A Test Engineer's Evaluation of Graphical Programming

By Steve Mackin, Endgate Corporation

Steve Mackin is currently a senior project engineer with Endgate Corporation in Sunnyvale, California. Steve was with Allied Signal Aerospace (formerly Garrett) for 13 years as an aerodynamic engineer responsible for the aerodynamic design and test of fans and compressors for propulsive jet engines. He holds a B.S. in Mechanical Engineering from Iowa State.

Steve is a member of the HP VEE electronic users group and can be reached at smackin@endgate.com. Visit the Endgate Corporation home page at http://www.endgate.com.

Introduction

The development of graphical programming has dramatically changed the way that test and measurement software is written today. Graphical programming has made it possible to create reliable software faster than ever before and two major suppliers have emerged as leaders in graphical programming software. This article will seek to compare traditional programming to graphical, explore basic programming elements of HP VEE and National Instruments LabVIEW, explain their differences and, through an example program, demonstrate their differences. This article will compare basic programming elements only, not advanced libraries or instrument control drivers.

Traditional Programming vs. Graphical Programming

Traditional Programming

Traditional programming languages are text based. Lines of code are created utilizing keywords and syntax rules.

The process for creating a program is shown in the figure at the right (figure 1).

The process outlined above is very iterative. Some of the new development environments help programmers by checking some aspects of syntax during editing, providing buttons to compile, link and execute the code, and positioning the cursor of the editor at the location of compile and link errors. While this is very helpful it is still too easy to create code that compiles, links, and executes but does not produce the desired result.

Programming with traditional languages is difficult because the programmer has to manage the execution and data flow using keywords, syntax, variable names, and data structures. Variable names are assigned by the programmer and need to be typed into the code in numerous places. It is relatively easy to mistype the variable name, use the wrong variable name, or use the right name in the wrong place.

The debug environment was created to allow programmers to view the execution and data flow. The debug environment can show what is wrong but it cannot eliminate the root cause of the problem. While creating a program the

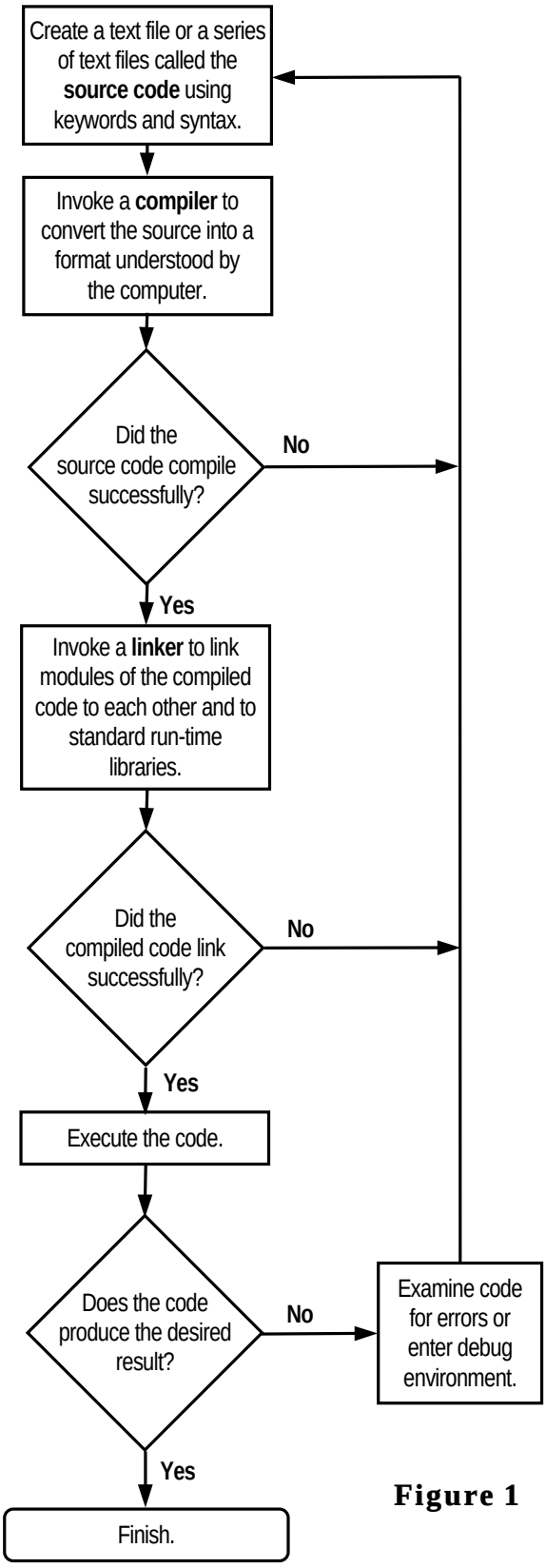


Figure 1

programmer encrypts the execution and data flow. To derive the execution flow by just looking at source code can be very difficult even for the author. That is why it is called code!

Graphical Programming

Graphical programming is vastly different from traditional programming because the data flow and execution flow are managed graphically. A diagram is created using objects and lines. The lines primarily represent data and the objects create, analyze, or display the data. Execution flow typically follows the data flow. When execution flow cannot follow the data flow the execution flow is still managed graphically. A graphical program is easy for the author or anyone else to decipher and easy to support.

The process for creating a graphical program is shown in the figure to the right (figure 2).

As you can see the program creation process is relatively simple. Because objects are selected from pull down menus it is possible to leverage reliable, consistent (execution, user interface, and display), and useful code. Most of the objects utilized would require hundreds or even thousands of lines of traditional code. Execution and data flow are clearly represented in the diagram created that represents the program. There are fewer opportunities for human error and problems are easier to locate. The finished code is easily interpreted by both the author and other programmers.

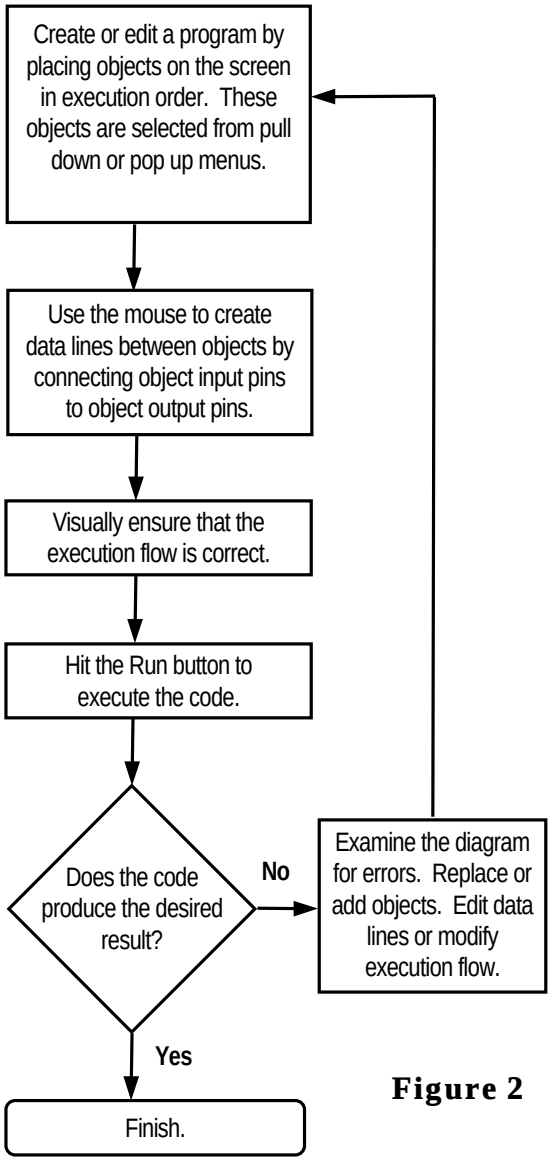


Figure 2

LabVIEW vs. HP VEE

LabVIEW from National Instruments and VEE; from Hewlett Packard are the two major graphical programming languages in use today. While they perform similar functions they use significantly different methods. Both have significant strong points as well as drawbacks.

Basic Components

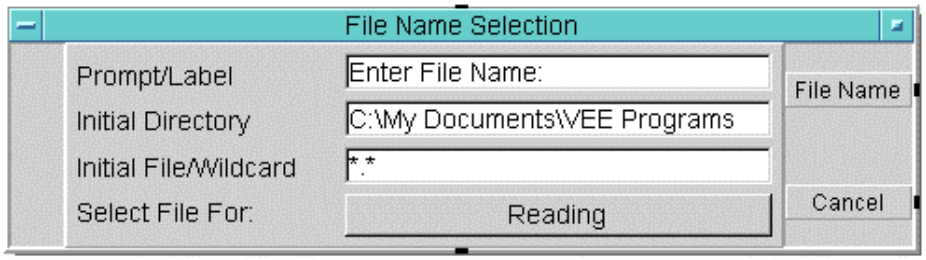
Primitive objects are objects, which create, analyze, or display data. These objects are selected from pull down or pop up menus. They can be assembled and linked with lines creating graphical programs, which can be made into functions. Functions in LabVIEW are called VI's (vee-eyes not sixes) and functions in HP VEE are called User Functions. A pin is a point on an object which indicates that a connection may be made at that location. Most objects contain at least one pin and there are significant differences in the way pins are handled in LabVIEW and HP VEE. Lines represent data and can either be inputs to the object or outputs from the object. Lines can also be used to sequence the execution of objects in both LabVIEW and HP VEE. The color and style of lines indicate the type of data represented by the lines. LabVIEW and HP VEE have many data types and styles in common but each also employs unique data types and styles. The utilization of data types and styles in graphical programming is a very complex subject and beyond the scope of this article but some basic differences will be described.

Object Comparison

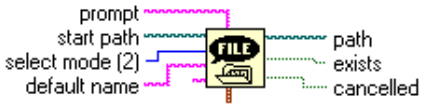
1. LabVIEW objects are very graphical compared to HP VEE objects. They use shapes, colors, and text to convey lots of information. All LabVIEW objects have labels and the programmer can type in a meaningful label. HP VEE objects are always rectangular and they also have labels which the programmer can use to describe what the object does. A HP VEE programmer can color objects and utilize bitmaps if he or she wants to alter the appearance of objects.

