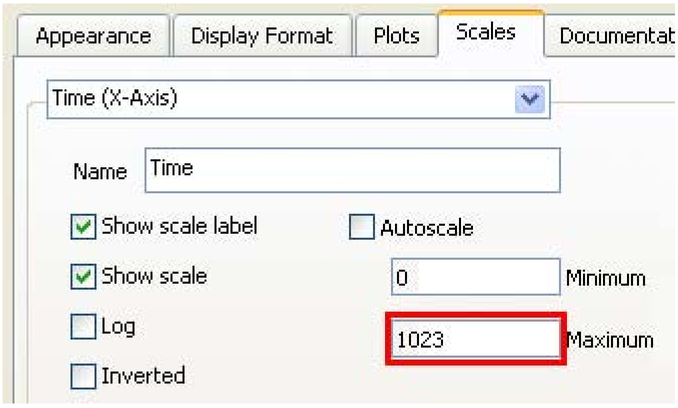


Figure 1.81 X-Axis Maximum

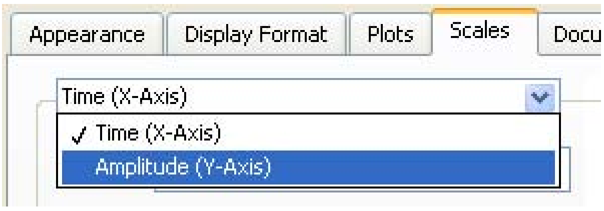
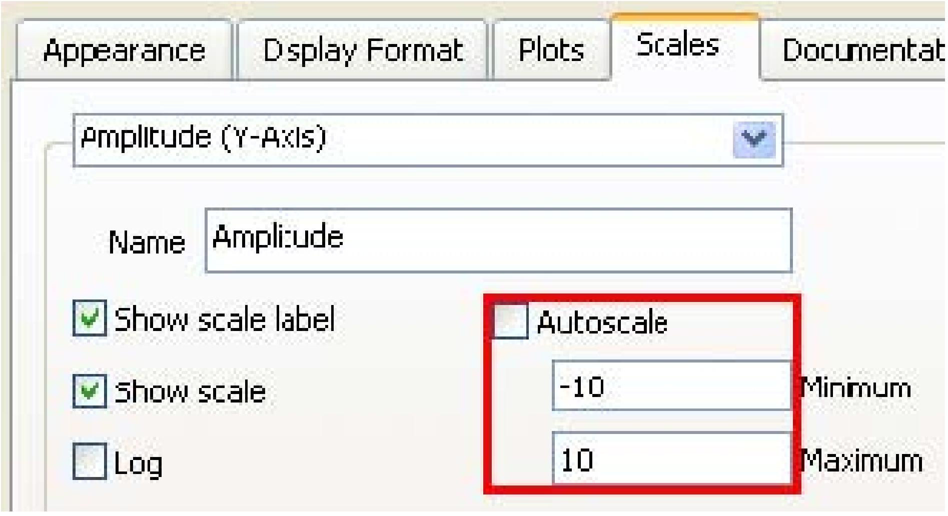


Figure 1.82 Selecting Y-Axis

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

De-Selecting Autoscale



In the Block Diagram window, re-arrange the **Amplitude** and **Sine Wave** terminals and finish the program as shown in Interactive Sine Wave Diagram.

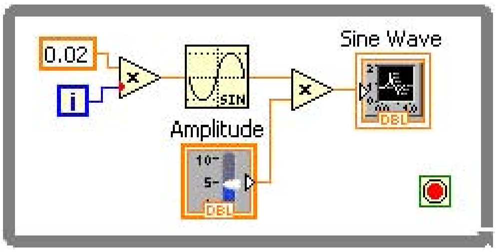


Figure 1.83 Interactive Sine Wave Diagram

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

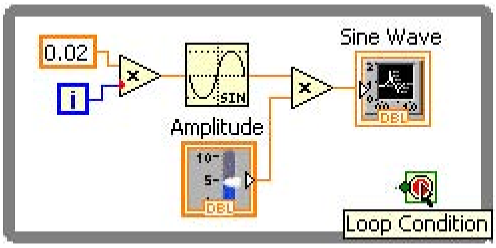


Figure 1.84 Loop Condition

And right click on the**Loop Control** and from the pop-up menu select**Create Control**. A stop terminal is created...

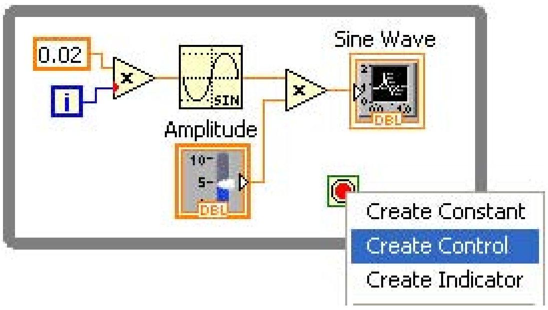


Figure 1.85 Create Loop Control

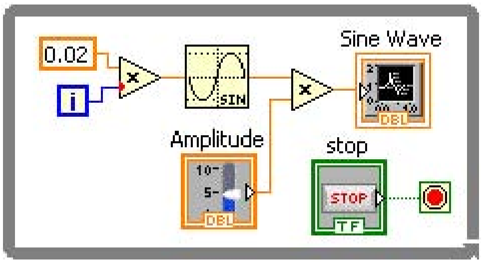


Figure 1.86 Interactive G Program

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

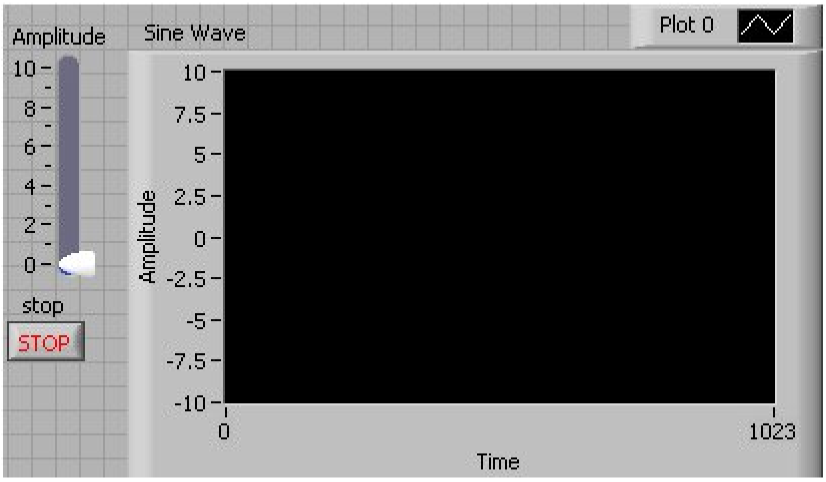


Figure 1.87 Interactive Program

Run the G program.

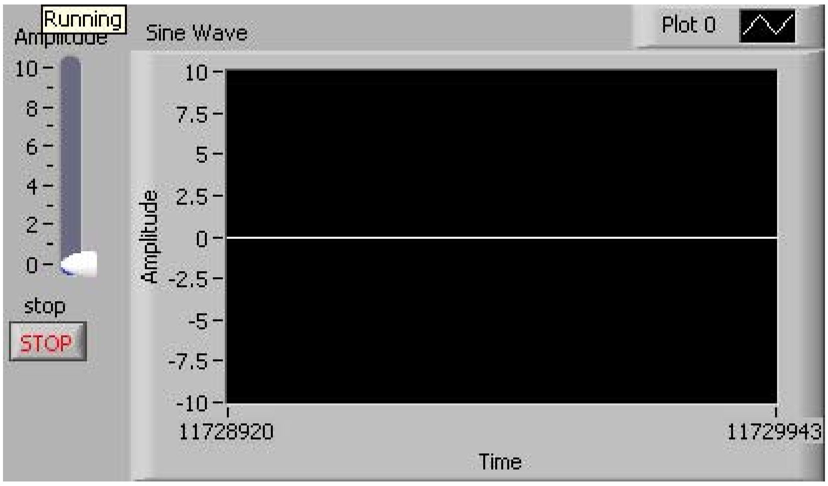


Figure 1.88 Interactive Program

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

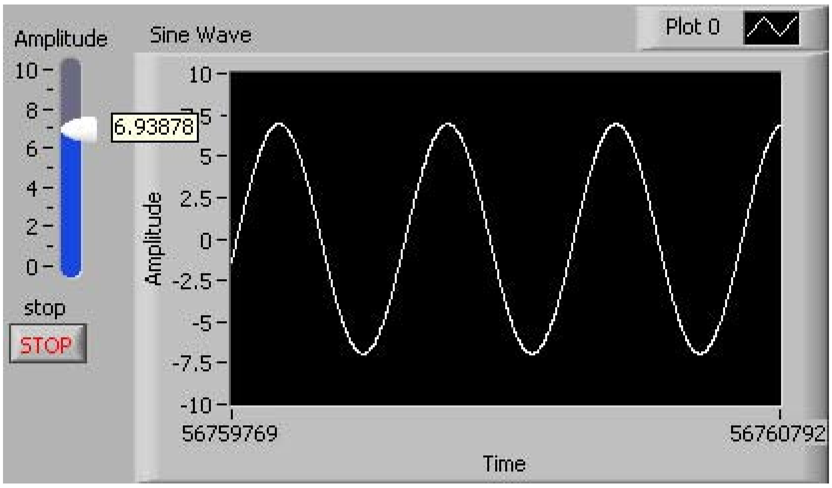


Figure 1.89 Interactive Program

To end the G program, simply click on the **stop** button. Congratulations. You have successfully completed and executed your first interactive G program.

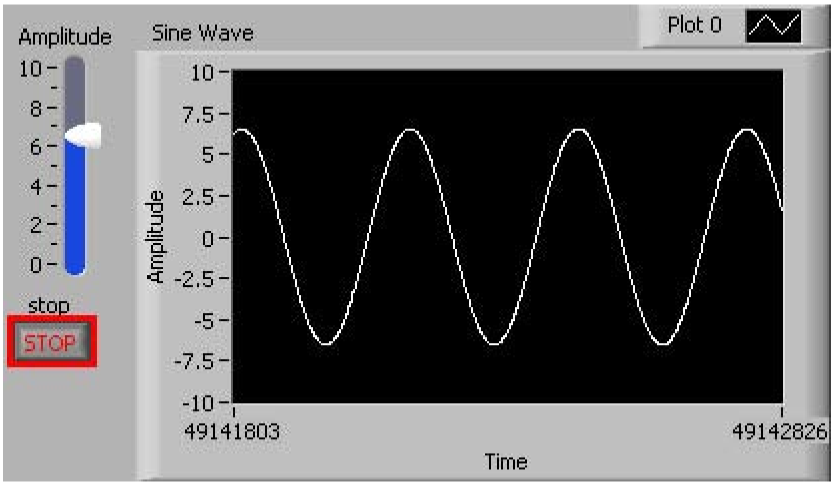


Figure 1.90 Interactive Program

1.10 Parallel Programming

Save a copy of Interactivity.vi as Parallel Programming.vi. Select the while loop as shown in Select Diagram for Parallel Programming.

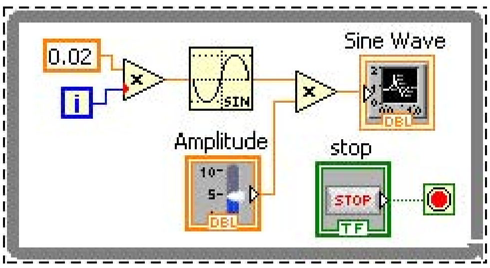


Figure 1.91 Select Diagram for Parallel Programming

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

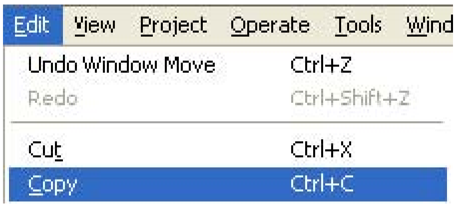


Figure 1.92 Copy Selected Diagram

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

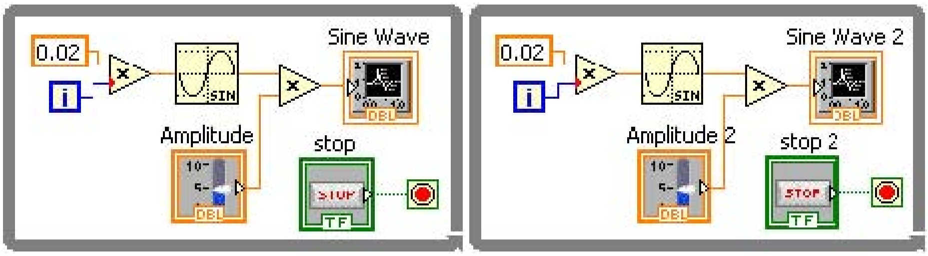


Figure 1.93 Paste Diagram

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

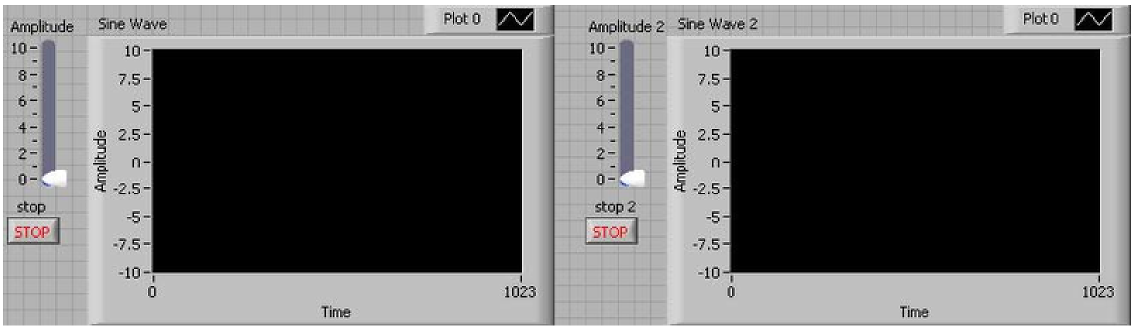


Figure 1.94 Parallel G Program

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

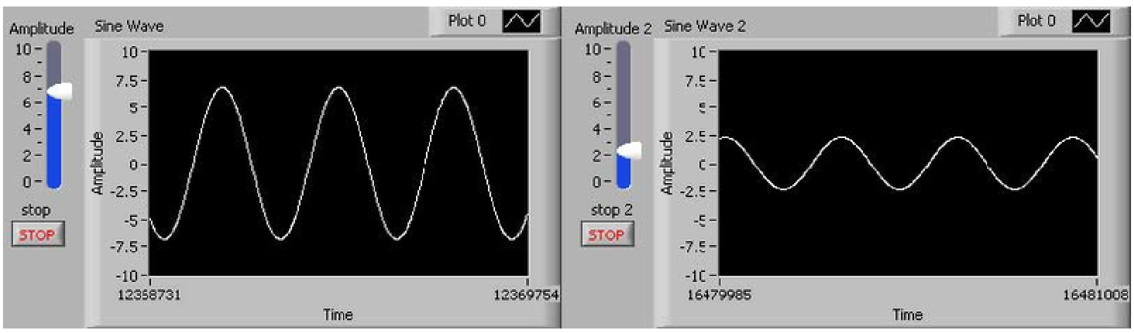


Figure 1.95 Parallel Interactive G Program

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 multicore G program.

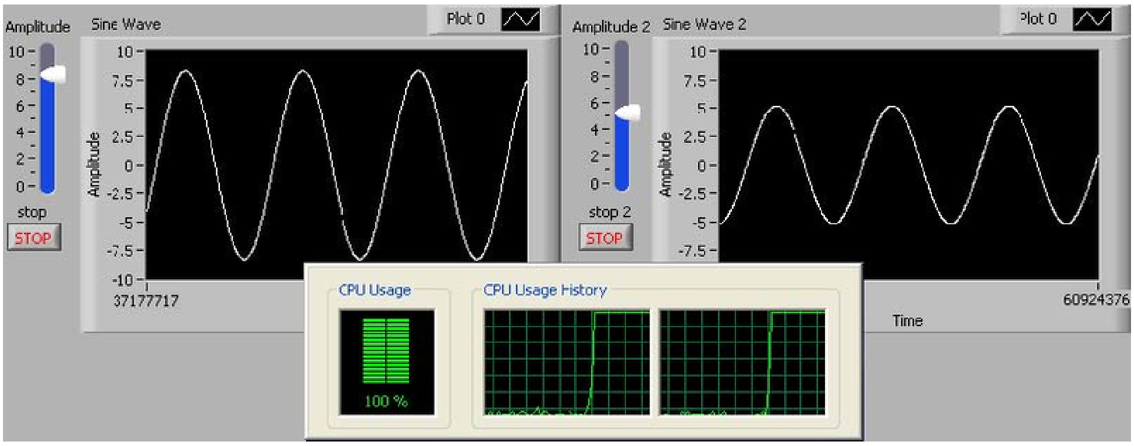


Figure 1.96 Interactive Multicore G Program

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.