2. All HP VEE objects can be resized with no impact on the execution and data flow. Most LabVIEW objects cannot be resized and the ones that can be resized can only be resized by creating more pins, which changes the program. All LabVIEW VI's are the same size so there is a limit to the number of pins, which will fit on an object. This may seem like a purely cosmetic concern but re-sizing objects can really enhance the readability of a graphical program (See the example program comparison at the end of this article).

3. HP VEE objects have two views (open and closed). Most HP VEE objects will allow the programmer to type data into the object to control the action of that object. Most LabVIEW objects require that data be wired into the object from other objects. The example below illustrates how information can be typed into the open view of a HP VEE object and the same information must be wired into a LabVIEW object.



VEE File Dialog Object



LabVIEW File Dialog Object

Figure 3

Pin Comparison

1. HP VEE objects have sequence pins: One pin on the top of the object and one on the bottom. These pins are used to sequence execution flow and are typically used only if the execution flow of the program cannot follow the data flow or if there is no data flow available for sequencing. LabVIEW objects do not have sequence pins. Objects are sequenced by data flow and if data flow cannot be used a multi-layer window scheme is required.

2. All HP VEE object input pins are on the left side of the object and all output pins are on the right side of the object. This makes HP VEE code easy to interpret. Input and output pins on LabVIEW objects can be anywhere (see file dialog object above).

3. LabVIEW pins are color-coded with the data type required and the pins do not typically convert the data type. HP VEE objects are not color-coded and data type conversion is typically automatic following certain rules. HP VEE programmers rarely need to be concerned with data typing because the rules HP VEE uses for conversion are very convenient. However, should the need arise, the rules can be overridden.

4. HP VEE uses data input pins to control attributes of control and indicator objects. These input pins do not influence data or execution flow. To indicate this the wires connected to these pins are dashed. To control attributes of control and indicator objects in LabVIEW, attribute nodes are created using a pull-down menu on the object. These attribute nodes can be placed anywhere in the program and are often duplicated in numerous places. This makes interpreting the program difficult. On the other hand, the number of attributes a programmer can control on LabVIEW controls and indicators is much greater than the number of attributes a programmer can control on HP VEE controls and indicators.

5. There is a method in LabVIEW to create an object(s) that represents the output or input pins of control or indicator objects (Local Variables). These objects can then be placed anywhere in the program. Input to an indicator object can physically be wired in one part of a program and indirectly wired in another. Doing this creates a whole new dimension for the data to flow. Instead of flowing on wires from left to right the data may enter the object from another location. This, of course, makes interpreting the program difficult. Local variables are mostly used to extract data from a control without using a wire. Essentially the data dives into the screen to resurface where it's needed. This can be accomplished in HP VEE using global variables. Set global and get global objects are used. The name of the global variable is typed into the open view of the set global object (it can also be sent in on a line). Get global objects can then placed into the program at the desired resurfacing point.

Line Comparison

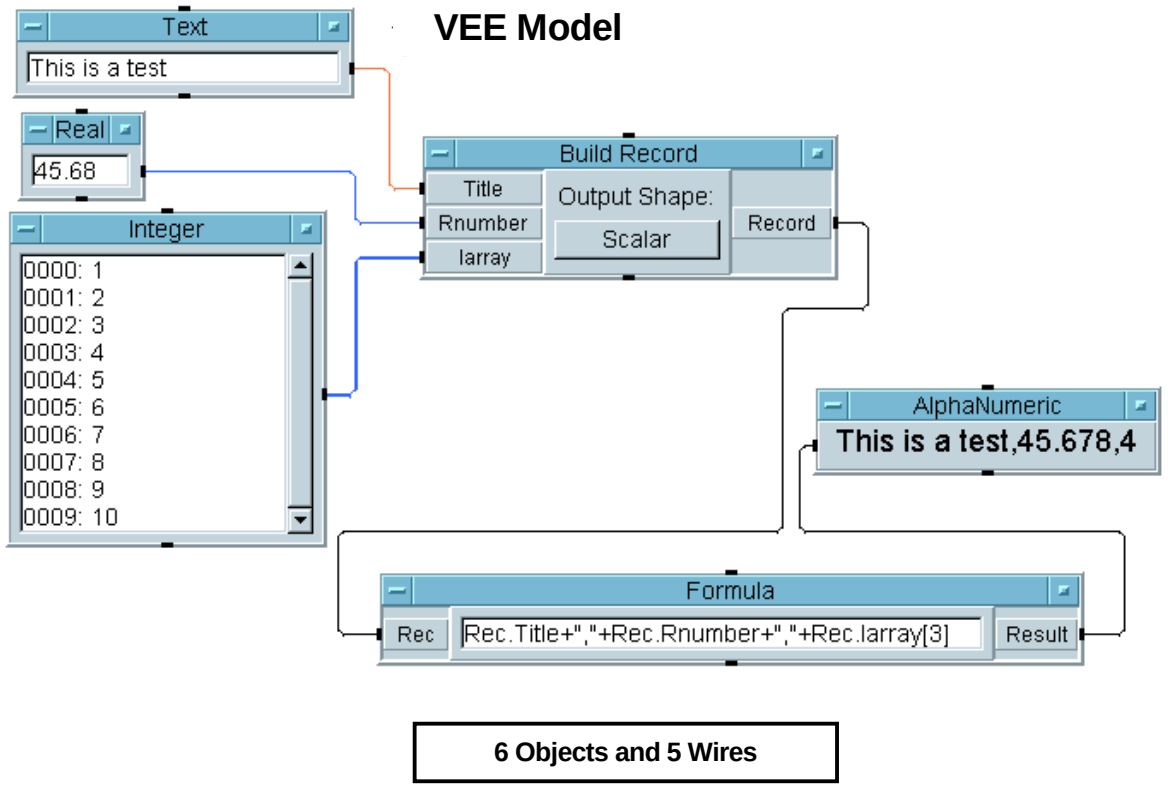
1. Both HP VEE and LabVIEW now have colored textured lines. The color represents the data type and the texture represents the shape.

2. Line routing in HP VEE is automatic both before and after lines are connected. Line routing in LabVIEW is manual before the objects is wired and automatic after the object is wired. LabVIEW's automatic line routing is virtually useless if the object is moved very far. Manual line routing in LabVIEW is a skill that needs to be learned. It is very easy to create LabVIEW programs that are difficult to interpret simply because of poor line routing.

Data Types and Shapes Comparison

1. LabVIEW data types and shapes are very similar to traditional programming language data types and shapes. HP VEE data types and shapes are more specific to the test and measurement field. For instance HP VEE has a Waveform data type which is a 1-D real array with time domain mapping information. Whether the unique data types in HP VEE are useful depends upon what type of data manipulation the program requires. If time domain type data is dominant HP VEE would definitely be easier to work with.
2. Data type conversion is one area where HP VEE really out-performs LabVIEW. Often programmers need to add strings and numbers together. In HP VEE strings and numbers are simply wired to an (a+b) formula object and the result is a string. The default number to string conversion is typically sufficient. To do the same task in LabVIEW a concatenate string object must be used and the number must be converted to a string first using a conversion object. While this seems like a minor nuisance it illustrates a fundamental difference between LabVIEW and HP VEE. LabVIEW programmers need to worry a lot more than HP VEE programmers about data type. This is especially true when creating VIs. In LabVIEW you nearly always end up specifying a data type for all the input pins even if the data type is of no significance. HP VEE programmers rarely force a data type on input pins.
3. Both LabVIEW and HP VEE have a very useful composite data type, which is created by combining other data types. A Cluster in LabVIEW is identical to a Record in HP VEE. LabVIEW and HP VEE both have tools for creating and obtaining information from Clusters/Records.

The method used in LabVIEW to extract data from arrays and clusters is very different from HP VEE. HP VEE programmers typically use a formula box and type in what they want to extract. LabVIEW programmers need to use objects specific to the type of data they are working with and the type of extraction they want to perform. The array or cluster along with constants needs to be wired to the extraction object to extract the desired data. Basically, HP VEE extractions are text based and LabVIEW extractions are graphical. See figure 4 on the following page.


VEE Model
Text
This is a test
Real
45.68
Integer
0000: 1
0001: 2
0002: 3
0003: 4
0004: 5
0005: 6
0006: 7
0007: 8
0008: 9
0009: 10
Build Record
Title
Rnumber
Iarray
Output Shape:
Scalar
Record
AlphaNumeric
This is a test,45.678,4
Formula
Rec
Rec.Title+","+Rec.Rnumber+","+Rec.Iarray[3]
Result
6 Objects and 5 Wires

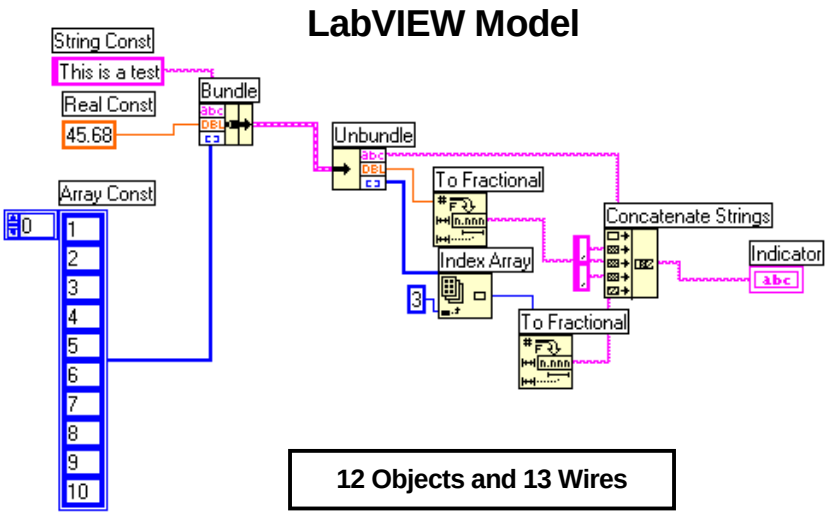

LabVIEW Model
String Const
This is a test
Real Const
45.68
Bundle
Unbundle
Array Const
0
1
2
3
4
5
6
7
8
9
10
To Fractional
Index Array
3
Concatenate Strings
Indicator
To Fractional
12 Objects and 13 Wires

Figure 4

In addition to these major differences between the two programs quite a number of more subtle differences should be noted.

Program Flow (Loops, Case Structures)
The techniques used to implement Loops and Case Structures in LabVIEW and HP VEE are very different and best described graphically.

Looping Structures
In LabVIEW, surrounding objects to be iterated on with a loop structure box creates loops. Input and output pins on the box can be set to index incoming data or collect outgoing data into an array. Shift registers may also be added to the loop to pass information from one iteration to the next. HP VEE loops are defined using loop objects and sequence pins. A collector object is used to collect outgoing data into an array. Looping structures in HP VEE are harder to identify than looping structures in LabVIEW. See figure 5.

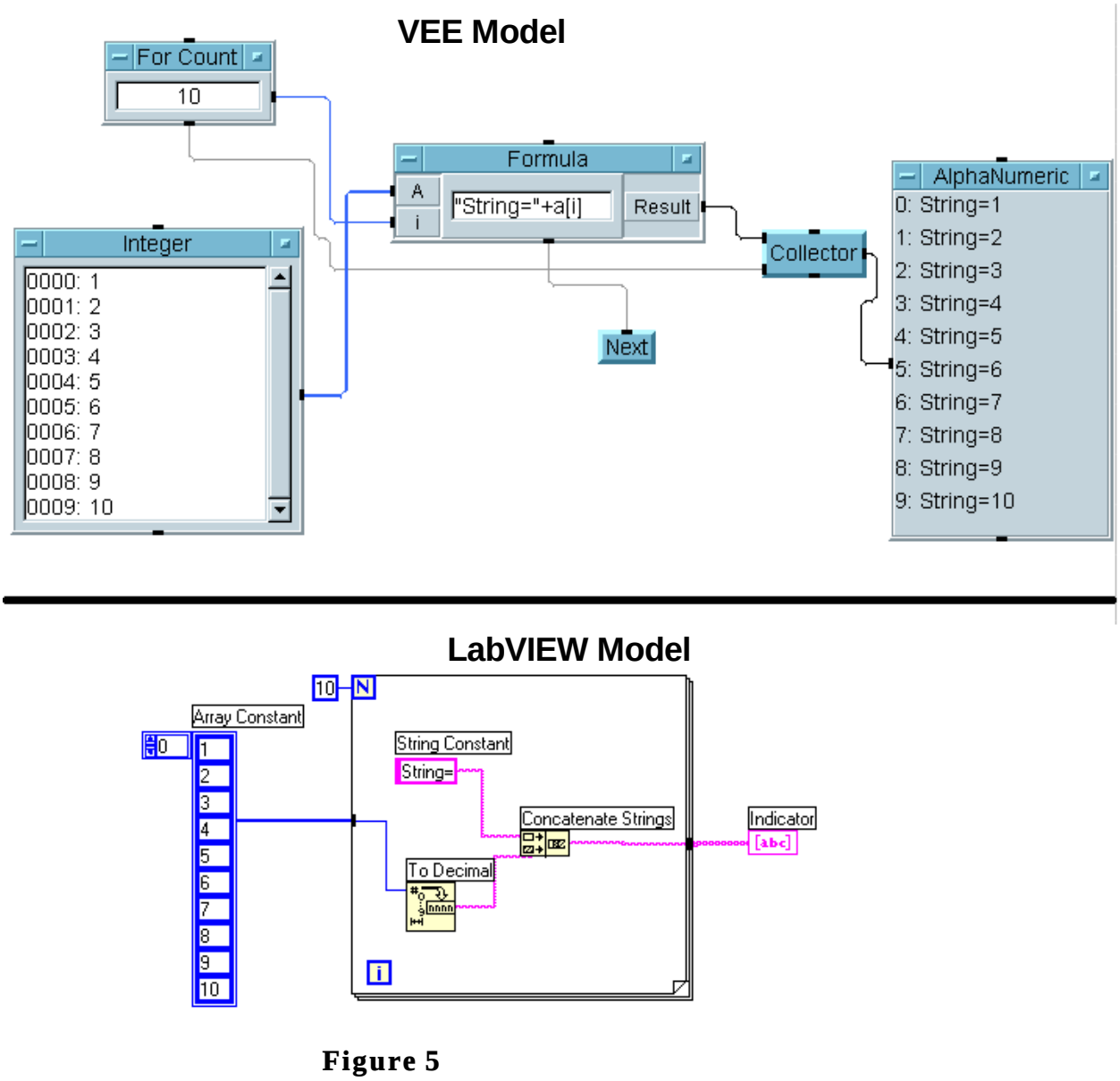
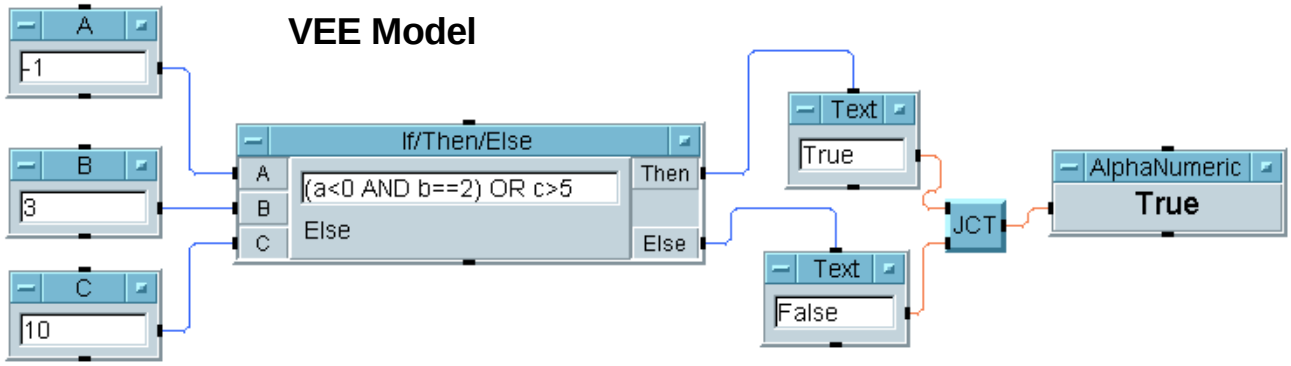


Figure 5

Case Structures

Like looping structures, Case Structures in LabVIEW are created by surrounding objects with a case structure box. Each case becomes a separate window. Case Structures in HP VEE are created using case objects and sequence pins. A major disadvantage of the LabVIEW technique is that only one case is visible at a time. Another disadvantage is that the case logic must be created graphically. See Figure 6 below.



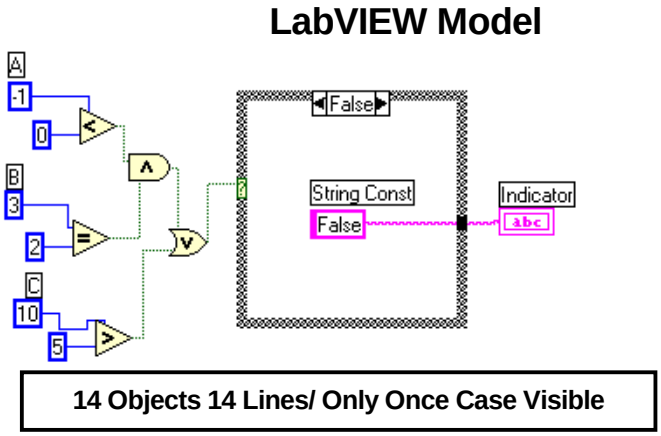


Figure 6

User Interface Panel

In both LabVIEW and HP VEE a programmer can create a user interface panel. To create a user interface panel in LabVIEW the programmer selects a control or display object from a pop up menu while the user interface panel is in focus (the top window). Once an object is selected and placed on the panel an object that represents the display or control is created on the LabVIEW diagram where it is wired into a program. To create a user interface panel in HP VEE, the programmer selects the object to be placed on the user interface and chooses an edit option to add the object to the panel. One advantage of the HP VEE method is that user interface view of controls and displays can be seen and operated from the programming window. There is no need to switch between windows to control and input or view the results of a program. This is illustrated in figure 7 below.

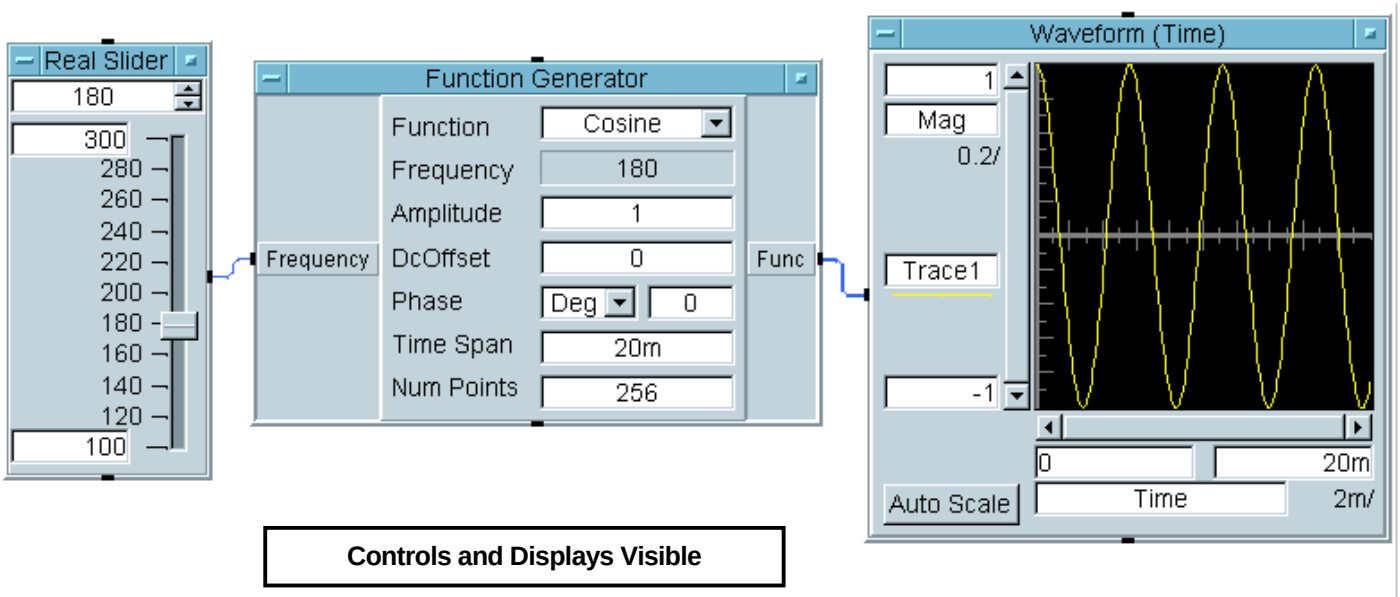


Figure 7

Subroutines (User Functions/VIs)

The main difference between HP VEE user functions and LabVIEW VIs is the technique used to call them and the control a programmer has on the location and size of a pop-up display.