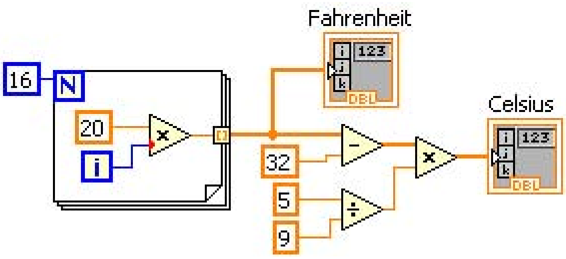


Figure 1.97 Polymorphic G Diagram

Chapter 2 Data Types

Data Type	Block Diagram Terminal		Interactive User Interface		Comments
	Input	Output	Input	Output	
Extended					80-bit floating pointnumeric
Double					64-bit floating pointnumeric
Single					32-bit floating pointnumeric
ComplexExtended					160-bit floating pointnumeric
ComplexDouble					128-bit floating pointnumeric

Figure 2.1

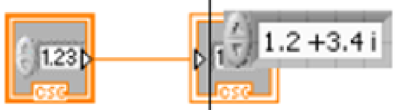
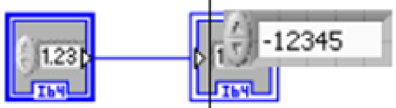
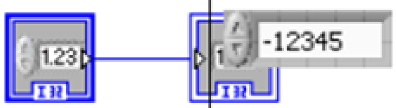
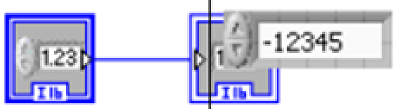
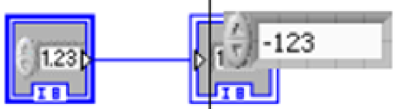
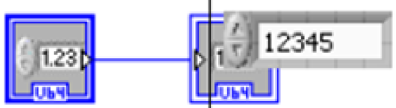
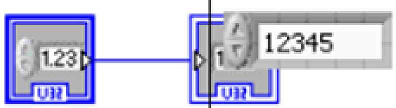
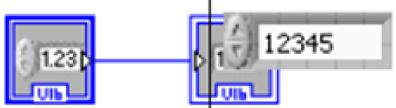
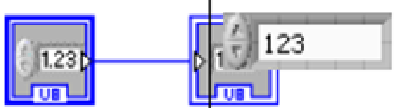
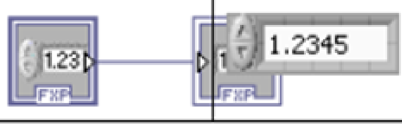
ComplexSingle			64-bit floating pointnumeric
Integer 64			64-bit signed integernumeric
Integer 32			32-bit signed integernumeric
Integer 16			16-bit signed integernumeric
Integer 8			8-bit signed integernumeric
UnsignedInteger 64			64-bt unsigned integer numeric
UnsignedInteger 32			32-bit unsigned integer numeric
UnsignedInteger 16			16-bit unsigned integer numeric
UnsignedInteger 8			8-bit unsigned integer numeric
Fixed Point			Mantissa,fractional fixed point numeric representation

Figure 2.2

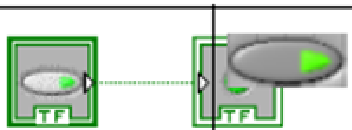
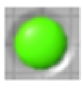
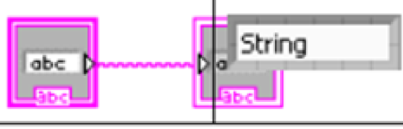
Boolean			True or False.
String			Text