Calling User Functions/VIs

To call a LabVIEW VI it needs to be placed on the diagram and wired with data (unless very advanced features are used). HP VEE User Functions can be placed on the diagram and wired with data but they may be called from a formula or sequencer object. This capability is immensely powerful and a tremendous shortcut. In figure 8 (following page) notice the way a string concatenation is typed into the function call.

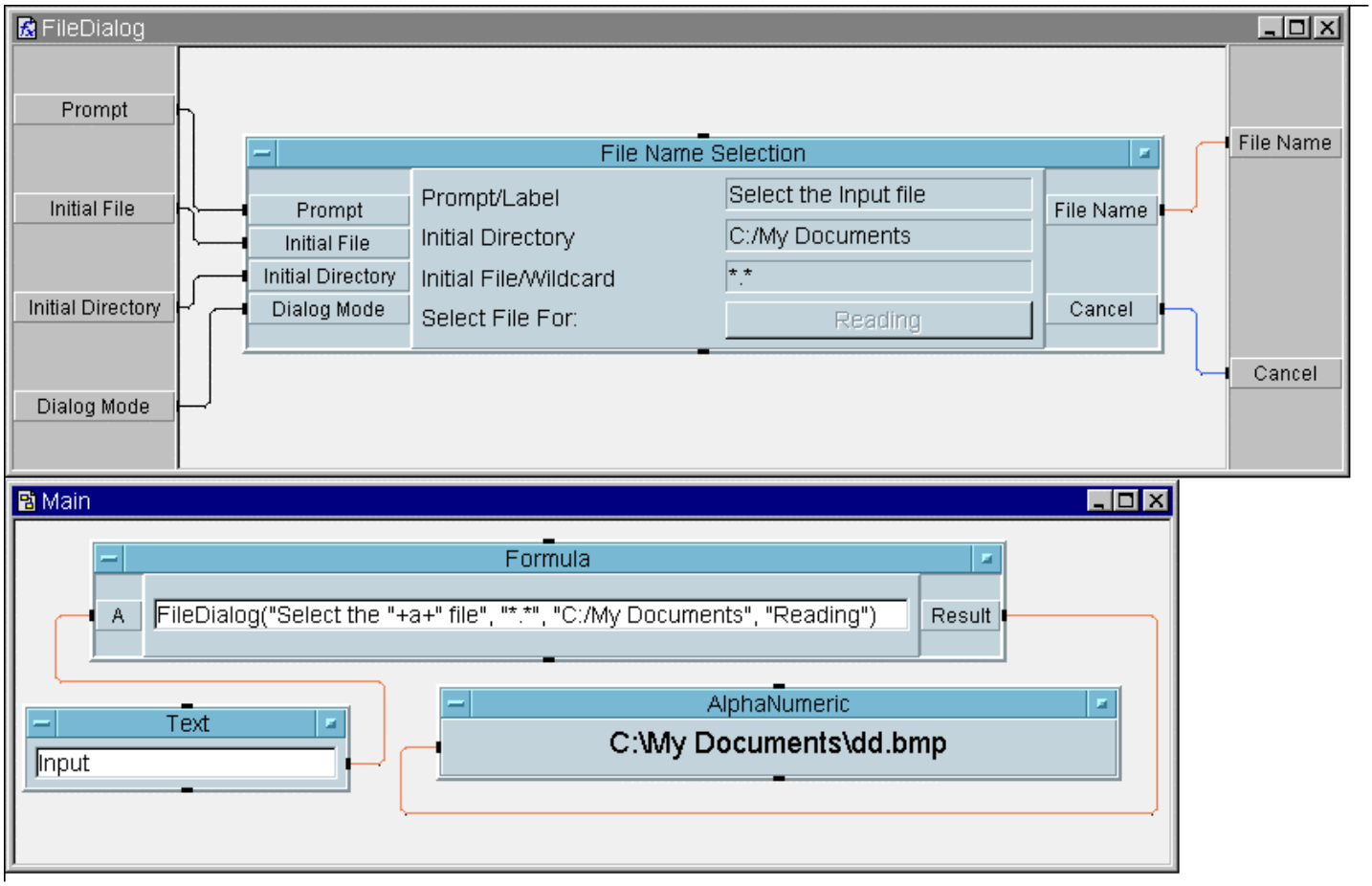


Figure 8

Pop-up Displays
Both LabVIEW and HP VEE functions can pop-up user interface displays if the programmer desires. Only HP VEE allows the programmer to control the precise location and size of the panel.

Development Environment User Interface
Differences in the development environment exist primarily in cursor modes (tools) and debug probing features.

Cursor Modes
LabVIEW programmers are responsible for selecting the correct cursor mode. LabVIEW refers to cursor mode selection as tool selection. The use of cursor tools is patterned after paint programs. There are four basic tools:

1. Operating Tool – The mode the cursor must be in to operate controls

2. Positioning Tool – The mode the cursor must be in to select, move and resize certain objects. This tool must also be used to select, move and delete lines.

3. Labeling Tool – The mode the cursor must be in to do any kind of ASCII input.

4. Wiring Tool – The mode the cursor must be in to wire objects.

Toggling between these modes is usually done using keyboard short cuts. The space bar toggles between the Positioning tool and the Wiring Tool. The tab key toggles through all four of these modes. There are also several other tools for debug and coloring but these tools must be selected from a pop up menu. Programming LabVIEW is like playing a piano. The programmer needs to use both hands, interpret what needs to be done by looking at the screen, and select the right tool. The tool also needs to be unselected when a task is complete or the programmer will end up wiring lines to objects he or she wanted to move or select.

HP VEE's cursor mode is automatically selected by HP VEE depending on where the cursor is located. This is a great feature. For example if the cursor is:

1. close to pins it becomes a wiring tool.
2. over an object it becomes a positioning tool.
3. over ASCII data it becomes a labeling tool.
4. on a line it becomes a probe
5. over nothing it becomes a scrolling tool.

This makes writing programs in HP VEE much easier than LabVIEW. The programmer does not have to think about acquiring tools, using them, and then switching to other tools. It is like working on a car with a magic hand that changes into a wrench or screwdriver depending on whether it is close to a nut or a screw.

To illustrate the difference this table compares the number of mouse clicks, tab, and space key hits required to create the Extraction Example. Remember; before a majority of mouse clicks the cursor must be exactly positioned.

	Mouse Clicks (Single and Double + Tab and Space Hits for LabVIEW)	Other Key Hits	Total
LabVIEW	119	78	**197**
HP VEE	45	98	**143**

Execution and Debug

LabVIEW has historically had an edge over HP VEE in execution speed. But with the latest revision of HP VEE the gap is closing. Determining execution speed would require benchmark test cases and is beyond the scope of this article. Before deciding which program to use for an application where execution speed is critical it would certainly be worthwhile to benchmark critical functions with both applications.

Both LabVIEW and HP VEE have classic debug features. Breakpoints can be set on any object and probes can be created to look at the data on any line. There is a major difference between LabVIEW and HP VEE probes. To create a LabVIEW probe the program must be paused or stopped and the probe tool must be used to select a line. A new window then appears to display data flowing on the line selected. The label of the new window matches a label attached to the line selected. Data only appears in the new window if the program is running. The process for creating HP VEE probes is similar. The program must be paused or stopped and then a click on the line creates a HP VEE probe. A window appears to reveal the data that currently exists on the line.

The big difference between LabVIEW and HP VEE is that HP VEE does not require that the program be running in order to view probe information. This means that probe locations need not be preplanned. In LabVIEW with numerous probe windows open and the program running, screen space can be quickly used up making line tag identification difficult. A HP VEE programmer can very quickly probe every line in a program. To make probes that display data while the program is running alphanumeric display objects may be added to the program with no impact on the user interface.

Example Program Comparison

Introduction

The best way to compare these two programming languages is to examine an identical program written in both. The program shown comes with LabVIEW 4.0. The program is a two-channel oscilloscope simulation. This program contains all the basic elements described above and does not utilize advanced libraries or require instruments.

The HP VEE model was written to duplicate the functions of the LabVIEW model. It was not written to duplicate the exact program logic of the LabVIEW model. The functions of the LabVIEW model are as follows:

1. Display square and sine wave functions with noise.

2. Drive what functions to display using a slider.

3. Drive the X scale of the waveform chart using a knob.

4. Drive the Y scale of the waveform chart using a knob.

5. Simulate an oscilloscope trigger using switches and knobs.

6. If trigger source is off roll the waveform chart and disable slope and level buttons.

7. If trigger source is on simulate an oscilloscope's trigger mechanism.

8. Disable the slope, source, and level controls if Channel A is selected

9. Set trigger source to external if Channel A is selected

10. Set trigger level to zero if external trigger is selected.

11. Provide a button to stop the program

12. Provide a button to pop up information about the program.

User Interface Comparison

The size of buttons and graphs is of no significance since they can be resized in either program.