Figure 2.3

Chapter 3 Operators

3.1 Numeric

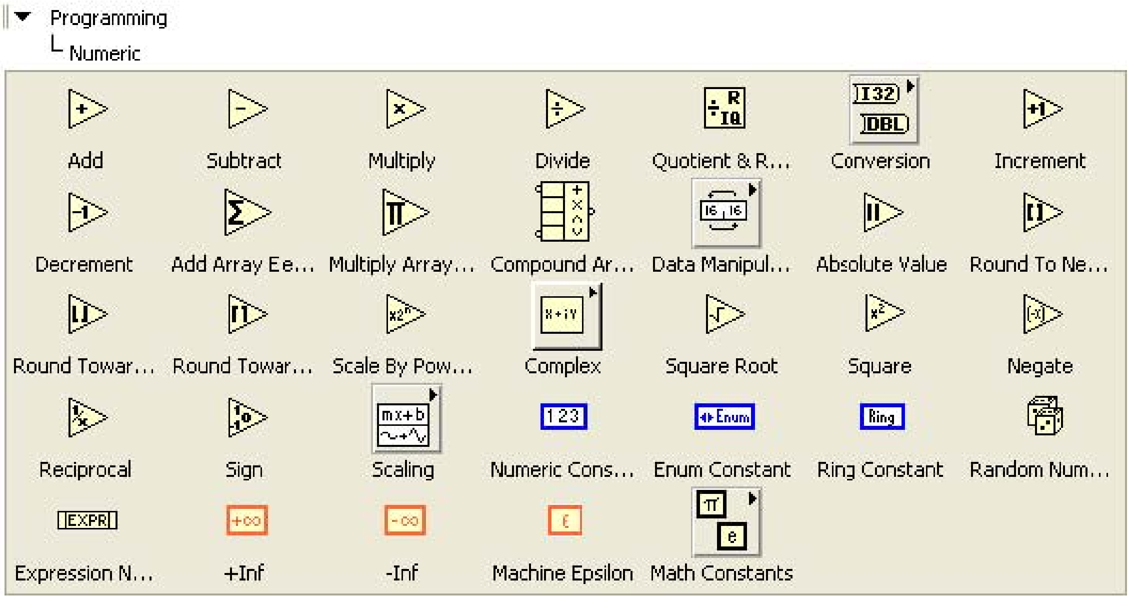


Figure 3.1 Numeric Operators

Figure 3.2 Complex Numeric Operations Figure : Numeric Conversion Operators

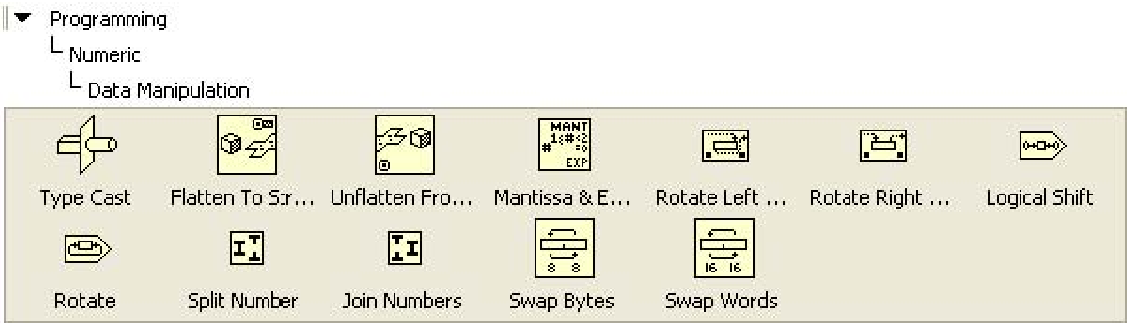


Figure 3.3 Numeric Data Manipulation Operators

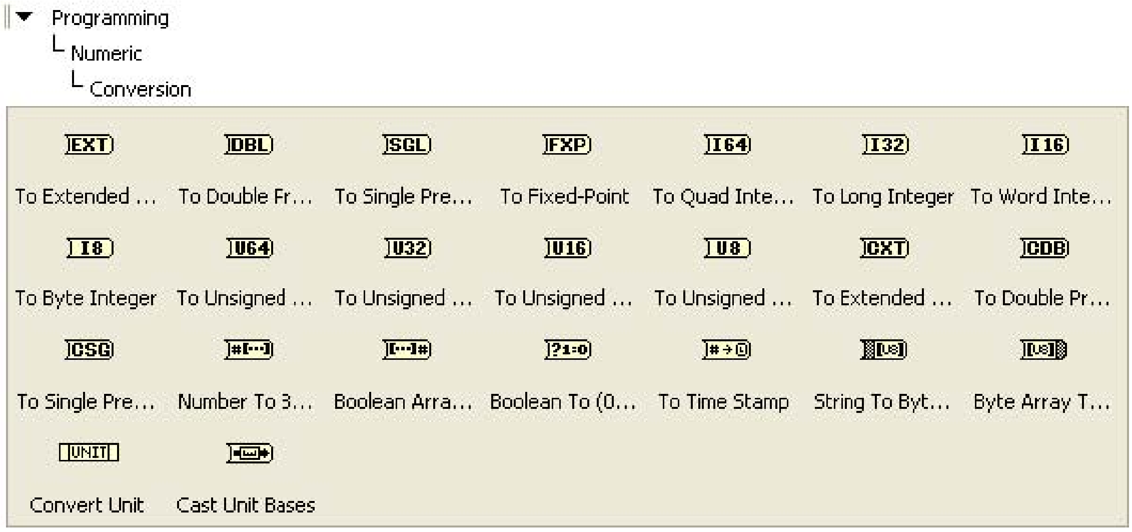


Figure 3.4 Numeric Conversion Operators

3.2 Boolean

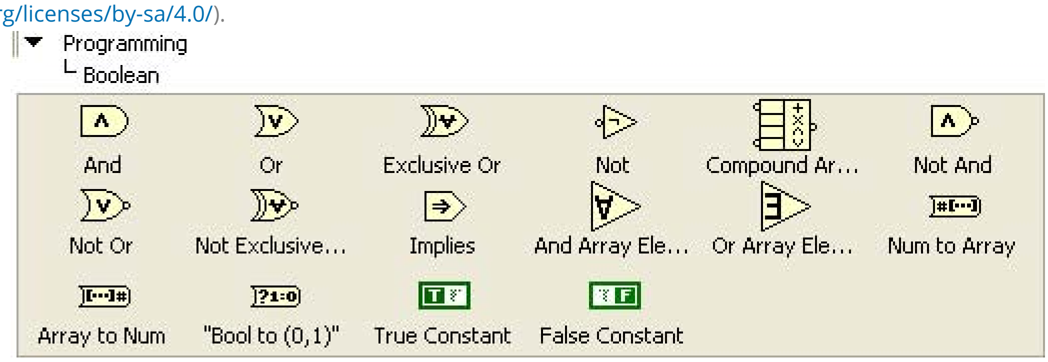


Figure 3.5 Boolean Operators

3.3 Comparison

Figure 3.6 Comparison Operators

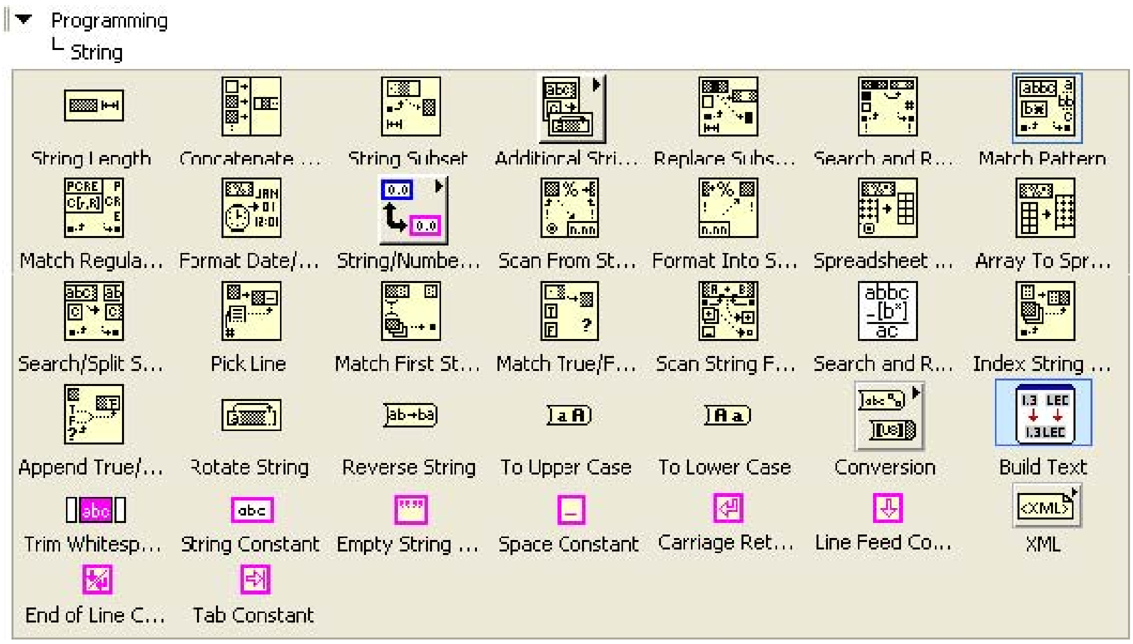


Figure 3.7 String Operators

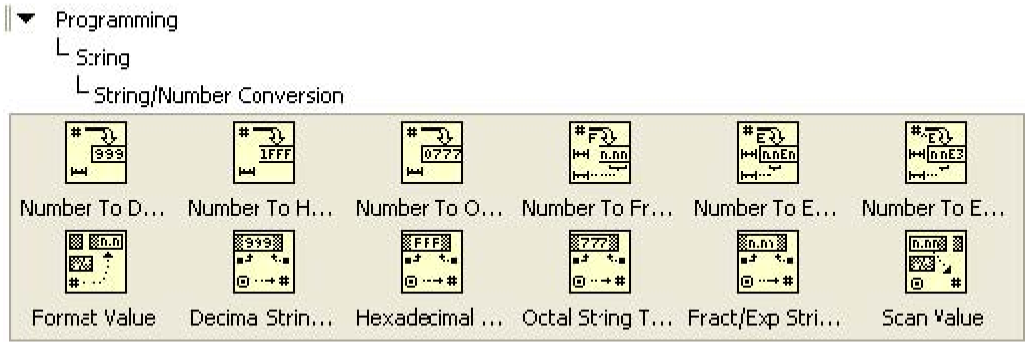


Figure 3.8 String/Number Operators

3.4 Math

3.4.1 Math Constants

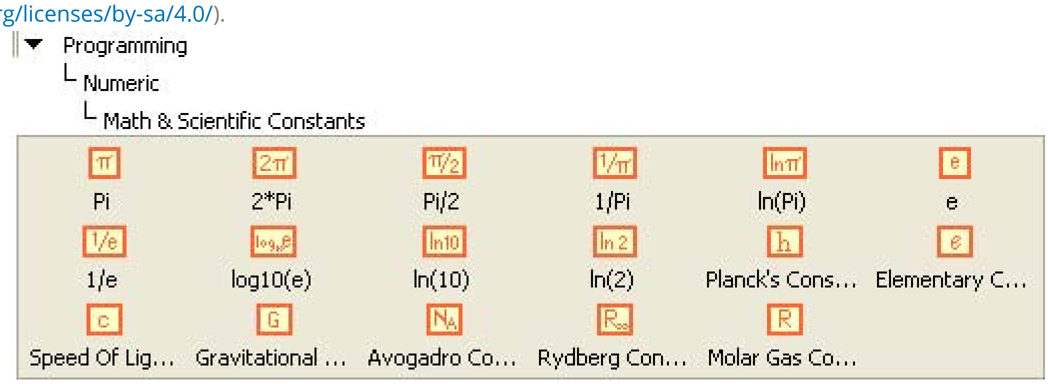


Figure 3.9 Mathematical Constants

3.4.2 Trigonometric Functions

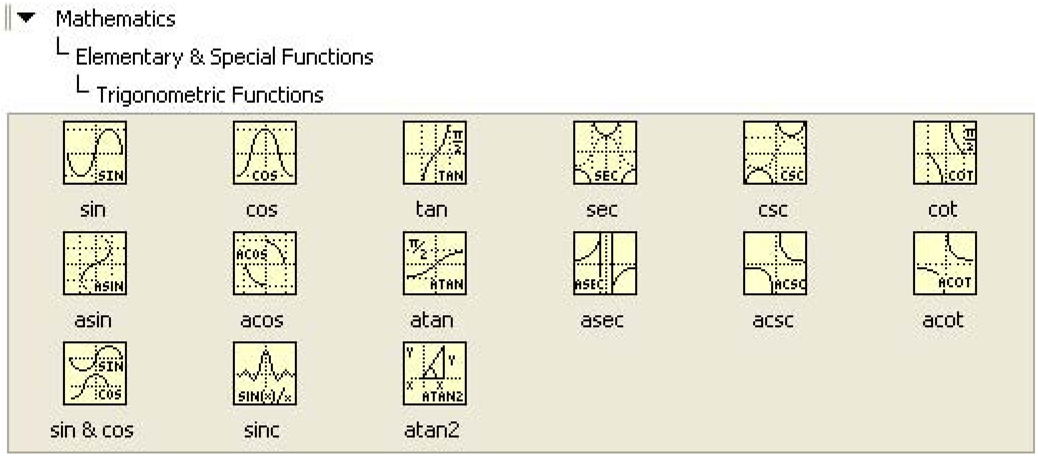


Figure 3.10 Trigonometric Functions

3.4.3 Exponential and Logarithmic Functions

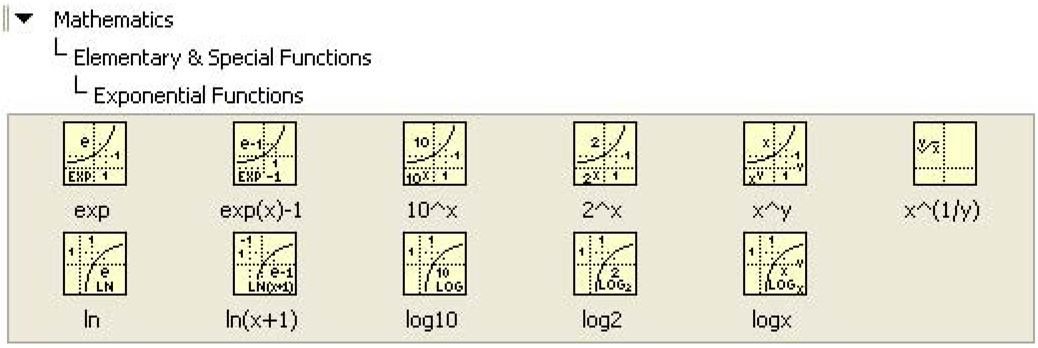


Figure 3.11 Exponential and Logarithmic Functions

3.4.4 Hyperbolic Functions

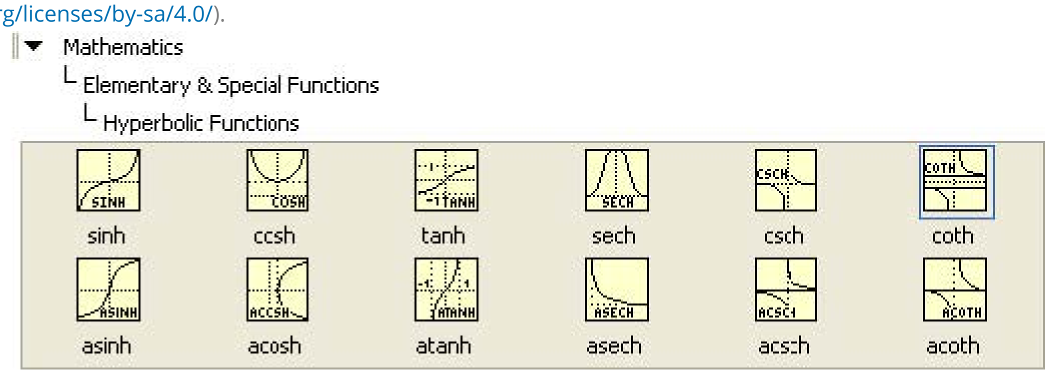


Figure 3.12 Hyperbolic Functions

Chapter 4 Arrays and Clusters

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

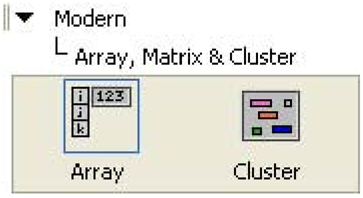


Figure 4.1 Array Structure

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 Front Panel window, the data type of the array is undefined as indicated by the grayed out portion of the array.

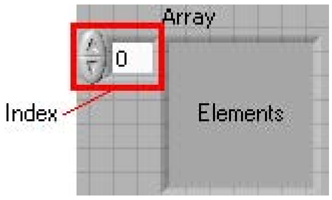


Figure 4.2 Index and Elements of an Array

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.

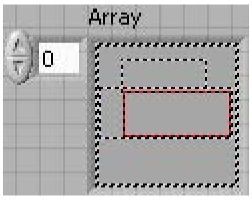


Figure 4.3 Creating Arrays

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

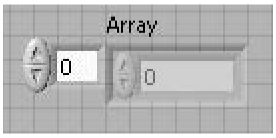


Figure 4.4 Empty Arrays

To define elements of an input array, select the element's **index** and enter the appropriate value. Figure 6.5 defines a numeric array with one element at **index** 0.

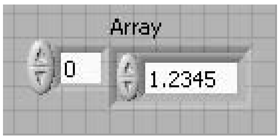


Figure 4.5 Defining Array Elements

G arrays are zero-based. The last element index of an **N** element array is **N1**. Last Array Element and Undefined Nth Element are those of a 10 element array.

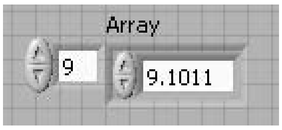


Figure 4.6 Last Array Element

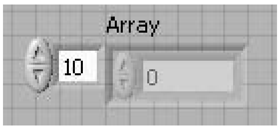


Figure 4.7 Undefined Nth Element

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.

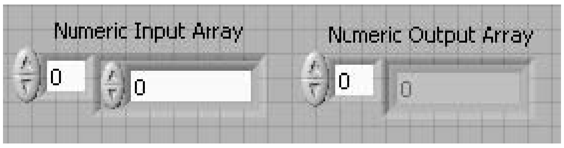


Figure 4.8 Input and Output Arrays

4.1 Multidimensional Arrays

To create multidimensional arrays, click on the array's **index** and select **Add Dimension** from the menu. Multidimensional Array shows a 2-dimensional array.

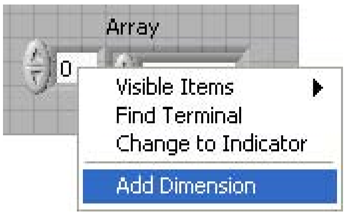


Figure 4.9 Creating Multidimensional Arrays

Figure 4.10 Multidimensional Array

4.2 Array Operators

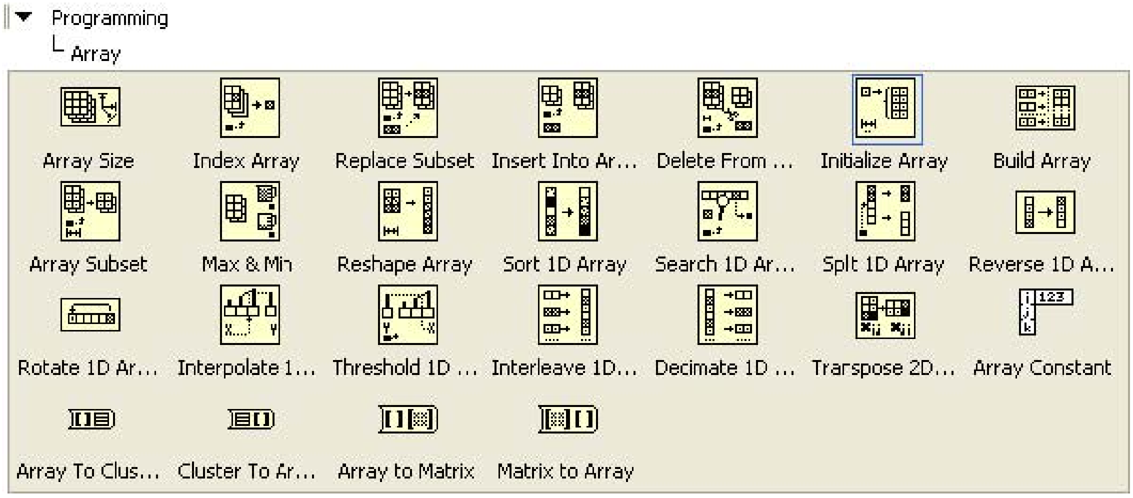


Figure 4.11 Array Operators

4.3 Clusters

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

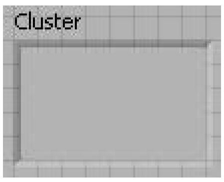


Figure 4.12 Empty Cluster

Select the various data types and drag them onto the **cluster** structure. Figure Figure 4.13shows an **Error Cluster** consisting of a Boolean **Error**, a numeric **ID** and a string **Message** data types.

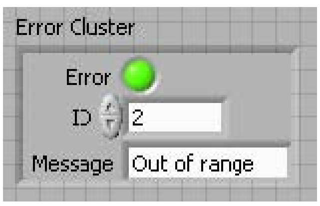


Figure 4.13 Cluster Example

4.4 Cluster Operators

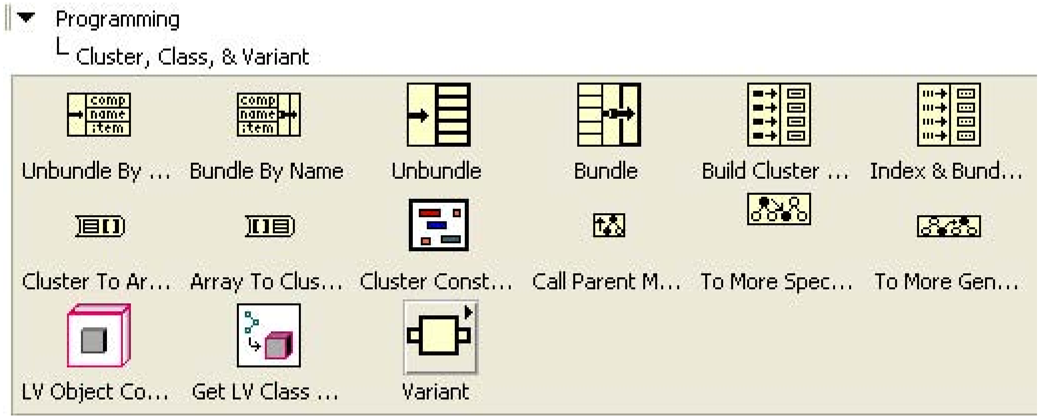


Figure 4.14 Cluster Operators

Chapter 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**.

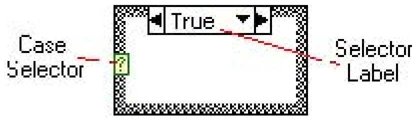


Figure 5.1 Case Structure

5.1.1 Boolean Selection

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

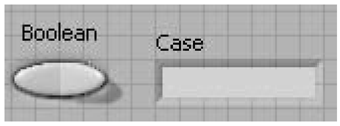


Figure 5.2 Case Selection User Interface

Arrange the diagram to look as in Case Selection G Diagram.

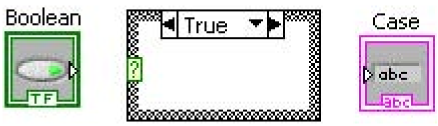


Figure 5.3 Case Selection G Diagram

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

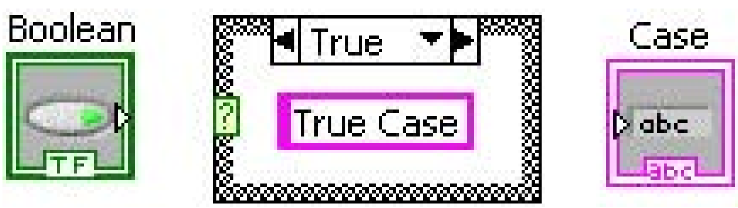


Figure 5.4 True Case Diagram

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.

Selecting False Case

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

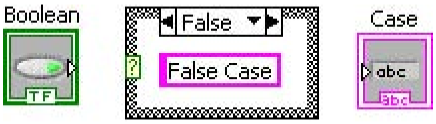


Figure 5.5 False Case Diagram

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

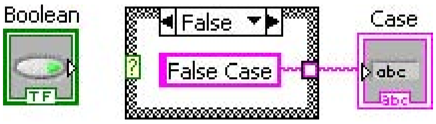


Figure 5.6 Wiring Case Structures

Select the**True** case and wire the string constant to the **case structure tunnel**. Complete the diagram as shown in Completed Case Diagram.

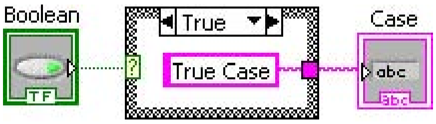


Figure 5.7 Completed Case Diagram

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 Front Panel window, toggle the Boolean input control and run the program.

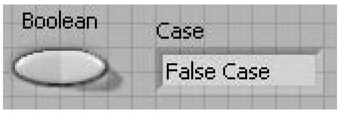


Figure 5.8 False Selection

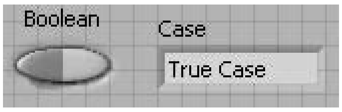


Figure 5.9 True Selection

5.1.2 Multicase Selection

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

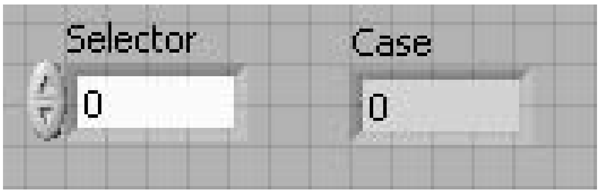


Figure 5.10 Multicase GUI

In the Block Diagram window, create a **case structure**, select the **False** case and arrange the terminals as shown in Multicase.

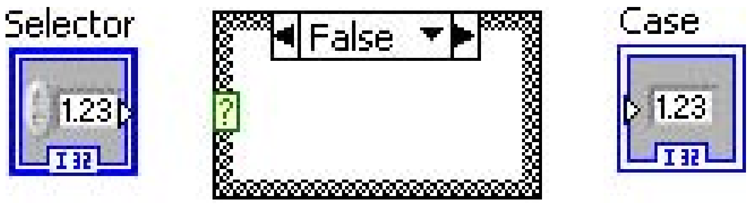


Figure 5.11 Multicase

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

Figure 5.12 Multicase Selector

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

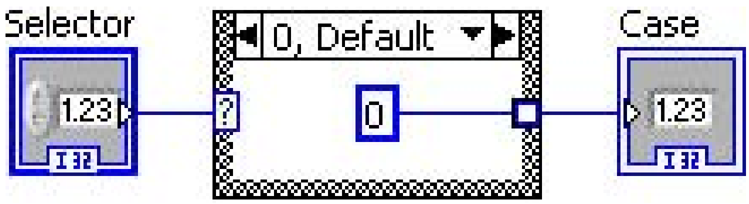


Figure 5.13 Default Case

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 Case 1.

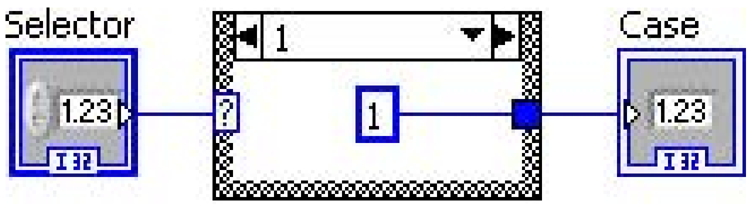


Figure 5.14 Case 1

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

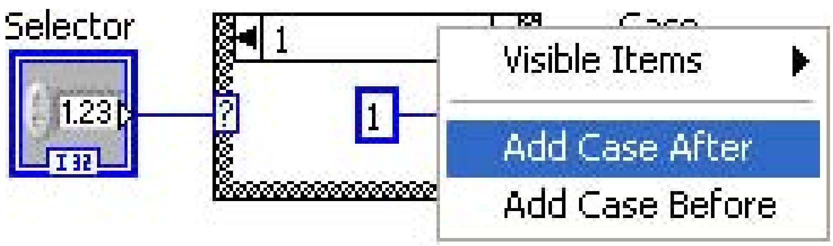


Figure 5.15 Adding Cases

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

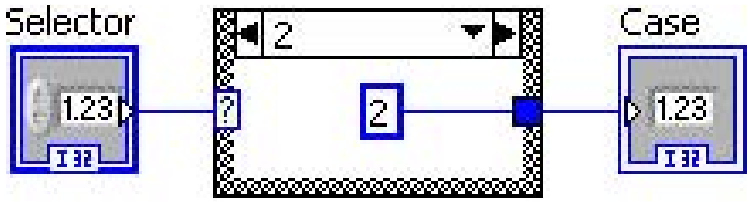


Figure 5.16 Case 2

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

Figure 5.17 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.

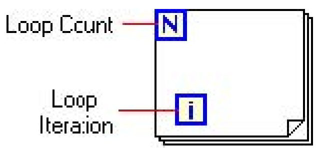


Figure 5.18 For Loop Structure

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**N1**. This is illustrated in Loop Count and Final Loop Iteration.