Differences

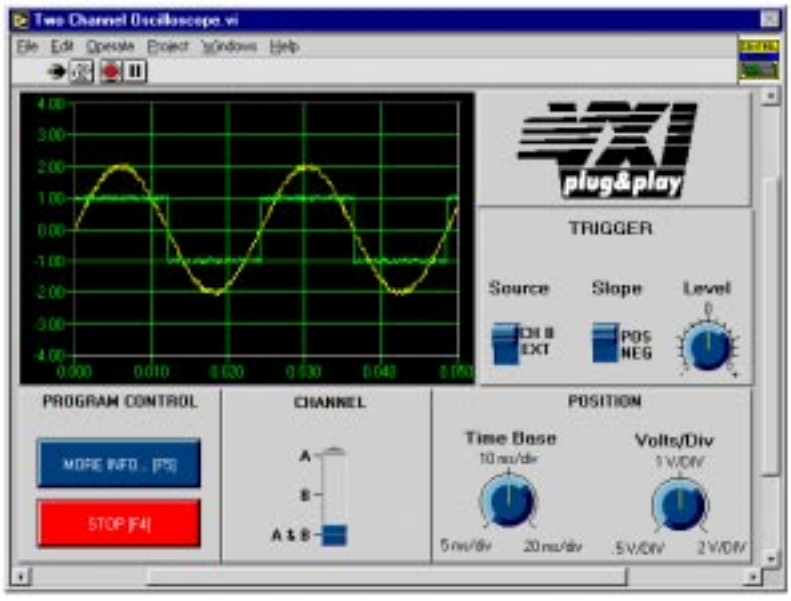
1. The tic increment of any HP VEE graph cannot be set. The programmer can only set min and max values for the axis. The increment is chosen by HP VEE. Therefore, the HP VEE model knobs have been labeled "Duration" and "Span". The LabVIEW knobs are misleading. The knobs indicate that they control the increment of the X and Y-axis. The knobs actually control the min and max values for the axis. The increment value for the axis is also input into the display but it is only a suggestion, LabVIEW chooses an increment that is appropriate based on the size of the graph. If the LabVIEW graph is resized the knobs no longer control the increment.

2. The legend (Trace 1, Trace 2) cannot be eliminated from the HP VEE waveform display without eliminating scale information.

Another difference in the user interface is the method used to disable the source, slope and level inputs. All LabVIEW objects have an attribute, which controls whether the object is enabled or disabled. If the control is disabled it is grayed out on the user interface. There is no such feature in HP VEE. The method used to enable or disable controls is to show or not show them. For comparison, objects have been hidden in the HP VEE program by using a blank pop-up user function, which covers and disables control. If necessary the pop-up panels could have been covered with a grayed out bit map of the buttons. This would precisely replicate the LabVIEW feature.

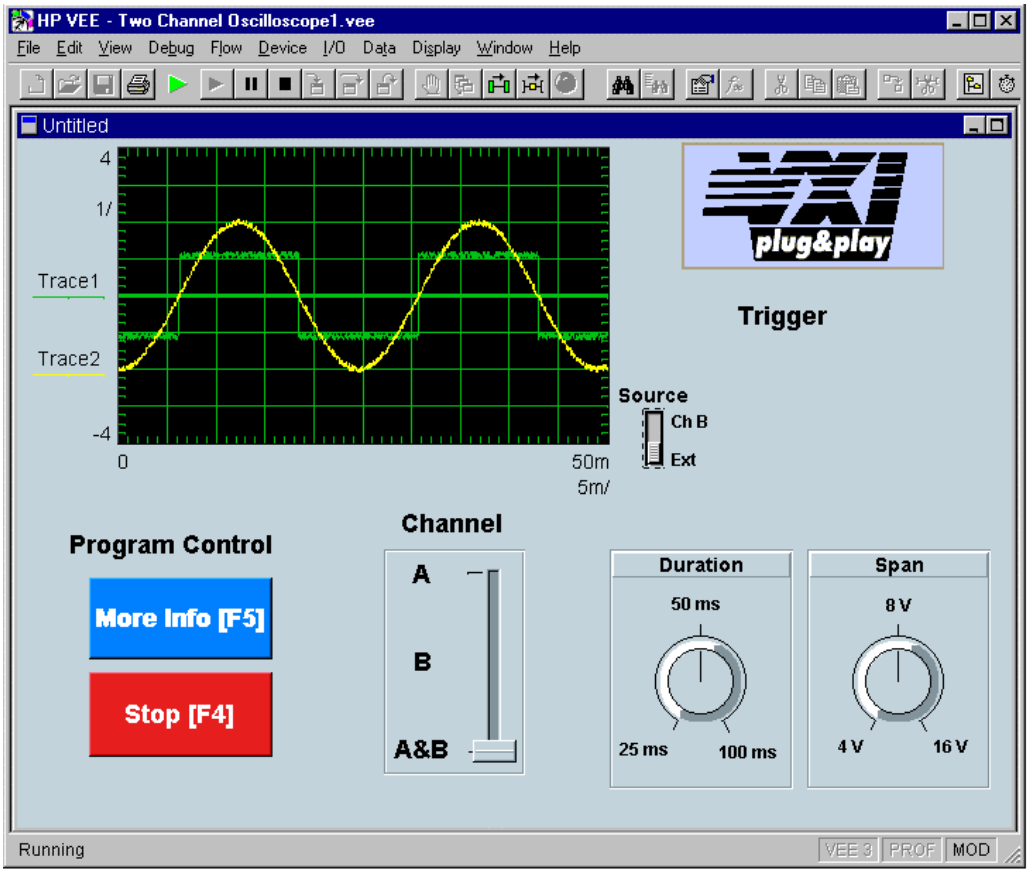
Note: When Channel A is selected in the HP VEE model all of the trigger buttons are hidden.

VEE User Interface Panel

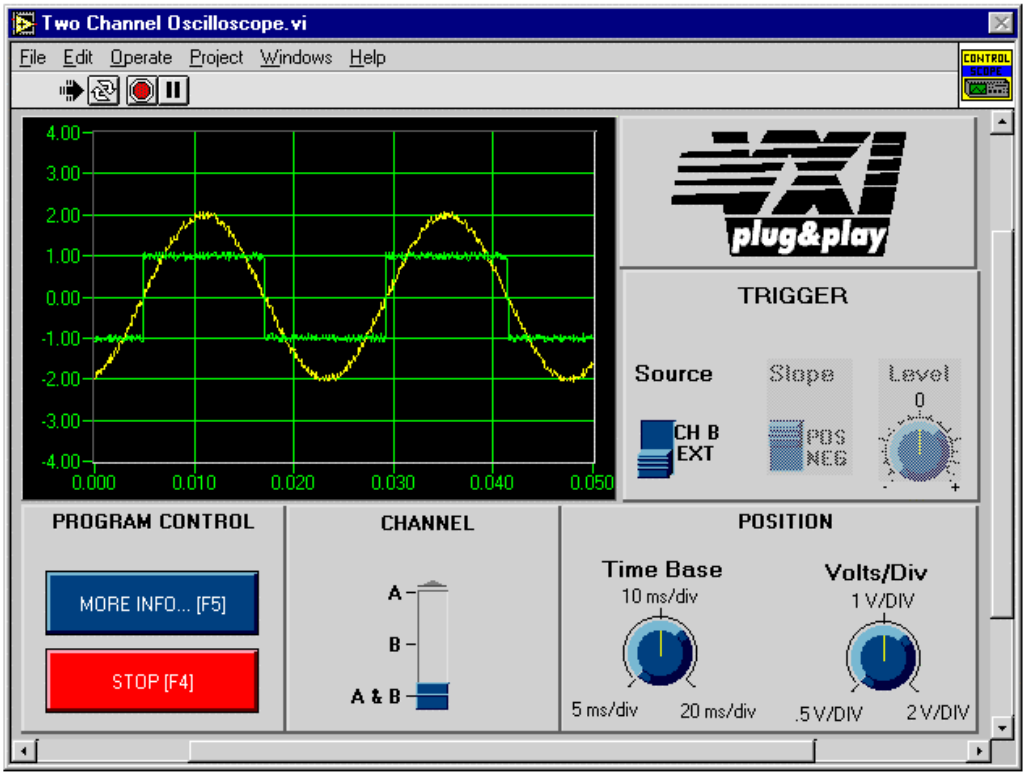


LabVIEW User Interface Panel

Figure 9



VEE Model User Interface with Source=Ext



LabVIEW User Interface with Source=Ext

Figure 10

Program Comparison: LabVIEW Model

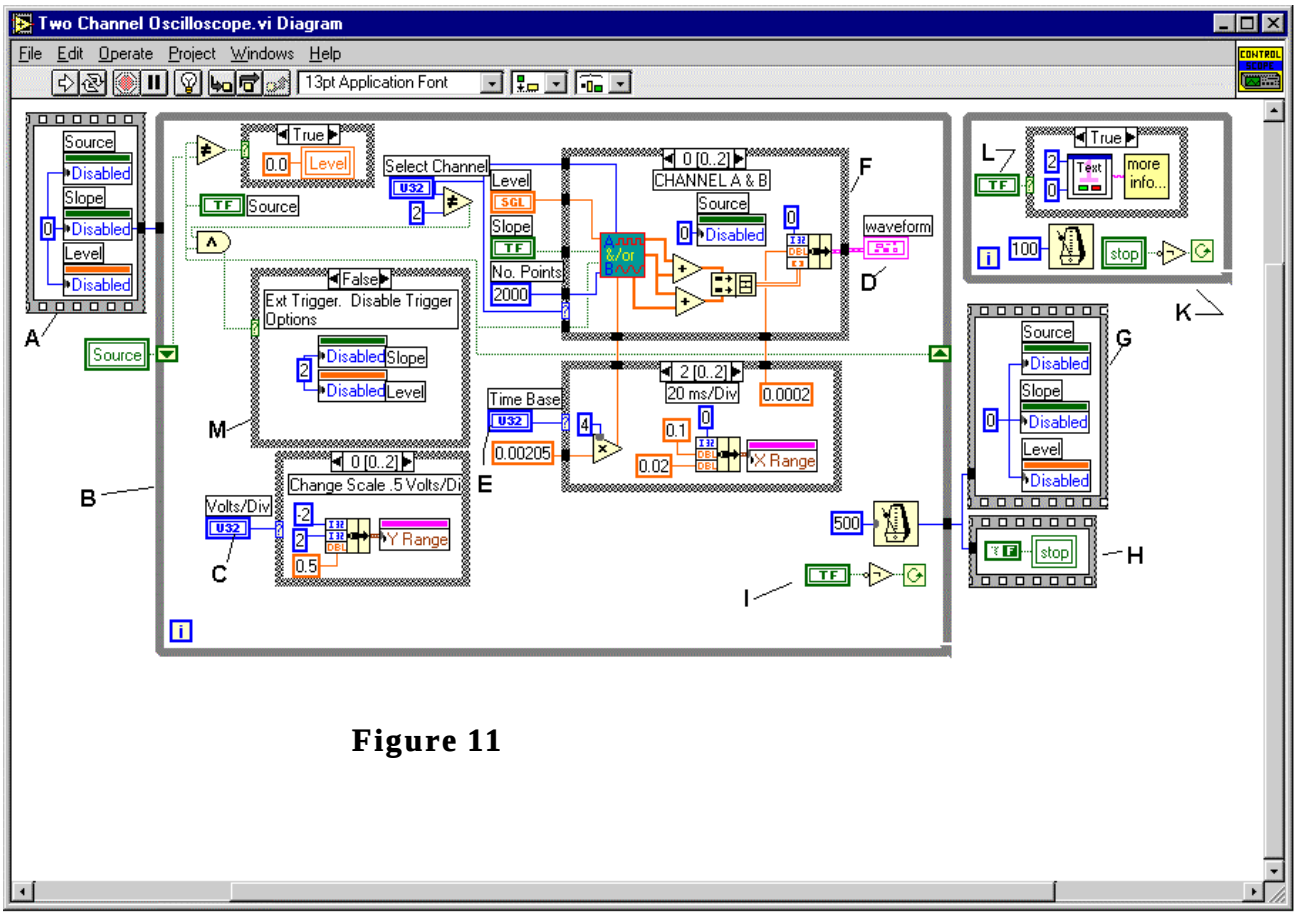


Figure 11

Explanation of LabVIEW Main Program (figure 11)

The sequence frame (Labeled "A") contains attribute nodes for the trigger controls. The purpose of this sequence frame is to enable the switches if they were left disabled by the user aborting the program. The While Loop (Labeled "B") contains most of the program logic and controls. The Volt/Div control (Labeled "C") is wired to a case structure, which contains constants that are built into a cluster and then wired into the Y Range attribute of the waveform display (Labeled "D"). The Time Base control (Labeled "E") is wired to a case structure that controls the X Range attribute of the waveform display. This case structure also sets up frequency and time domain information for the case structure above. The rest of the controls (Select Channel, Level, and Slope) are wired into the case structure labeled "F". This case structure is controlled by the Select Channel control and passes data into a VI (see figure 12) that generates the waveform data with trigger simulation. The waveform data is then combined with time domain information by a build cluster object and routed to the waveform display. The case structure labeled "M" is used to disable the Slope and Level controls if the logic that is wired into it is false. The sequence frames labeled "G" and "H" are sequenced by a dummy data line. They enable the source, slope, and level controls and reset the stop switch once the While Loop is terminated by the stop button labeled "I". The While Loop labeled "K" runs in parallel with While Loop "B". This While Loop continuously checks to see if the More Information button labeled "L" is pressed. If the More Information button is pressed then the case structure inside loop "B" pops up a panel showing information. While Loop "K" is controlled by the same stop button (Labeled "I") as While Loop "B". Using a Local Variable labeled "stop" does this.

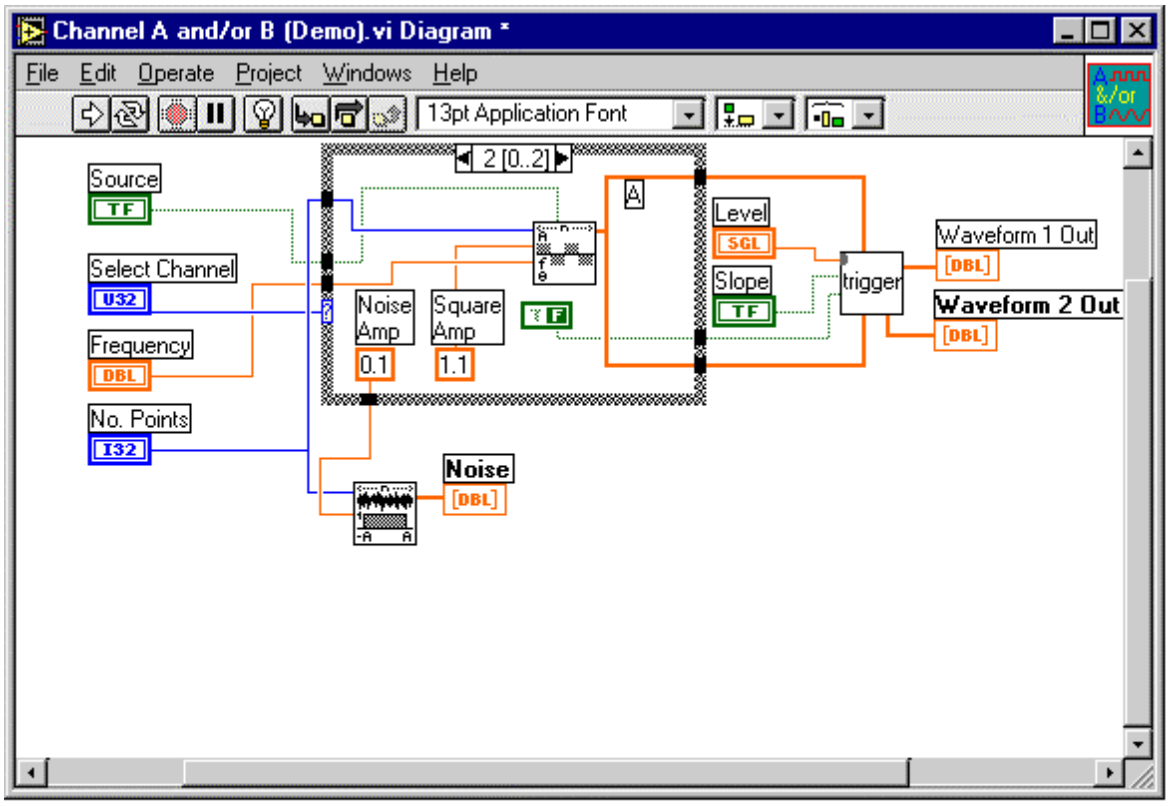


Figure 12

Figure 12 is the diagram for Channel A and/or B (Demo).vi. Inputs to this routine are Source, Select Channel, Frequency, and No. Points. The outputs of this routine are Noise, Waveform 1 Out, Waveform 2 Out. The case structure in the middle of the program is driven by the Select Channel control. The case generates a sine wave, a square wave, or both depending on which channel is selected. The case also sets source equal to False if the channel selected is "A". The trigger VI called is described in the next figure (figure 13).

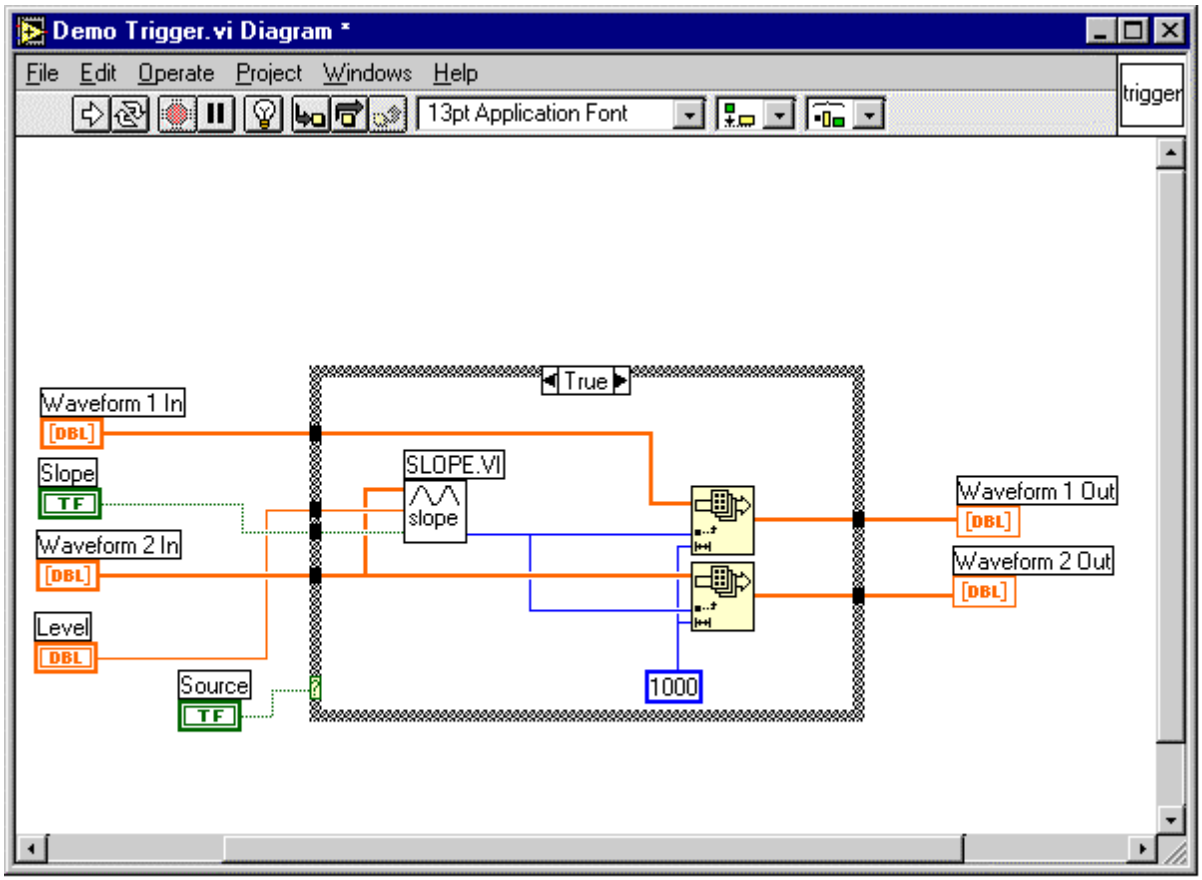


Figure 13

The trigger.vi simply rearranges the waveform array if the Source is True (Source = Ch B). Inputs to this routine are Waveform 1 in, Waveform 2 in, Level, and Source. Outputs of this routine are Waveform 1 out, and Waveform 2 out. If the source is true (Source = Ch B) then the routine Slope.vi is called and the index returned is used to extract a part of the waveform.