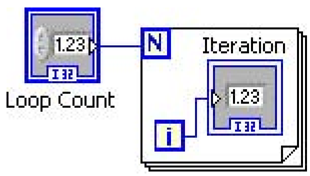


Figure 5.19 Loop Count

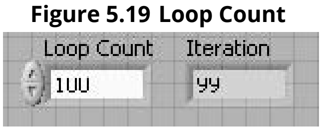


Figure 5.20 Final Loop Iteration

5.2.1 Shift Registers

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

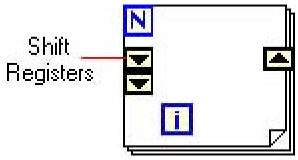


Figure 5.21 **Shift Registers**

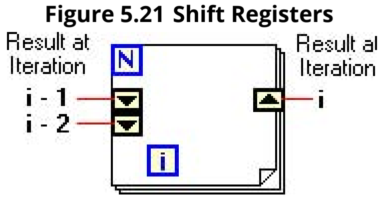


Figure 5.22 **Shift Registers**

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

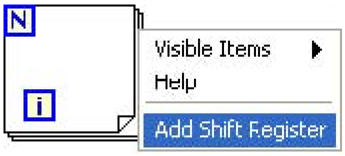


Figure 5.23 **Adding Shift Registers**

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

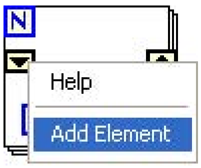


Figure 5.24 **Adding Shift Register Elements**

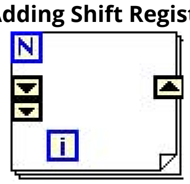


Figure 5.25 **Adding Shift Register Elements**

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

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

Figure 5.26 **(7.1)**

In the Front Panel window, select an integer 32 numeric input and output controls and labeled them **n** and **Fib(n)** respectively. Arrange the diagram as shown in Shift Register Example.

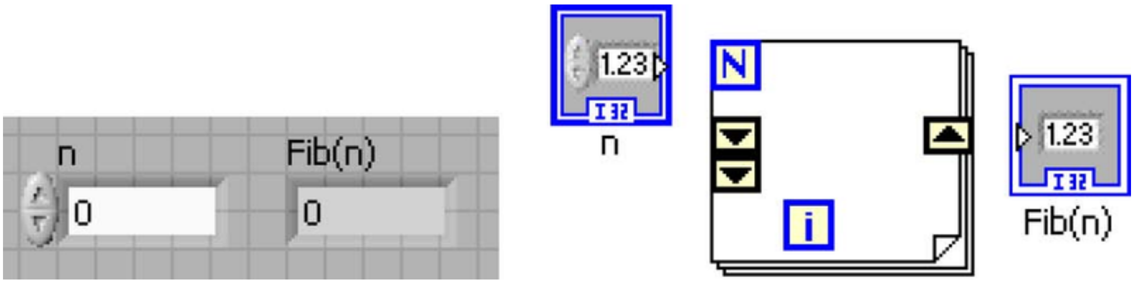


Figure 5.27 Shift Register Example

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 Fibonacci G Program.

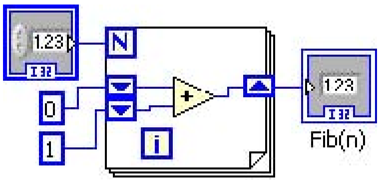


Figure 5.28 Fibonacci G Program

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 7.29 shows the result of Fib(8).

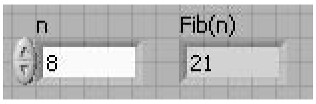


Figure 5.29 Fib(8) = 21

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.

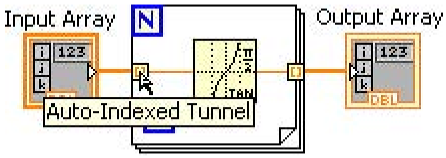


Figure 5.30 For Loop Auto-Indexing

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.

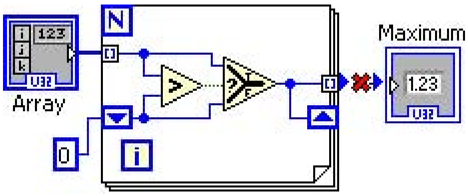


Figure 5.31 Broken Auto-Indexing

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

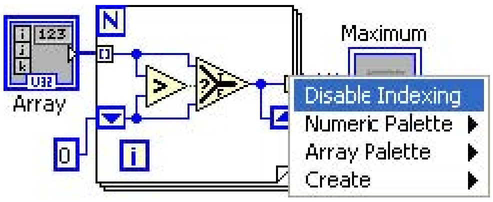


Figure 5.32 Disabling Auto-Indexing

The final diagram with the **Auto-Indexed Tunnel** disabled is shown in Disabled Auto-Indexing.

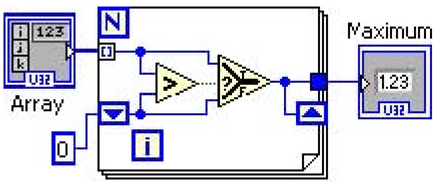


Figure 5.33 Disabled Auto-Indexing

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.

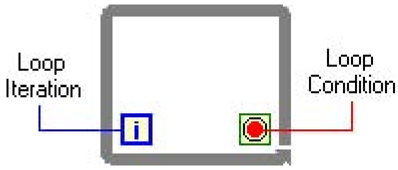


Figure 5.34 While Loop Structure

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 the following Figure 5.35 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**.

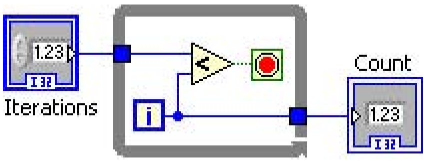


Figure 5.35 Stop If True

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.

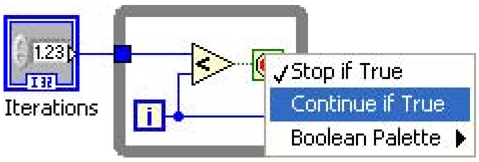


Figure 5.36 Changing Loop Condition

Continue If True shows the **Loop Condition** set to **Continue if True**.

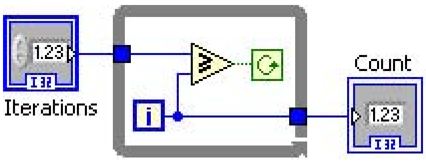


Figure 5.37 Continue If True

5.3.2 Shift Registers

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

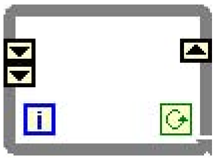


Figure 5.38 While Loop Shift Registers

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

$$e = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \sum_{n=0}^{n=\infty} \frac{1}{n!} = 1 + + \sum_{n=0}^{n=\infty} \frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots = 2.7182818284$$

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.

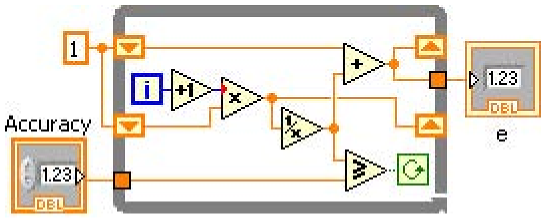


Figure 5.39 Computing e

Save the program as **e.vi.** The result of running this program is shown in Computed e to 5 Digits.

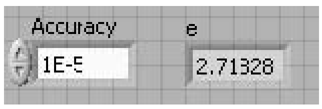


Figure 5.40 Computed e to 5 Digits

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.

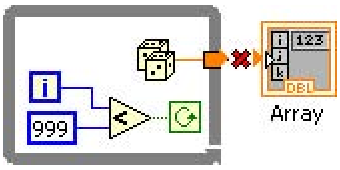


Figure 5.41 Disabled Auto-Indexing

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

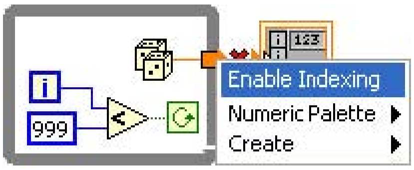


Figure 5.42 Enabling Auto-Indexing

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

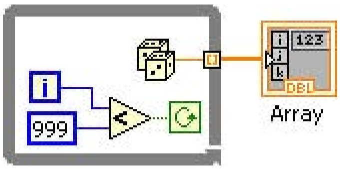


Figure 5.43 Auto-Indexing Enabled

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.

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.

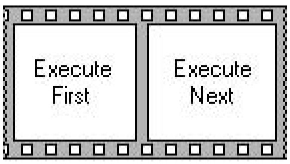


Figure 5.44 Sequence Structure

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.

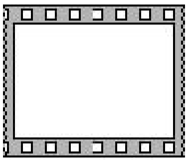


Figure 5.45 Sequence Frame

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 5.46 Adding Sequence Frames

Add two more frames to the sequence structure to get a three frame sequence as shown in Three Frame Sequence.

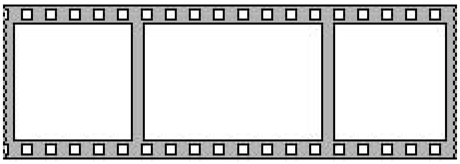


Figure 5.47 Three Frame Sequence

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

Figure 5.48 Tick Count Function

Drop the **Tick Count (ms)** function in the frst (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 Start and Stop Tick Counts.

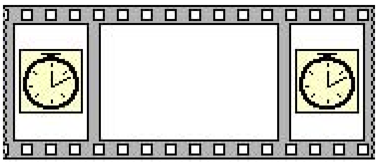


Figure 5.49 Start and Stop Tick Counts

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

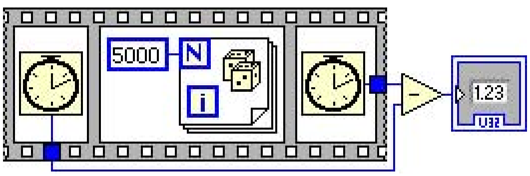


Figure 5.50 Timing G Program

5.4.2 Stacked Sequence

A **Stacked Sequence** provides a more compact representation 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 Stacked Sequence).

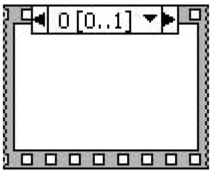


Figure 5.51 Stacked Sequence

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.

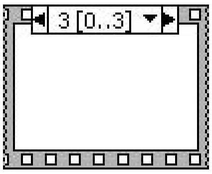


Figure 5.52 Four Frame Stacked Sequence

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.

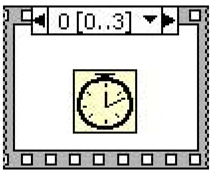


Figure 5.53 Adding Sequence Locals

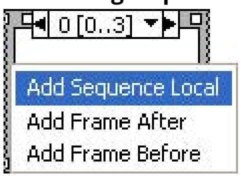


Figure 5.54 Adding Sequence Locals

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.

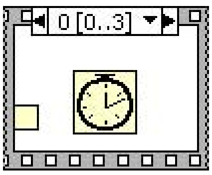


Figure 5.55 Sequence Local

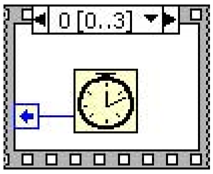


Figure 5.56 Sequence Local

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

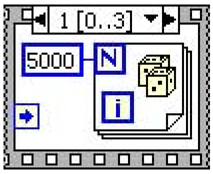


Figure 5.57 Frame to Time

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 Stop Time Stamp.

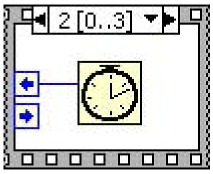


Figure 5.58 Stop Time Stamp

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 Stacked Timing G Program. To complete the diagram, wire the output of the **Subtract** function to the unsigned integer 32 output.

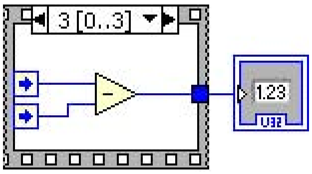


Figure 5.59 Stacked Timing G Program

It is important to note that the programs in Timing G Program and Stacked Timing G Program are programmatically identical.

Chapter 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

6.1 Connectors

Open the **Fibonacci.vi** for this example.

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

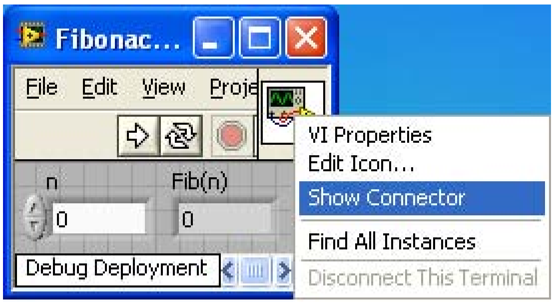


Figure 6.1 Show Connector Pane

This brings up the connector pane as shown in Connector Pane.

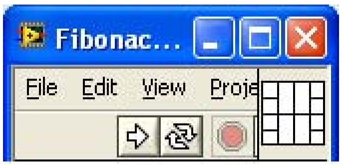


Figure 6.2 Connector Pane

Right click on the connector pane and select **Patterns**. A menu with connector patterns is presented from which you can select the appropriate pattern. For this example select the pattern highlighted in Select Connector Pattern.

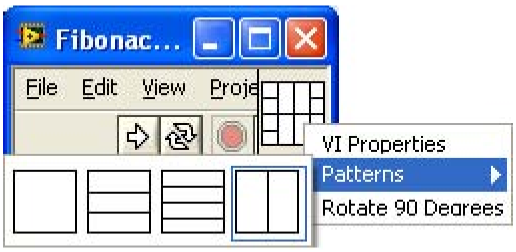


Figure 6.3 Select Connector Pattern

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

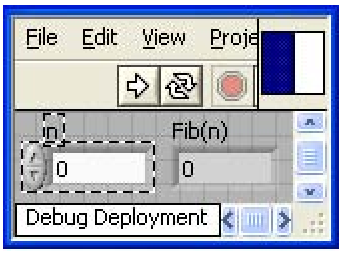


Figure 6.4 Associating Terminals

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

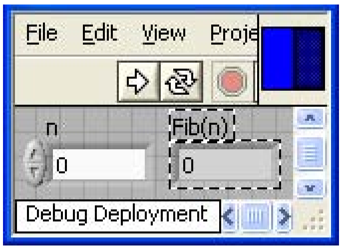


Figure 6.5 Connected Terminals

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: Icon Editor). 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.

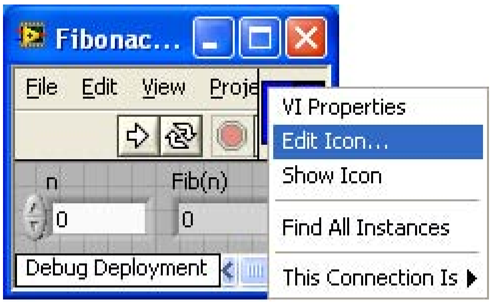


Figure 6.6 Selecting Icon Editor

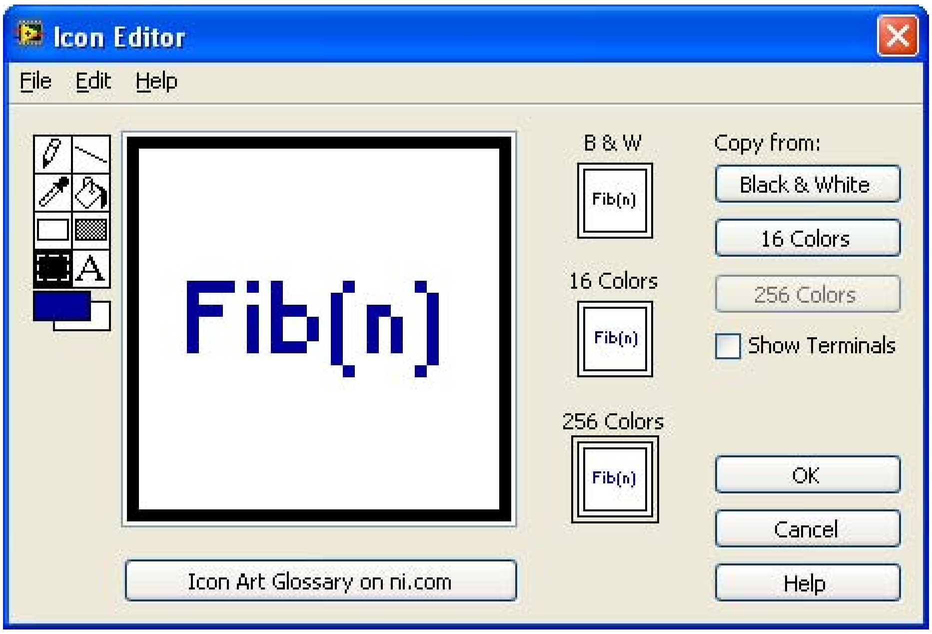


Figure 6.7 Icon Editor

6.3 Invoking Functions

To invoke functions, right click on the Block Diagram 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 6.8 Invoking Functions

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

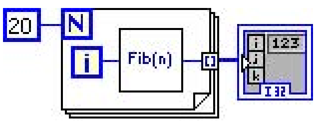


Figure 6.9 Fibonacci Series

Chapter 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:

yi = sin (0.2xi) (9.1)

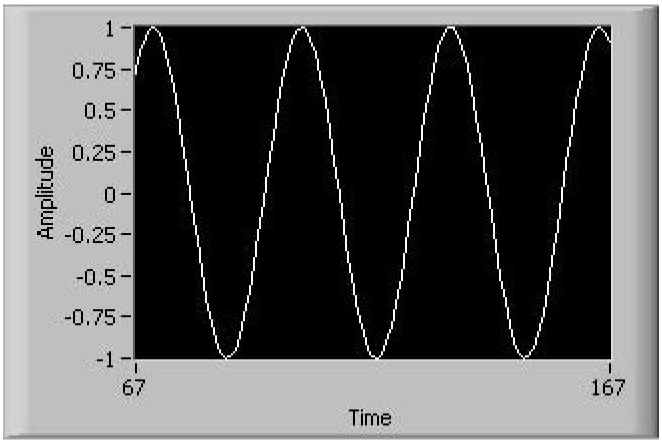


Figure 7.1 Waveform Chart

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 the following Figure Figure 7.2.

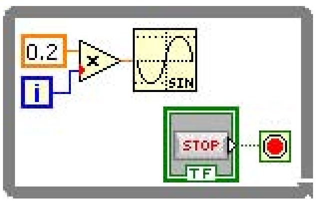


Figure 7.2 While Loop For Waveform Chart

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

Figure 7.3 Selecting Waveform Chart

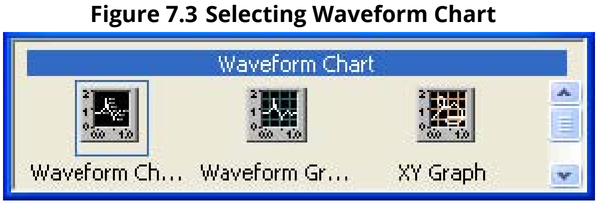


Figure 7.4

This places the Waveform Chart in the Front Panel window.

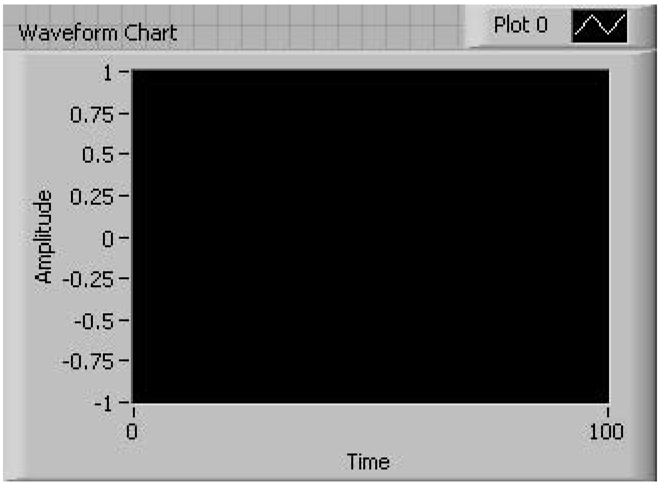


Figure 7.5 Waveform Chart in Front Panel window

In the Block Diagram 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.

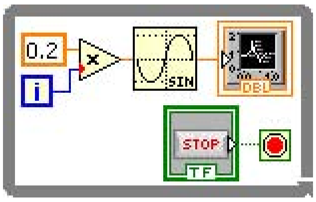


Figure 7.6 Waveform Chart Terminal

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.

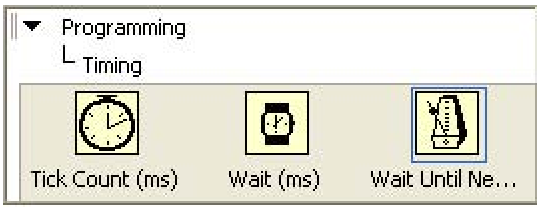


Figure 7.7 Wait Until Next ms Multiple

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 Waveform Chart Program.

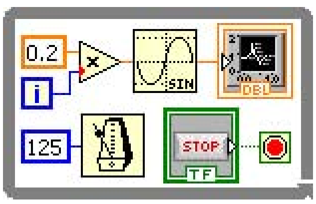


Figure 7.8 Waveform Chart Program

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

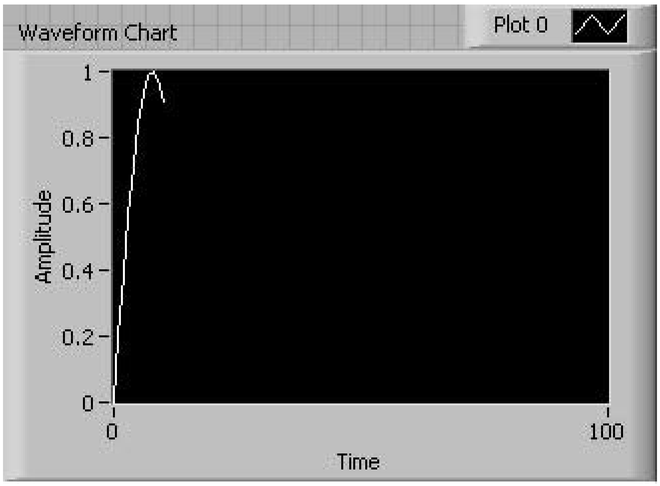


Figure 7.9 Waveform Chart Autoscaling

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.

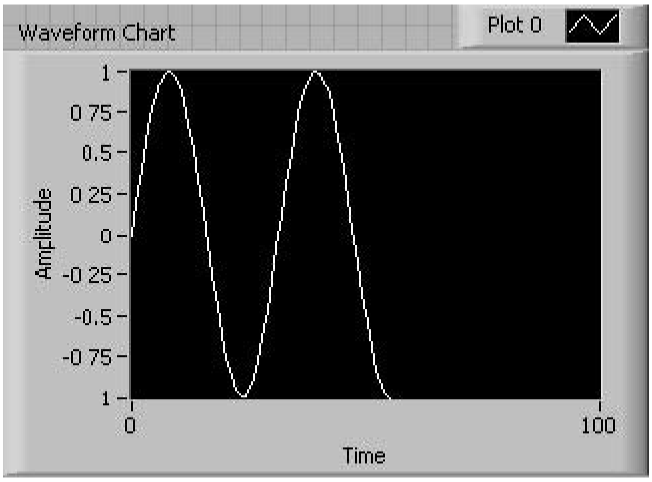


Figure 7.10 Accumulating Values for the Waveform Chart

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.

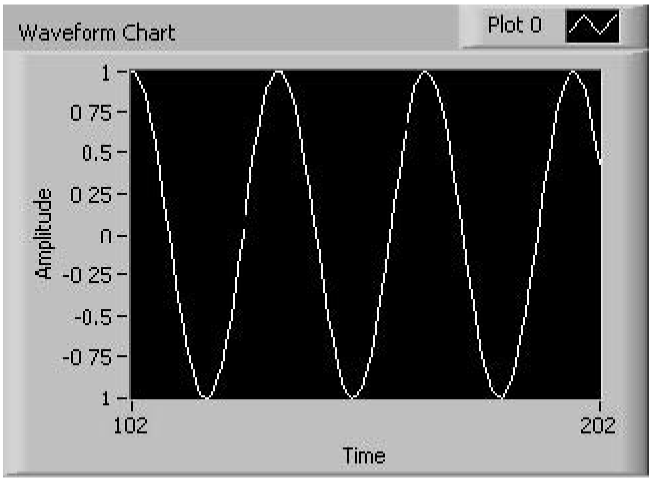


Figure 7.11 Scrolling X-Axis

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

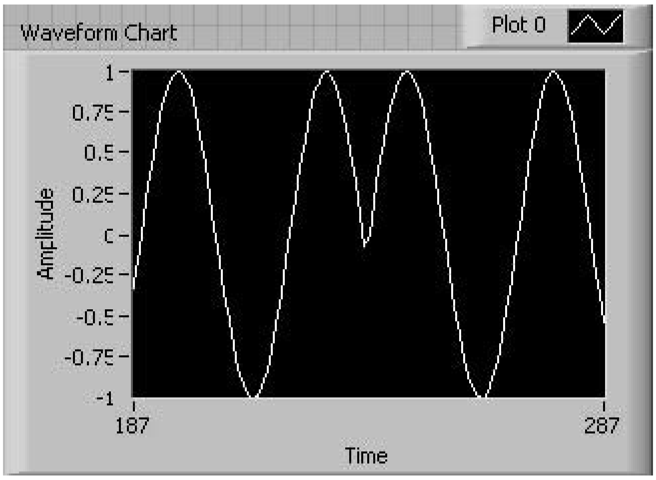


Figure 7.12 Graph History Retained Between Runs

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 Figure 7.14).

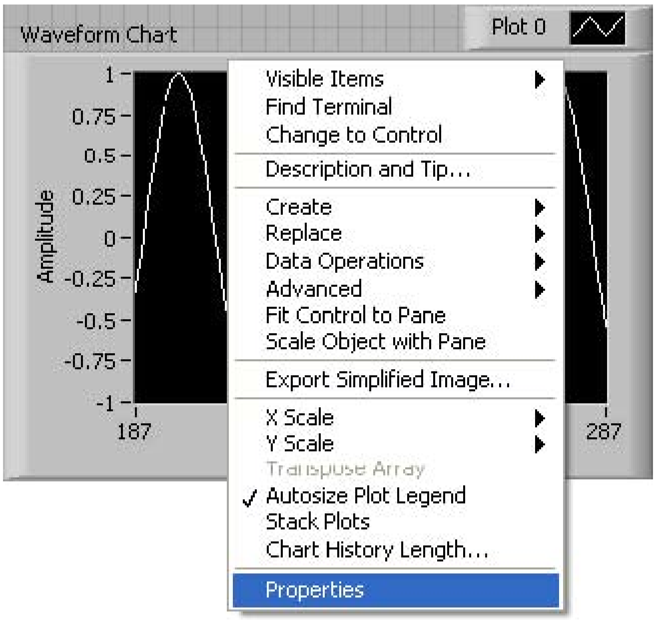


Figure 7.13 Waveform Chart Pop-Up Menu

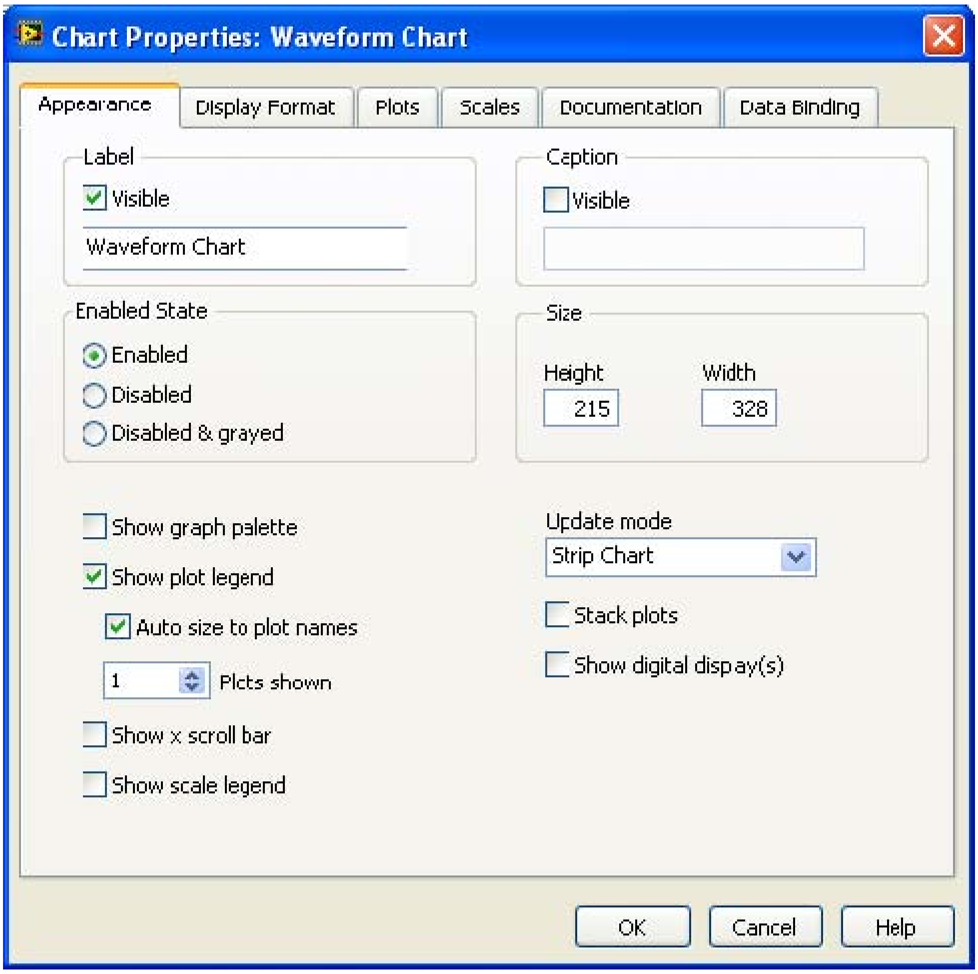


Figure 7.14 Waveform Chart Options Dialog Box

7.2 Waveform Graph

The **Waveform Graph** allows numeric arrays to be displayed graphically in the Front Panel 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:

$$y_i = sin\,(0.2xi)$$

Figure 7.15

for i = 0,1,2, ... , 99.

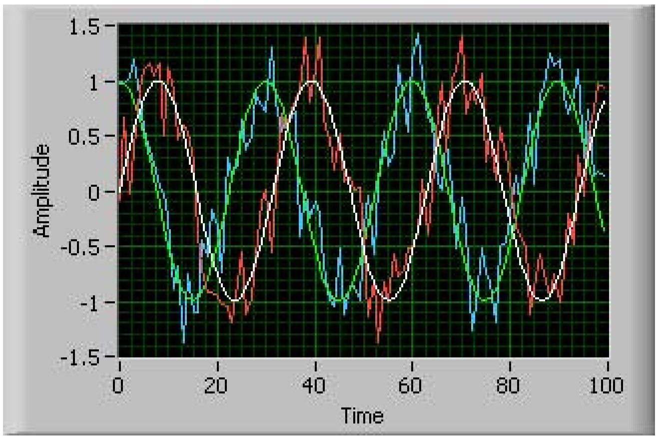


Figure 7.16 Waveform Graph

7.2.1 Single Plot

Start by building the following program shown in Figure For Loop Sine Wave.

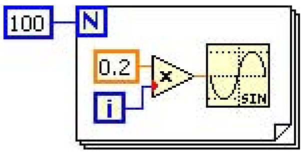


Figure 7.17 For Loop Sine Wave

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

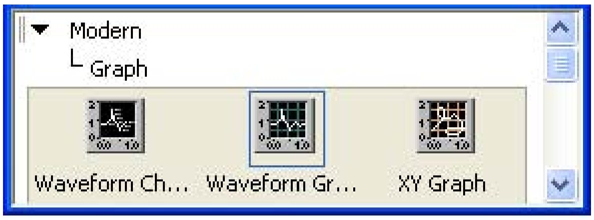


Figure 7.18 Select Waveform Graph

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

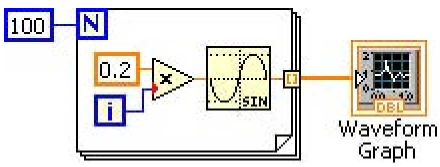


Figure 7.19 Waveform Graph Diagram

Run the program. The resulting graph is shown in Figure Sine Wave Graph.

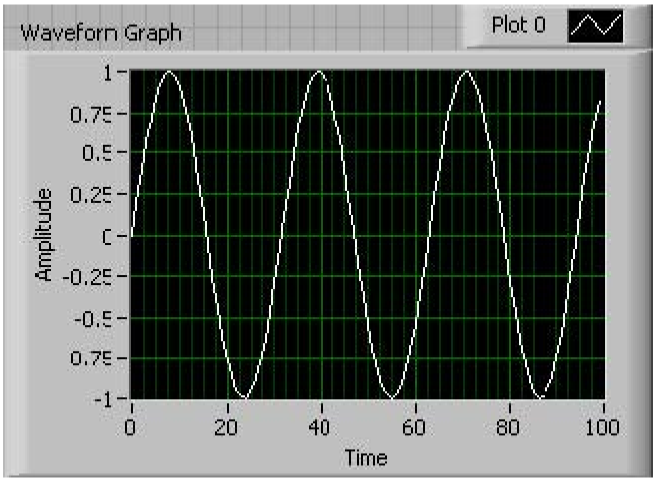


Figure 7.20 Sine Wave Graph

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 Sine and Noisy Sine Waveforms.