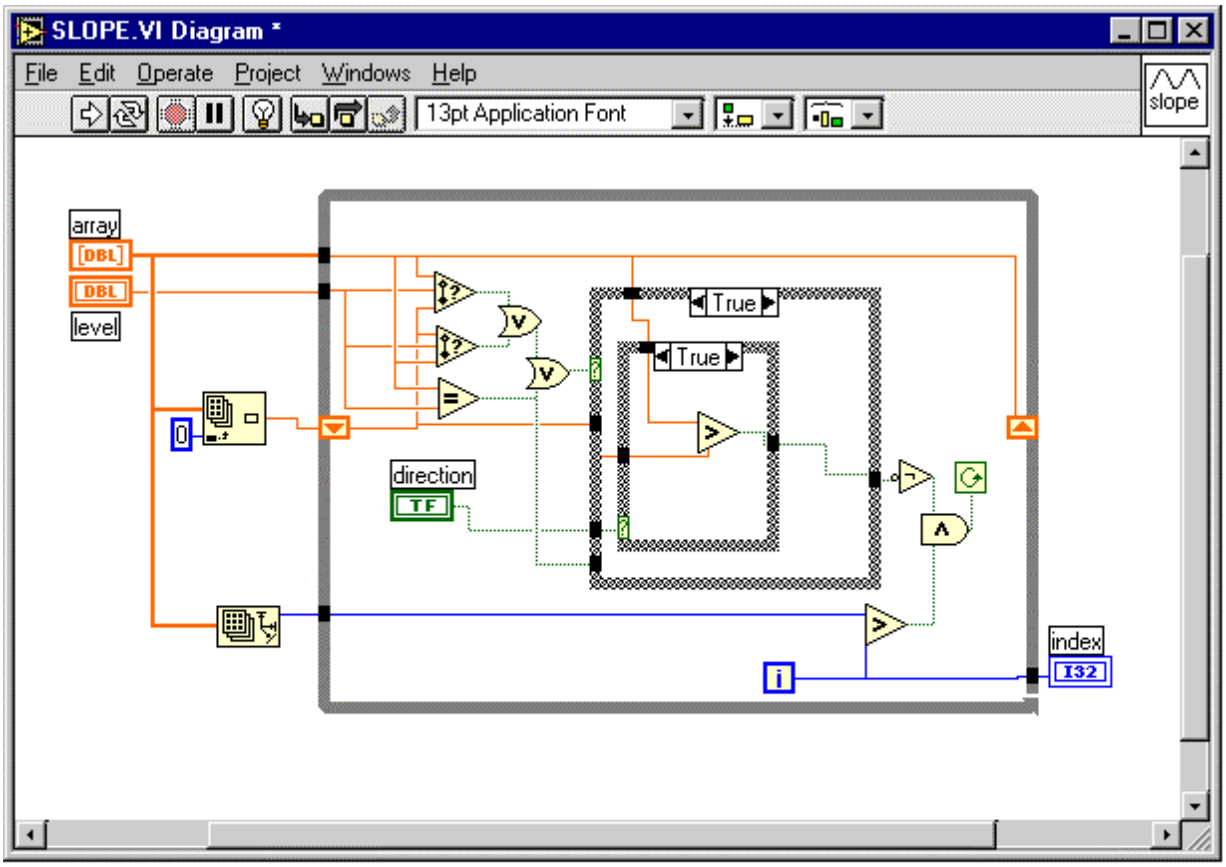


Figure 14

The purpose of the slope.vi routine is to return the index of an array based on the characteristic of the array. Inputs to this routine are array, level, and direction. The output of this routine is index. For comparison purposes the HP VEE function on the following page (figure 15) duplicates the logic exactly.

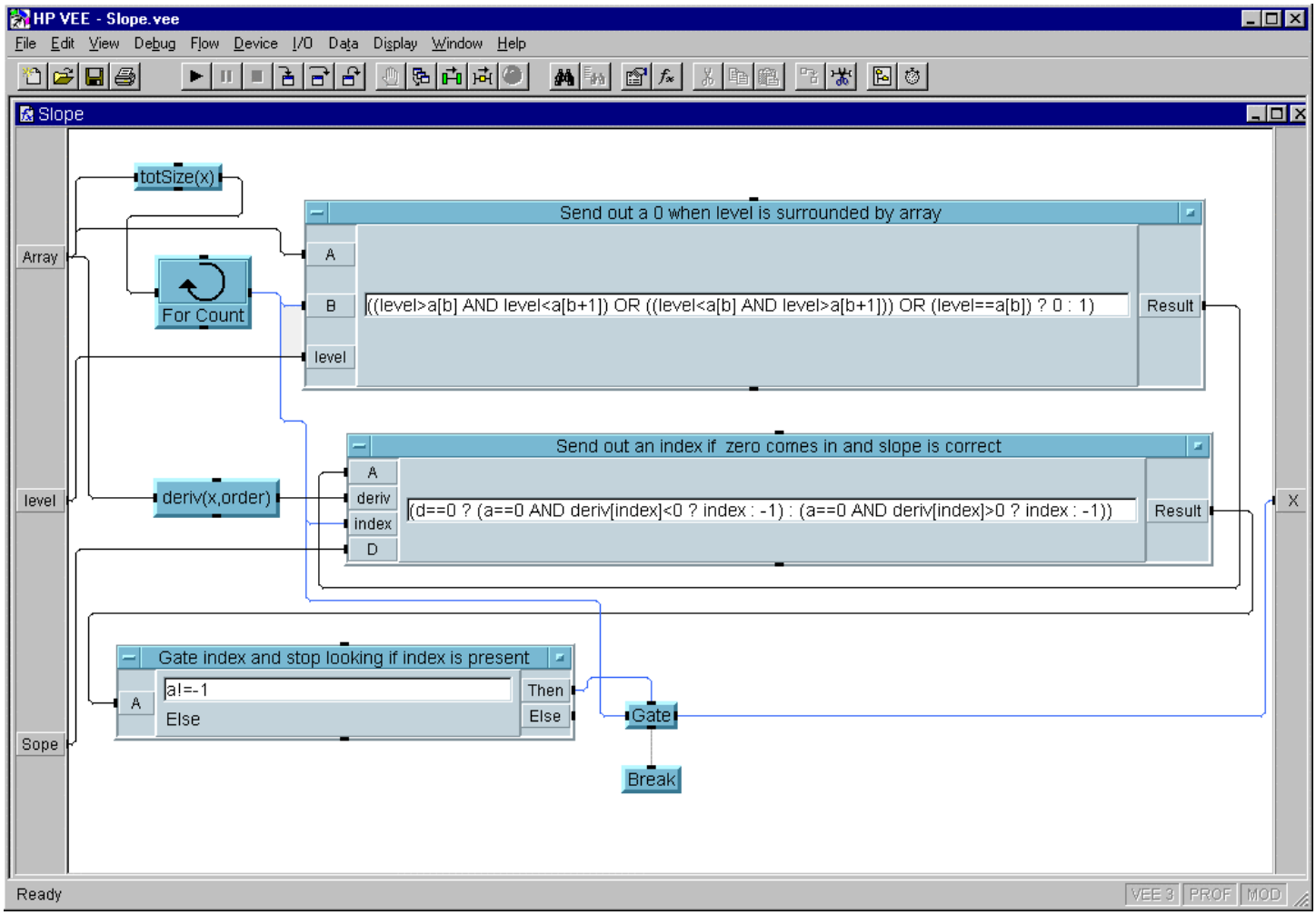

HP VEE - Slope.vee
File Edit View Debug Flow Device I/O Data Display Window Help
Slope
totSize(x)
Array
For Count
Send out a 0 when level is surrounded by array
A
B
level
((level>a[b] AND level<a[b+1]) OR ((level<a[b] AND level>a[b+1])) OR (level==a[b]) ? 0 : 1)
Result
Send out an index if zero comes in and slope is correct
A
deriv
index
D
level
deriv(x,order)
(d==0 ? (a==0 AND deriv[index]<0 ? index : -1) : (a==0 AND deriv[index]>0 ? index : -1))
Result
X
Gate index and stop looking if index is present
A
a!=-1
Else
Then
Else
Gate
Break
Sope
Ready
VEE 3
PROF
MOD