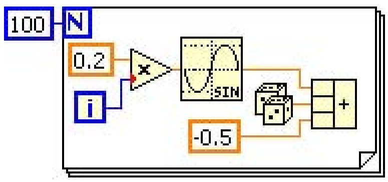


Figure 7.21 Sine and Noisy Sine Waveforms

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.

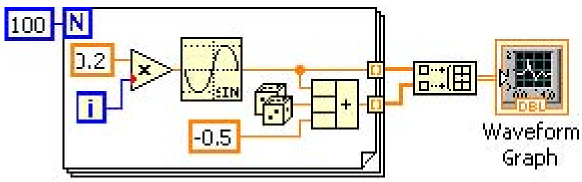


Figure 7.22 Bundle Arrays for Multiplotting

You can continue adding 1D arrays to be multiplotted into a single **Waveform Graph**. Run the program. The multiplot result is shown in Figure Multiplot.

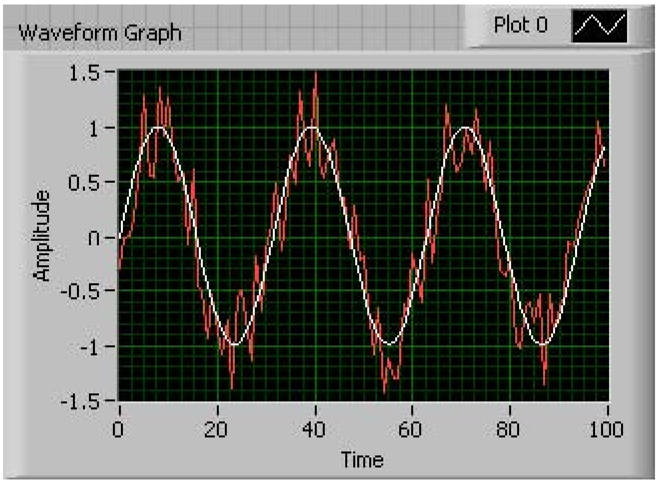


Figure 7.23 Multiplot

7.3 XY Graph

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

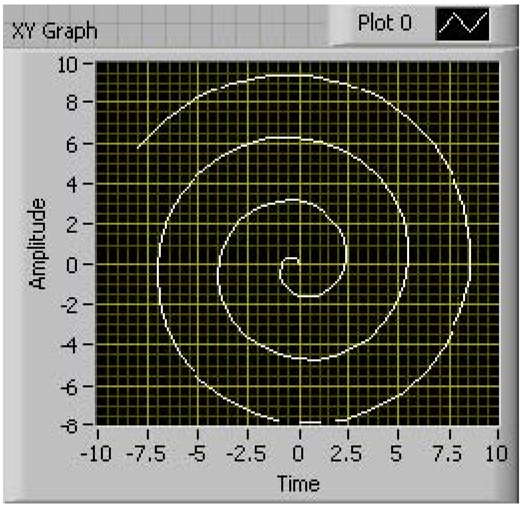


Figure 7.24 XY Graph

The example shown in Figure Spiral G Program generates the spiral shown in Figure Figure 7.24.

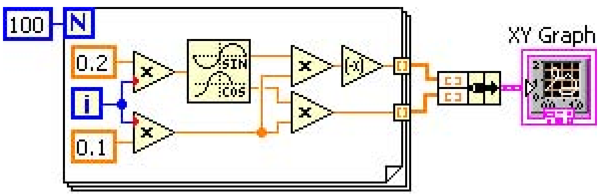


Figure 7.25 Spiral G Program

Chapter 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 Front Panel window at run time to provide seamless interaction with the G program.

Figure 8.1 Creating Interactive Programs

In the Front Panel window, from the **Functions »Modern »Numeric** select the vertical pointer slide. From the **Functions »Modern »Graph** select **Waveform Chart**.

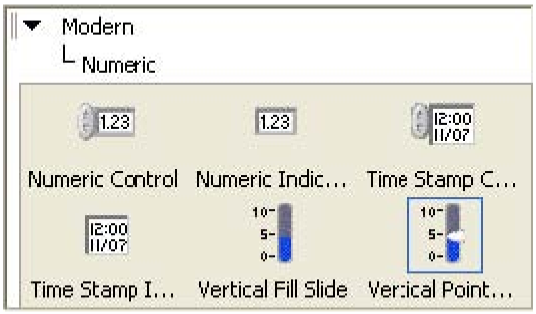


Figure 8.2 Vertical Pointer Slide and Waveform Chart

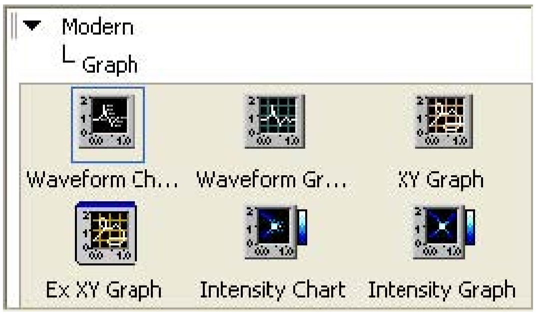


Figure 8.3 Vertical Pointer Slide and Waveform Chart

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.

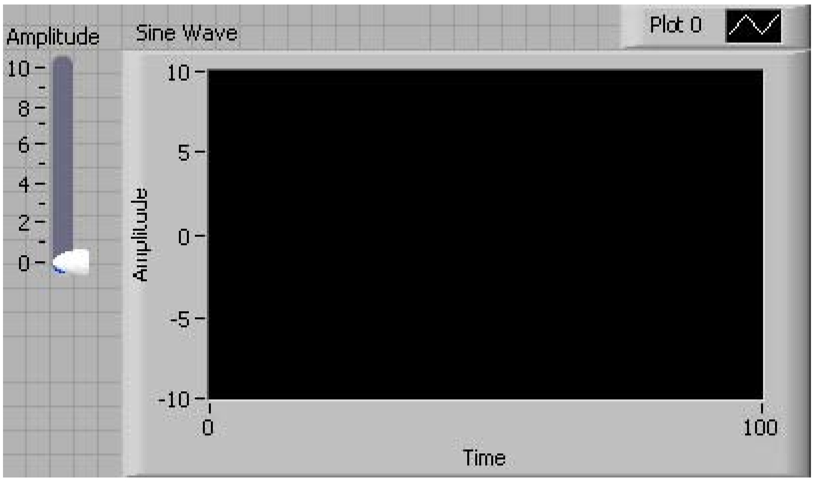


Figure 8.4 Slide & Waveform Chart in Front Panel window

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

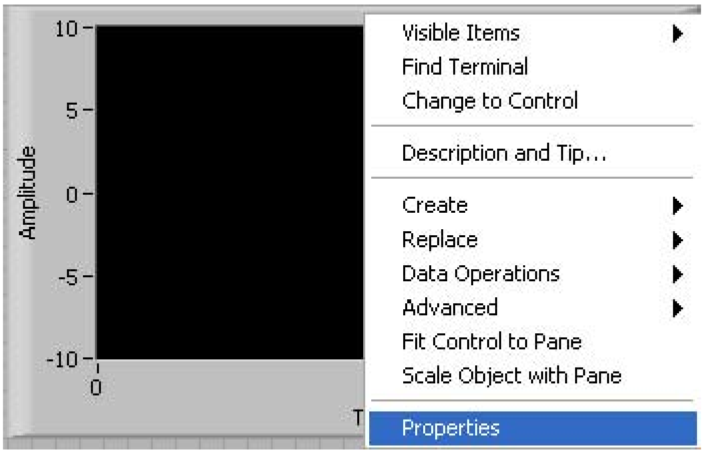


Figure 8.5 Selecting Chart Properties

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

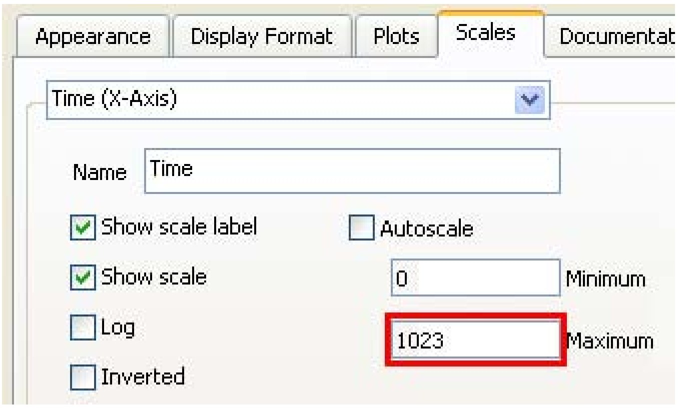


Figure 8.6 X-Axis Maximum

Click on the down arrow located to the right of **Time (XAxis)** and select **Amplitude (YAxis)**.

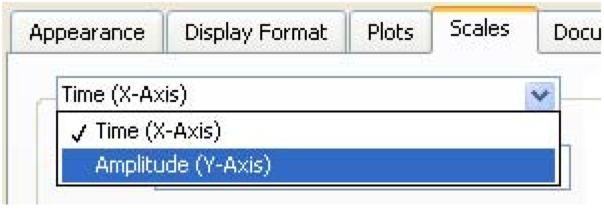


Figure 8.7 Selecting Y-Axis

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

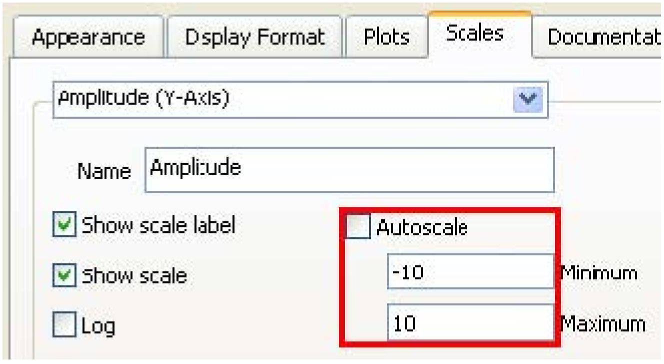


Figure 8.8 De-Selecting Autoscale

Rearrange **Amplitude** and **Sine Wave** terminals and finish the program as shown in Figure Figure 8.9. Scroll the mouse pointer over the **Loop Control**...

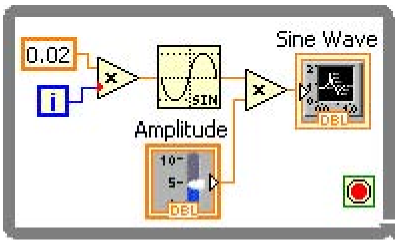


Figure 8.9 Interactive Sine Wave Diagram

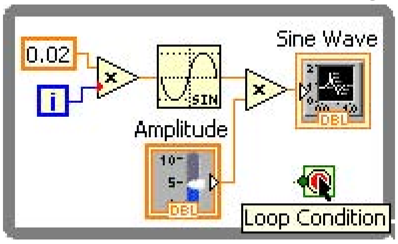


Figure 8.10 Loop Condition

And right click on the **Loop Control** and from the pop-up menu select **Create Control**. A **stop** terminal is created.

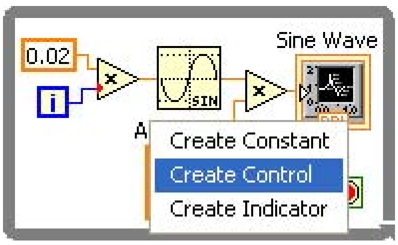


Figure 8.11 Create Control

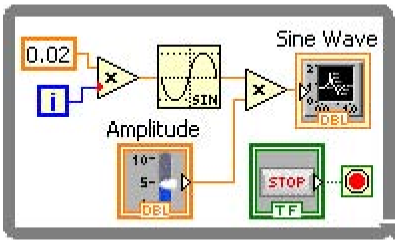


Figure 8.12 Interactive G Program

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

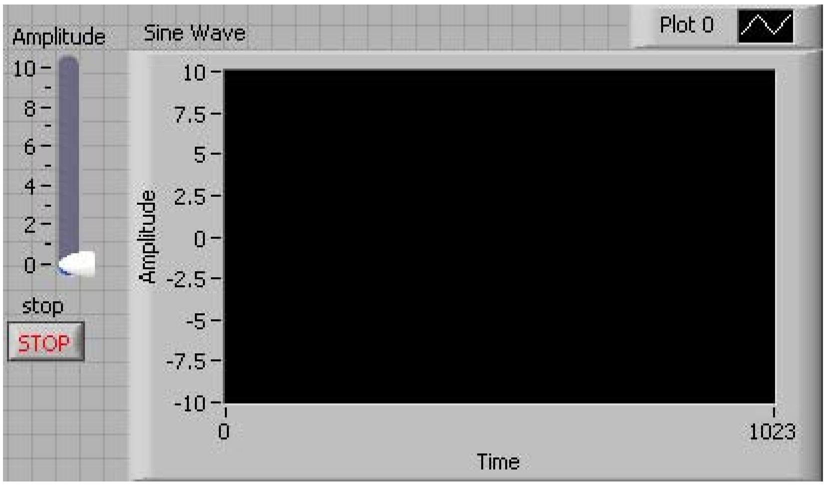


Figure 8.13 Interactive Program

Run the G program.

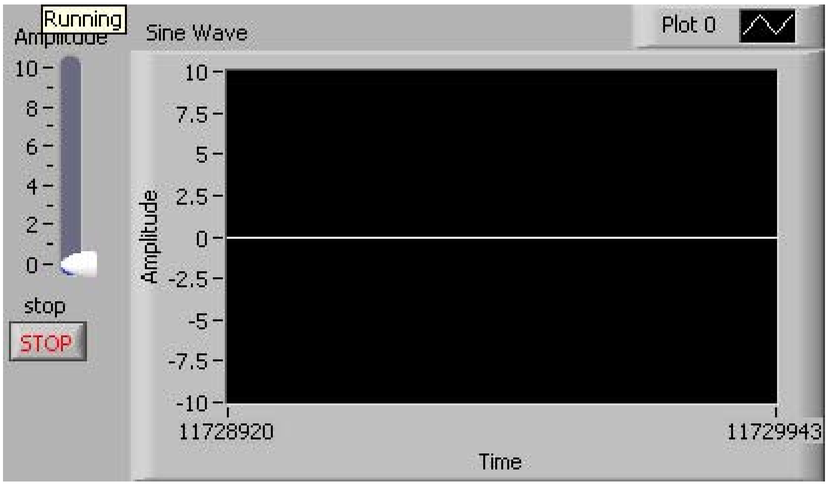


Figure 8.14 Interactive Program

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

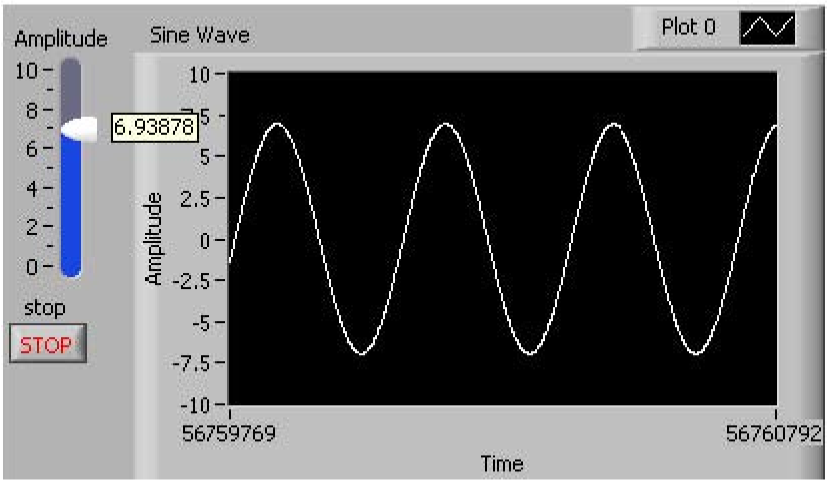


Figure 8.15 Interactive Program

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

Interactive Program

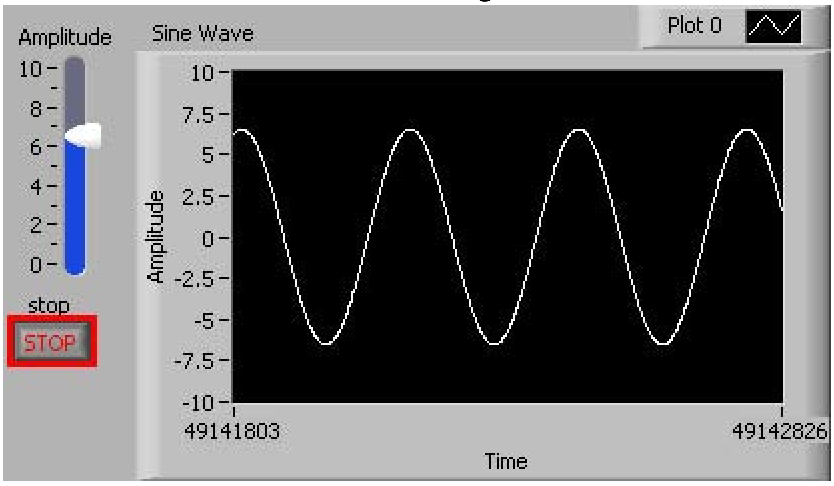


Figure 8.16

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

Chapter 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.

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

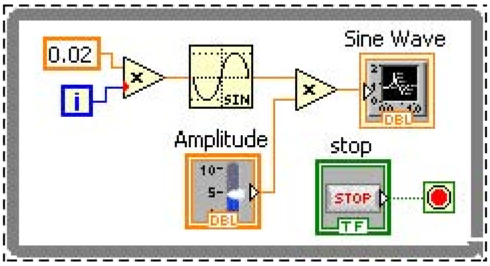


Figure 9.1 Select Diagram for Parallel Programming

From the menu select **Edit »Copy**.

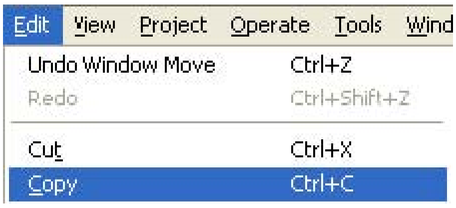


Figure 9.2 Copy Selected Diagram

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

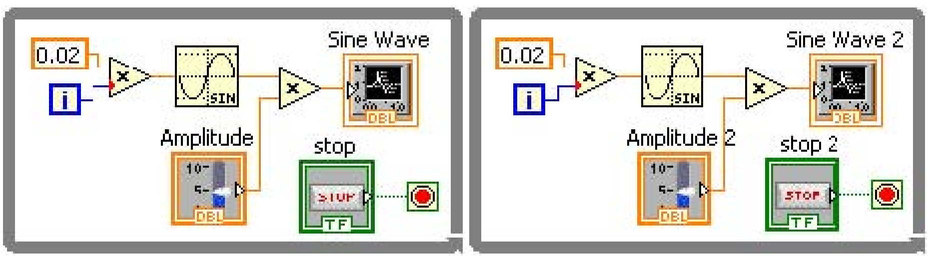


Figure 9.3 Paste Diagram

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

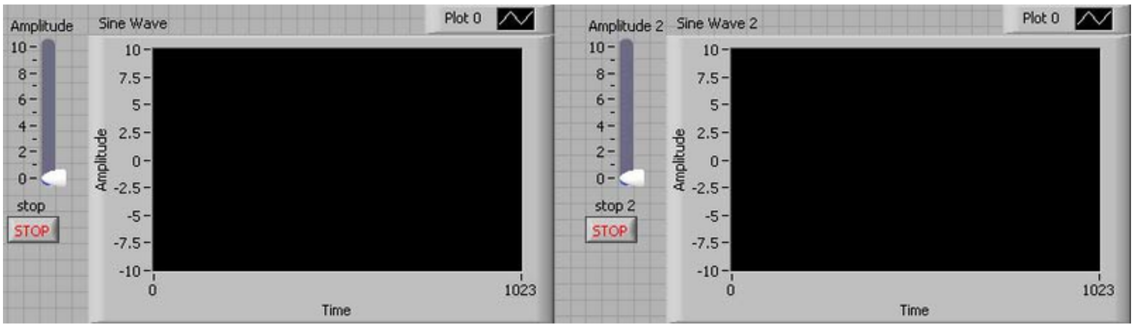


Figure 9.4 Parallel G Program

You have just completed your first parallel interactive program using G. Save the program, run it and interact with it.

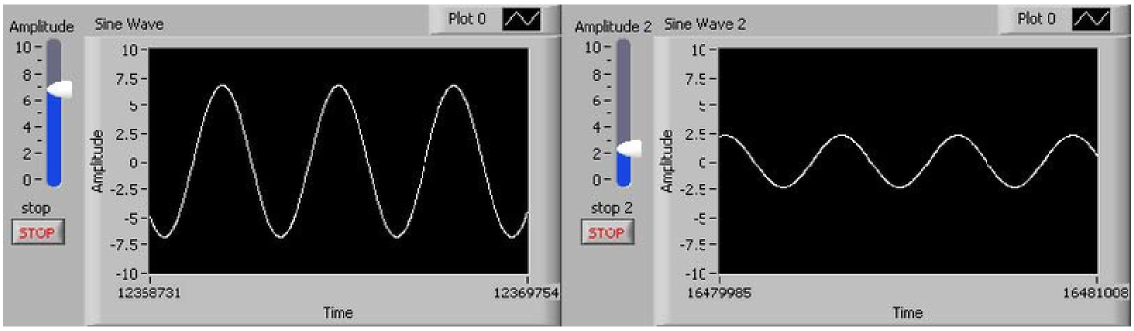


Figure 9.5 Parallel Interactive G Program

To end this program click on the **stop** and **stop 2** terminals.

Chapter 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.

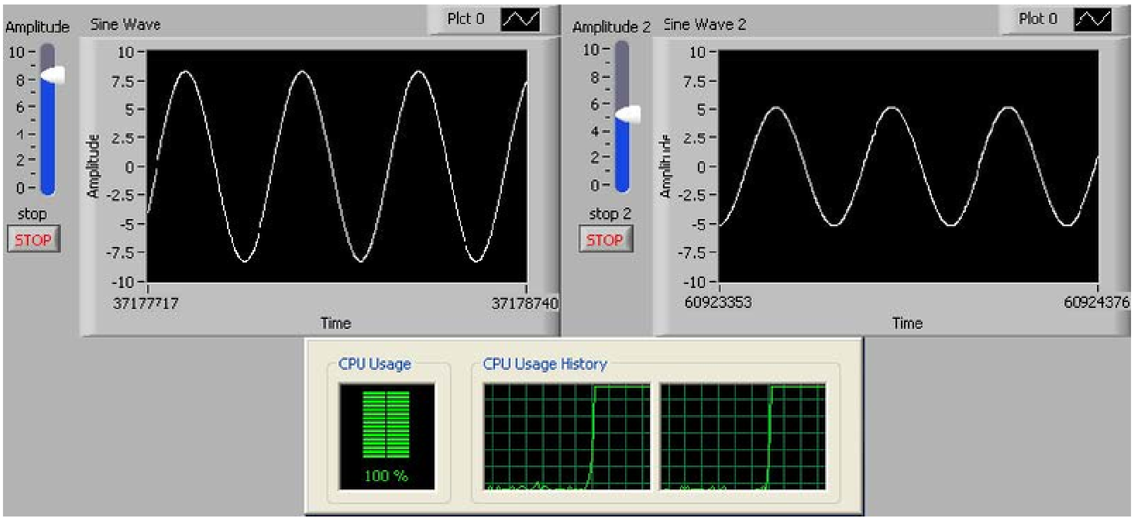


Figure 10.1 Interactive Multicore G Program

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 Data Parallelism shows a G program with 8 sequential frames to demonstrate the performance improvement via data parallelism.

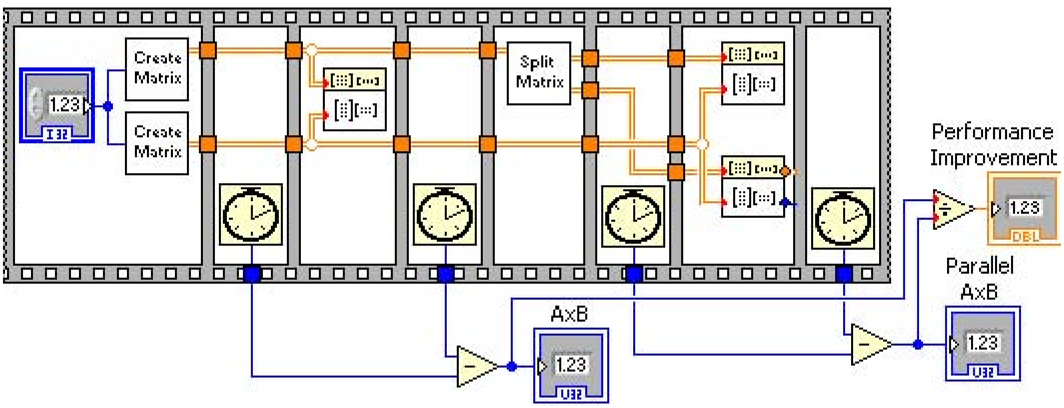


Figure 10.2 Data Parallelism

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 Creating a Square Matrix.

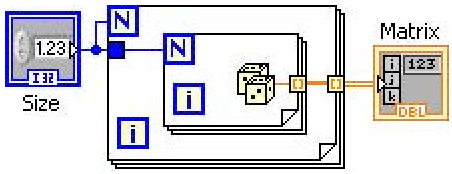


Figure 10.3 Creating a Square Matrix

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 Split Matrix into Top & Bottom.

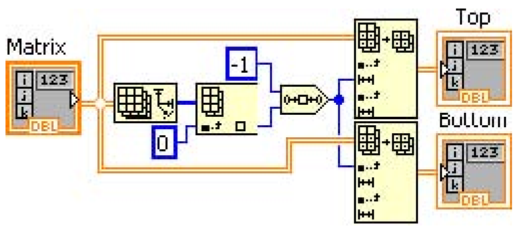


Figure 10.4 Split Matrix into Top & Bottom

Sequence Frame	Operation Description
First Frame	Generates two square matrices initialized with random numbers
Second Frame	Records start time for single core matrix multiply
Thrid 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 multicore computer.

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

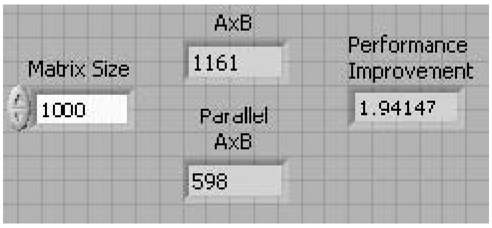


Figure 10.5 Data Parallelism Performance Improvement

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 -demodulation example where a noisy signal is modulated transmitted and demodulated. A typical diagram is shown in Figure Sequential Tasks.

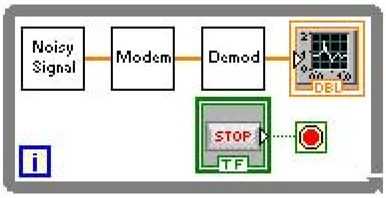


Figure 10.6 Sequential Tasks

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

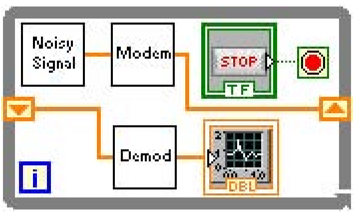


Figure 10.7 Pipelined Tasks

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

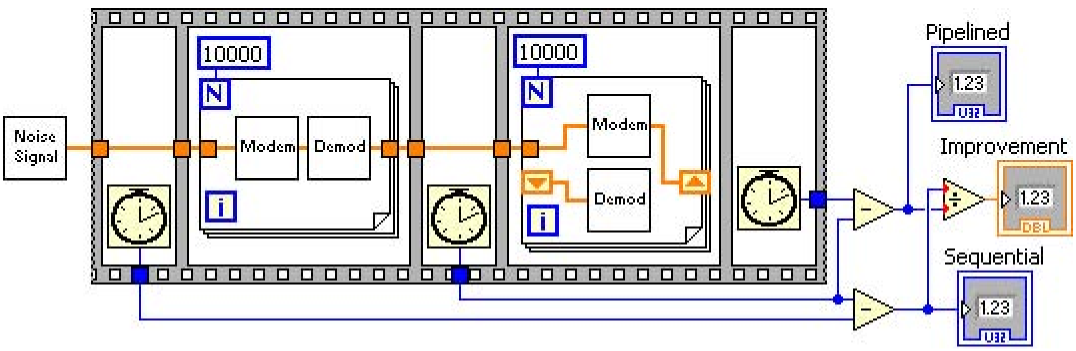


Figure 10.8 Task Pipelining Program Example

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

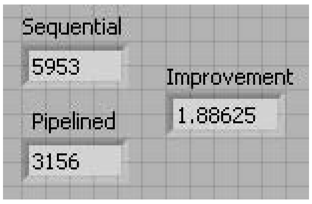


Figure 10.9 Pipelining Performance Improvement

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 Feedback Node).

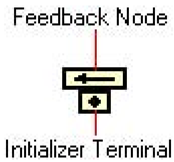


Figure 10.10 Feedback Node

To add a **Feedback Node**, right click on the Block Diagram 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 10.11 Feedback Node Direction

The diagram shown in Figure Pipelining with Feedback Node is programmatically identical to the diagram in Figure Pipelined Tasks.

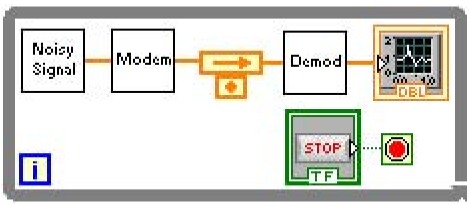


Figure 10.12 Pipelining with Feedback Node

Similarly, the diagram in Figure Pipelining Tasks with Feedback Nodes is programmatically identical to that in Figure Task Pipelining Program Example.

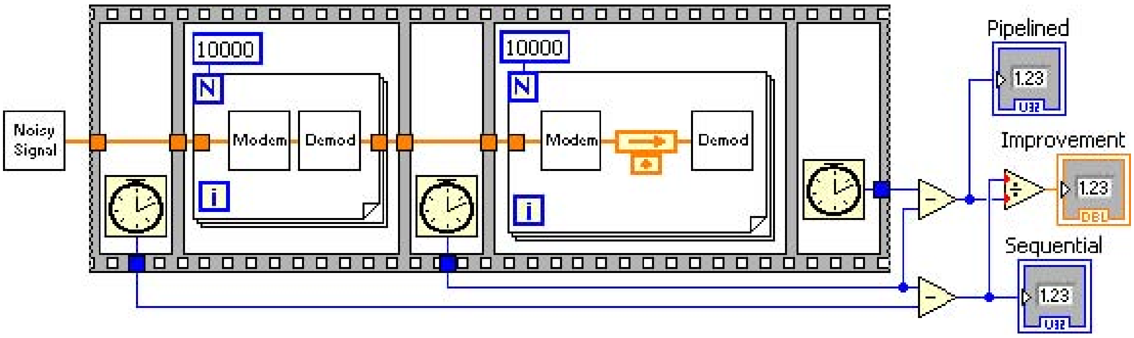


Figure 10.13 Pipelining Tasks with Feedback Nodes

Chapter 11 Input and Output

11.1 Writing to File

Consider the function in Figure Figure 11.1 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.

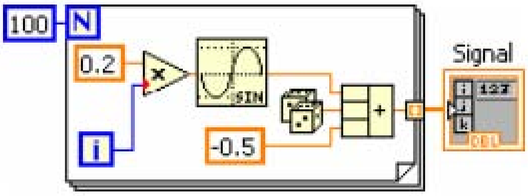


Figure 11.1 Noisy Signal Function

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.

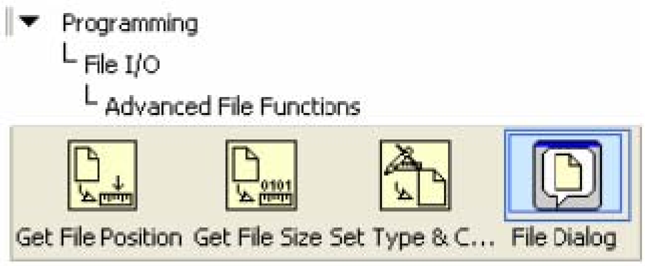


Figure 11.2 File Dialog

The **Configure File Dialog dialog** box automatically appears to configure the function. Accept the default configuration shown in Figure Figure 11.3 to create a single file by clicking the **OK** button.

Figure 11.3 Configure File Dialog

The resulting diagram after closing the configuration dialog box is shown in . 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 11.4 G File Dialog

Figure 11.5 View As Icon

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

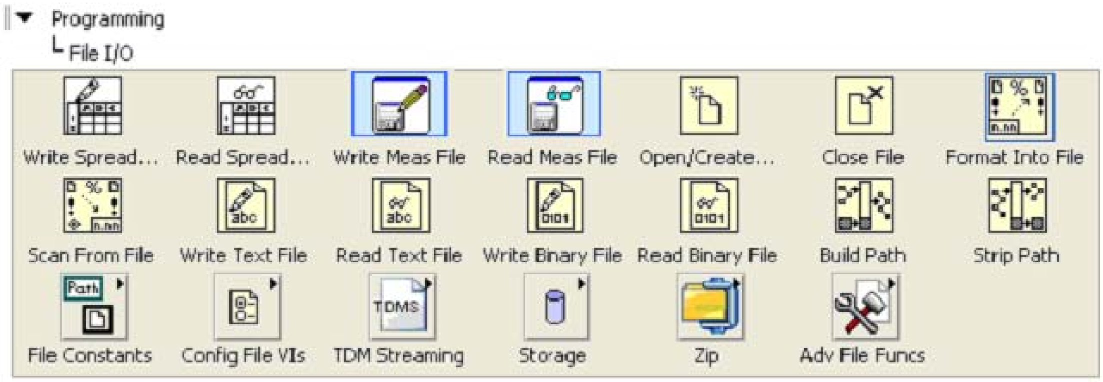


Figure 11.6 File Input and Output Operators

Arrange the File I/O operations as shown in Figure Figure 11.7.

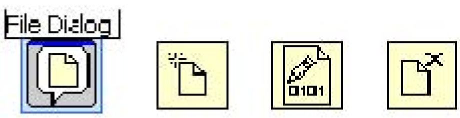


Figure 11.7 Open, Write and Close File Diagram

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

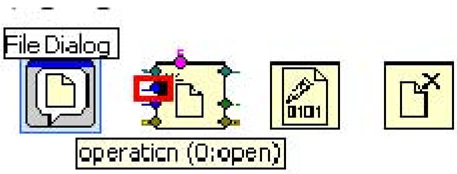


Figure 11.8 File Create Operation

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

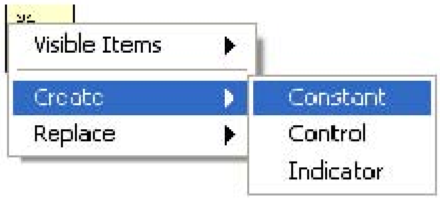


Figure 11.9 Create Operation Constant

Arrange the diagram to look as in Figure Figure 11.10.

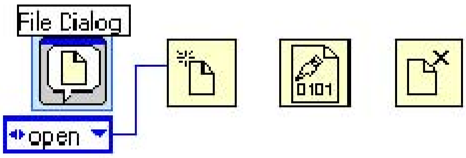


Figure 11.10 Operation Constant

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

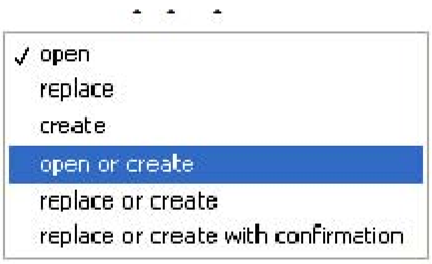


Figure 11.11 Open or Create File Operation

The resulting updated **operation** constant value is shown in Figure Figure 11.12.

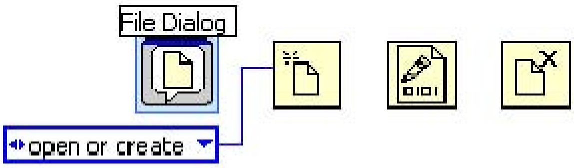


Figure 11.12 Create File to Write

Repeat the process to create a constant for the**access (0:read/write)** terminal (highlighted in Figure Figure 11.13).

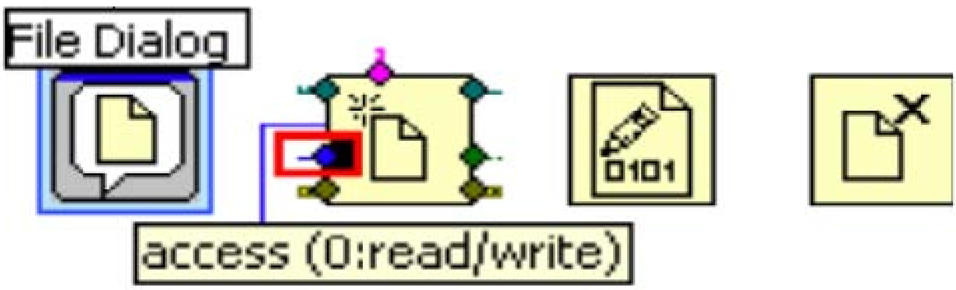


Figure 11.13 File Access Mode

Set the constant to **write-only**. Re-arrange the block diagram to look like the diagram shown in Figure Figure 11.14. At this point, the file is set to create a new file for writing.

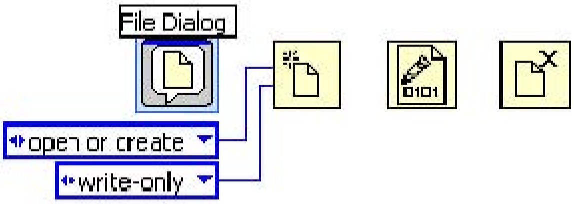


Figure 11.14 Write Only Mode

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

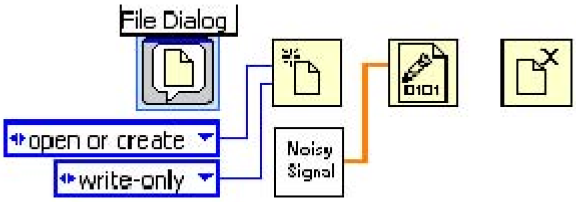


Figure 11.15 Writing Binary Data

Complete the diagram by connecting the **Open, Write** and **Close** file operations as shown in Figure Figure 11.16.

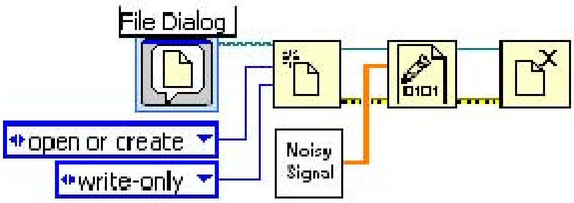


Figure 11.16 Writing to File G Program

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

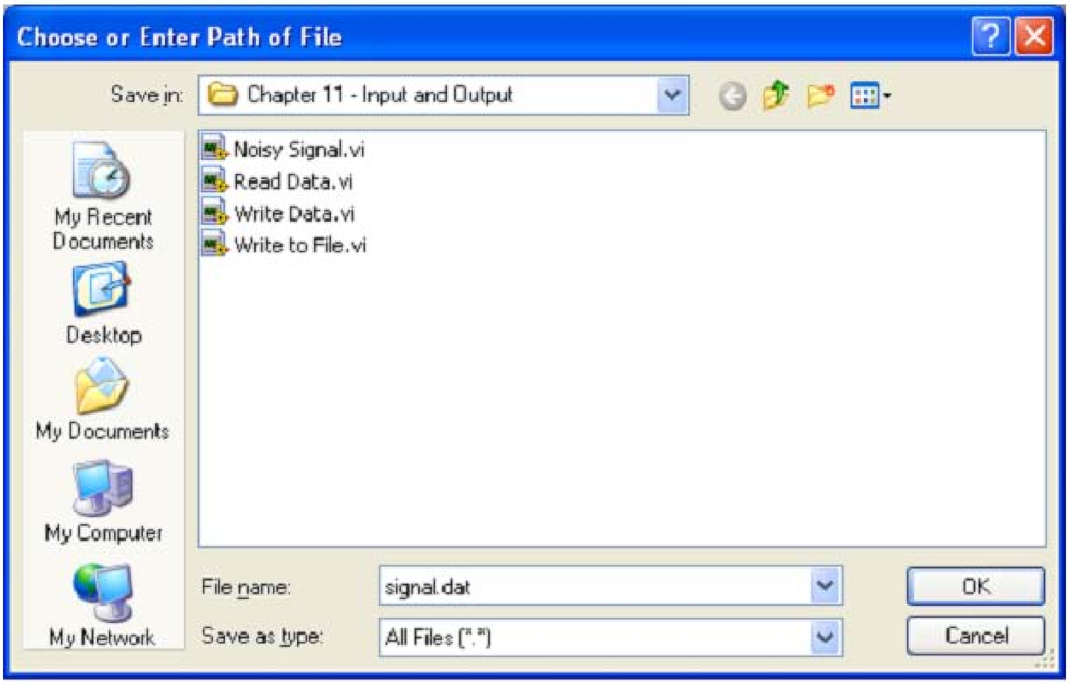


Figure 11.17 Create File Dialog Box

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

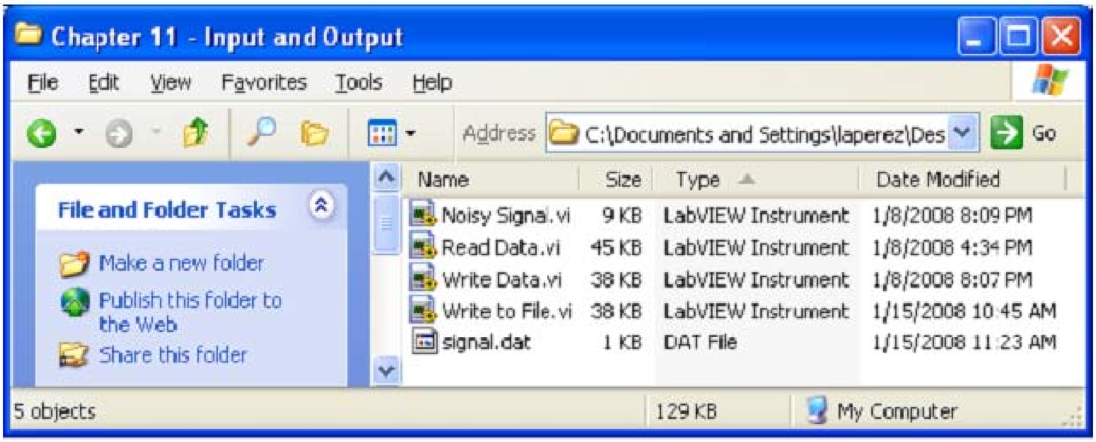


Figure 11.18 Data File signal.dat

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.

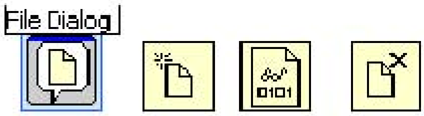


Figure 11.19 Operators to Read Files

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 Figure 11.20).

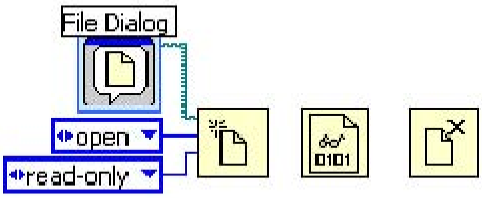


Figure 11.20 Set to Open and Read-Only

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 Figure 11.21 .

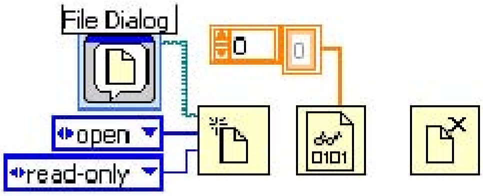


Figure 11.21 Data Type to Read

In the Front Panel window, drop a **Waveform Graph**.

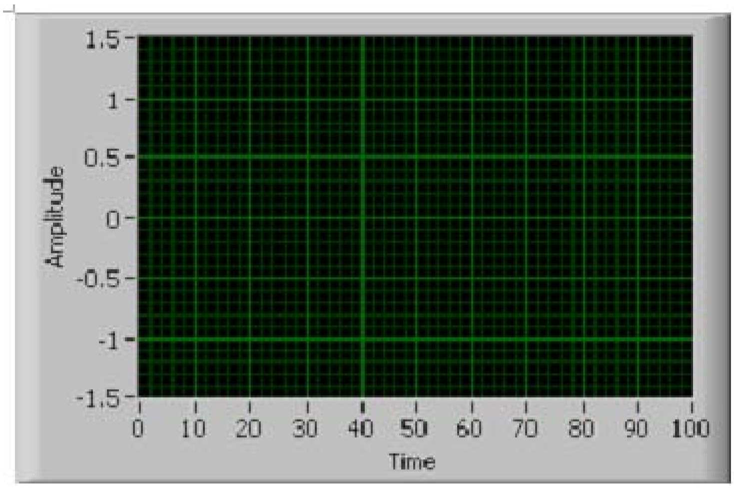


Figure 11.22 Graph for Data to be Read

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 Figure 11.23 .

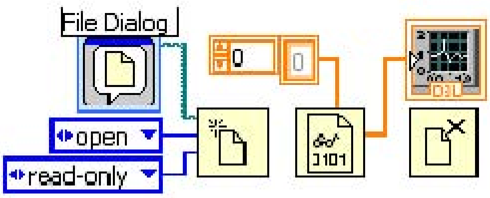


Figure 11.23 Data to be Read

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 Figure 11.24 .

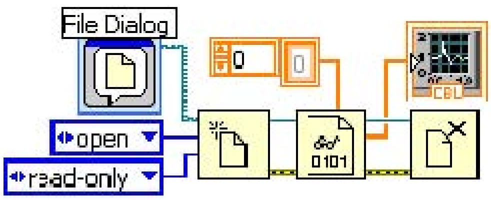


Figure 11.24 Read Binary Data G Program

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

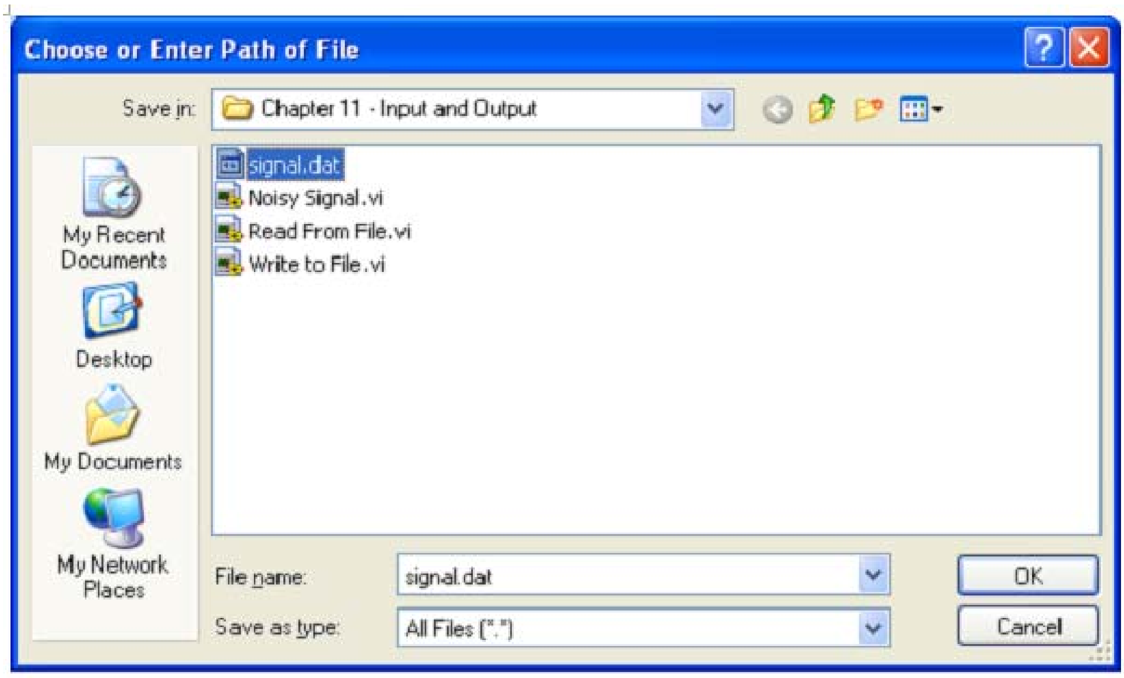


Figure 11.25 Select Binary File to Read From

The binary data in **signal.dat** is read and plotted in a **Waveform Graph**. The result is shown in Figure Figure 11.26.

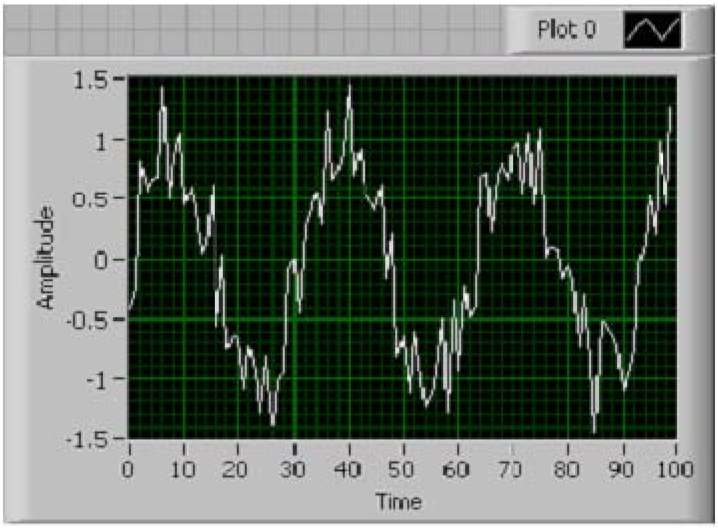


Figure 11.26 Read Data Graphed