Figure 15

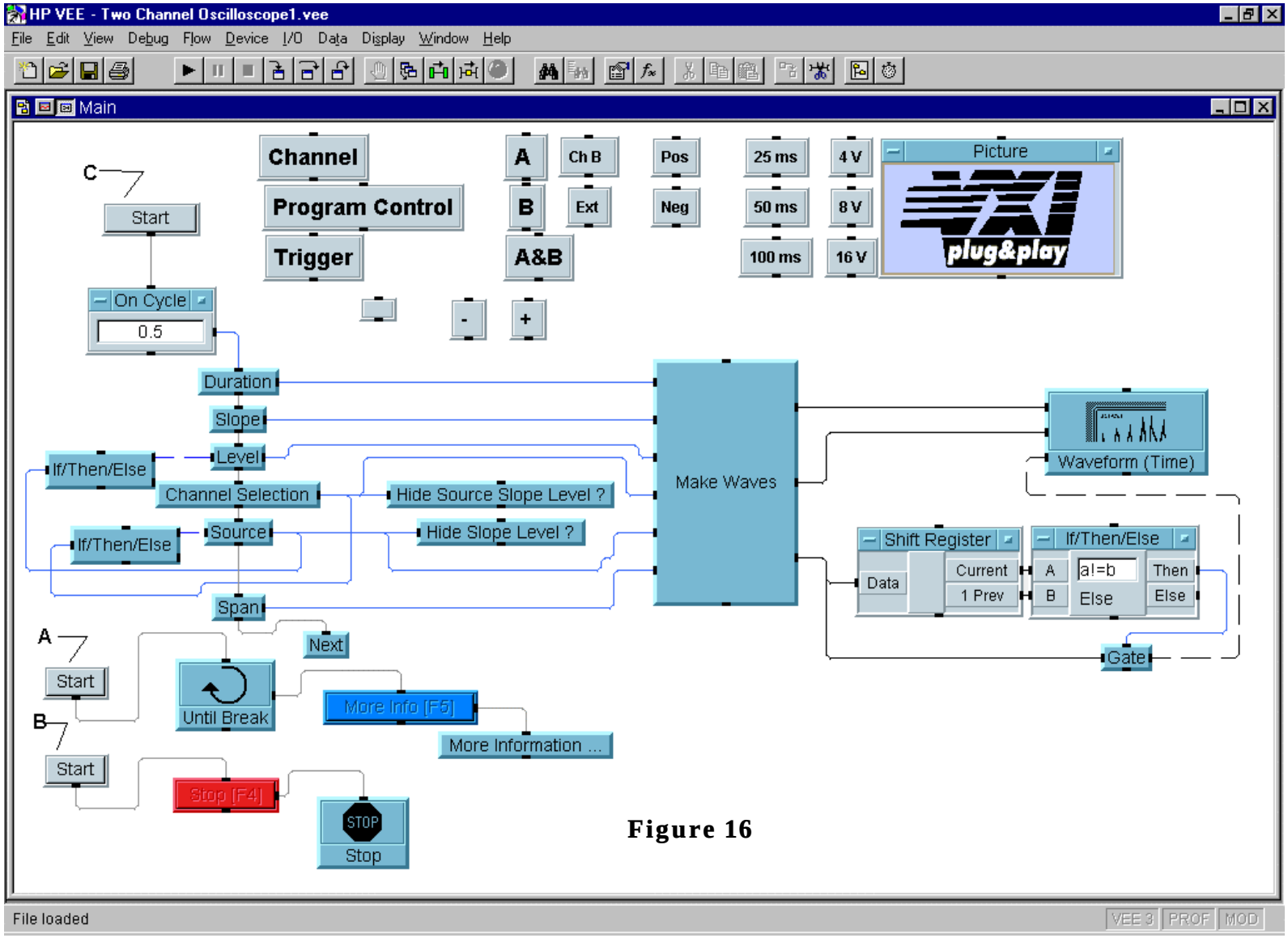

HP VEE - Two Channel Oscilloscope1.vee
File Edit View Debug Flow Device I/O Data Display Window Help
Main
Channel
Program Control
Trigger
A
B
A&B
Ch B
Ext
Pos
Neg
25 ms
50 ms
100 ms
4 V
8 V
16 V
Picture
C
Start
On Cycle
0.5
Duration
Slope
Level
If/Then/Else
Channel Selection
Hide Source Slope Level ?
Source
If/Then/Else
Hide Slope Level ?
Span
Next
Make Waves
Waveform (Time)
Shift Register
Data
Current
1 Prev
If/Then/Else
A
B
a!=b
Else
Then
Else
Gate
A
Start
B
Start
Until Break
More Info [F5]
More Information ...
Stop [F4]
STOP
Stop
File loaded
VEE 3
PROF
MOD

Figure 16

Program Comparison: HP VEE Model
Explanation of HP VEE Main Program

The HP VEE model consists of three threads and several labels used to create the user interface. The threads can be described as three independent programs that are all started when the run button is hit. A thread can also be run independently by hitting its Start button. The thread containing the More Information (labeled "A") button functions exactly like While Loop "K" in the LabVIEW model. The only function of the thread labeled "B" is to stop the program when the stop button is hit. The remaining thread (labeled "C") reads the controls, generates data, hides controls (if required), and displays the data using a waveform display. This thread contains an On Cycle object so the thread is executed every 0.5 sec. This was done to replicate the 500-millisecond wait in the LabVIEW model. The two If Then/Else objects (labeled "D" and "E") are used to reset the Level and Source controls depending on Source and Channel selection.

The object labeled Make Waves (figure 17) generates the waveform data and scale information depending on the control settings. The object labeled "Hide Source Slope Level ?" hides the trigger controls if channel A is selected. The object labeled "Hide Slope Level ?" hides two of the trigger controls if the source is set to external. The Shift Register If Then/Else, and Gate objects in the lower right hand corner simply pass scale information to the Waveform display if the scale information has changed since the last iteration. This logic was added to eliminate the flash that accompanies scale updates of HP VEE waveform displays. The labels at the top of the program are required in the programming window because everything displayed on the user interface panel must exist in the programming window.

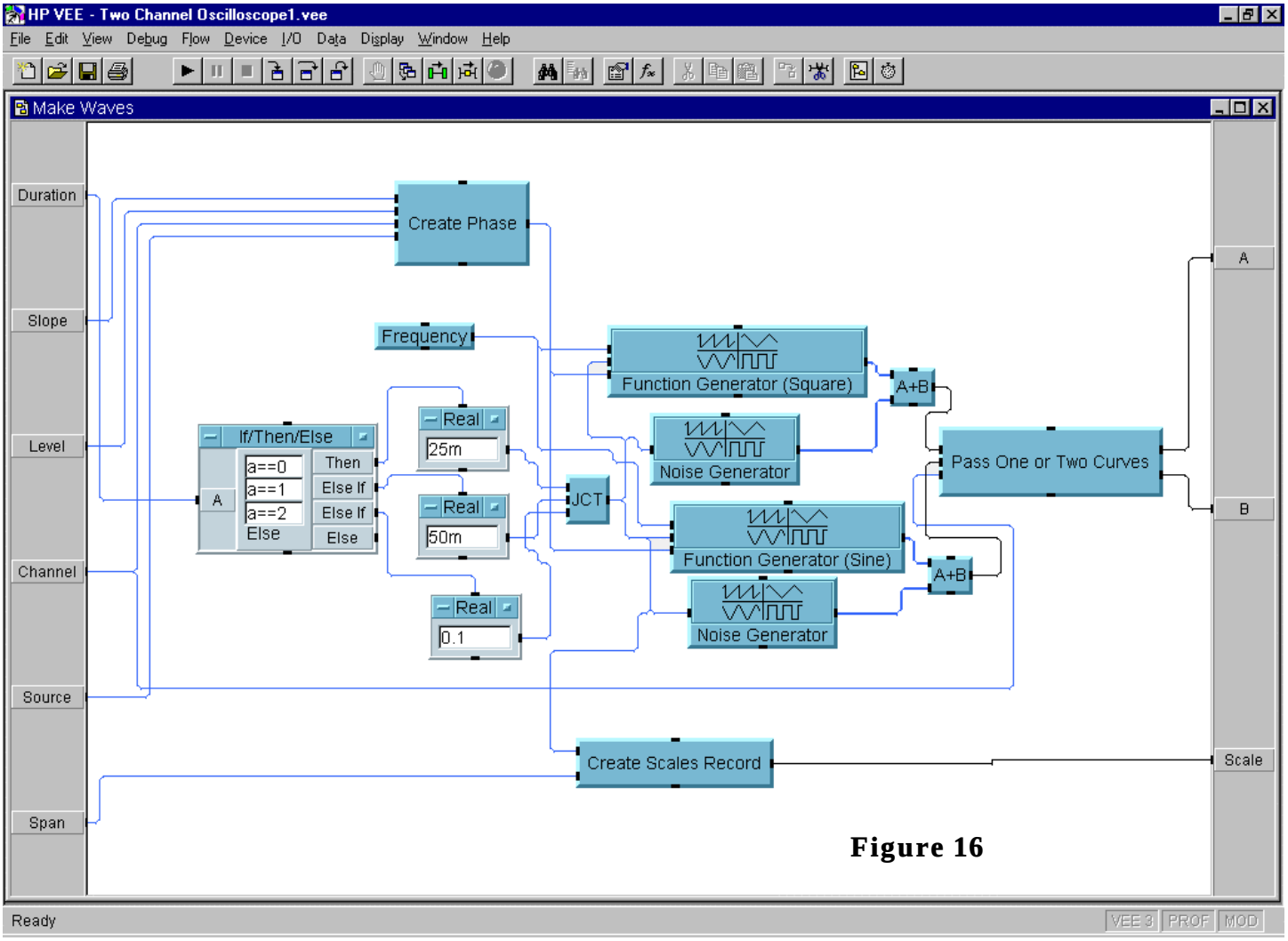


Figure 16

The Make Waves object (figure 17 on previous page) is a user function that accepts inputs from the controls and generates waveforms. HP VEE function generators are used to create the waveform data. The three inputs on the function generators are frequency, phase, and time span. The time span is based on the duration selected. The frequency is constant and the phase is varied by the Create Phase object to simulate a scope trigger mechanism. The object labeled Create Scales Record accepts Duration and Span as input and returns a HP VEE scale record. The object labeled Pass One or Two Curves accepts the two waveforms and the Channel Selected and returns one or two waveforms.

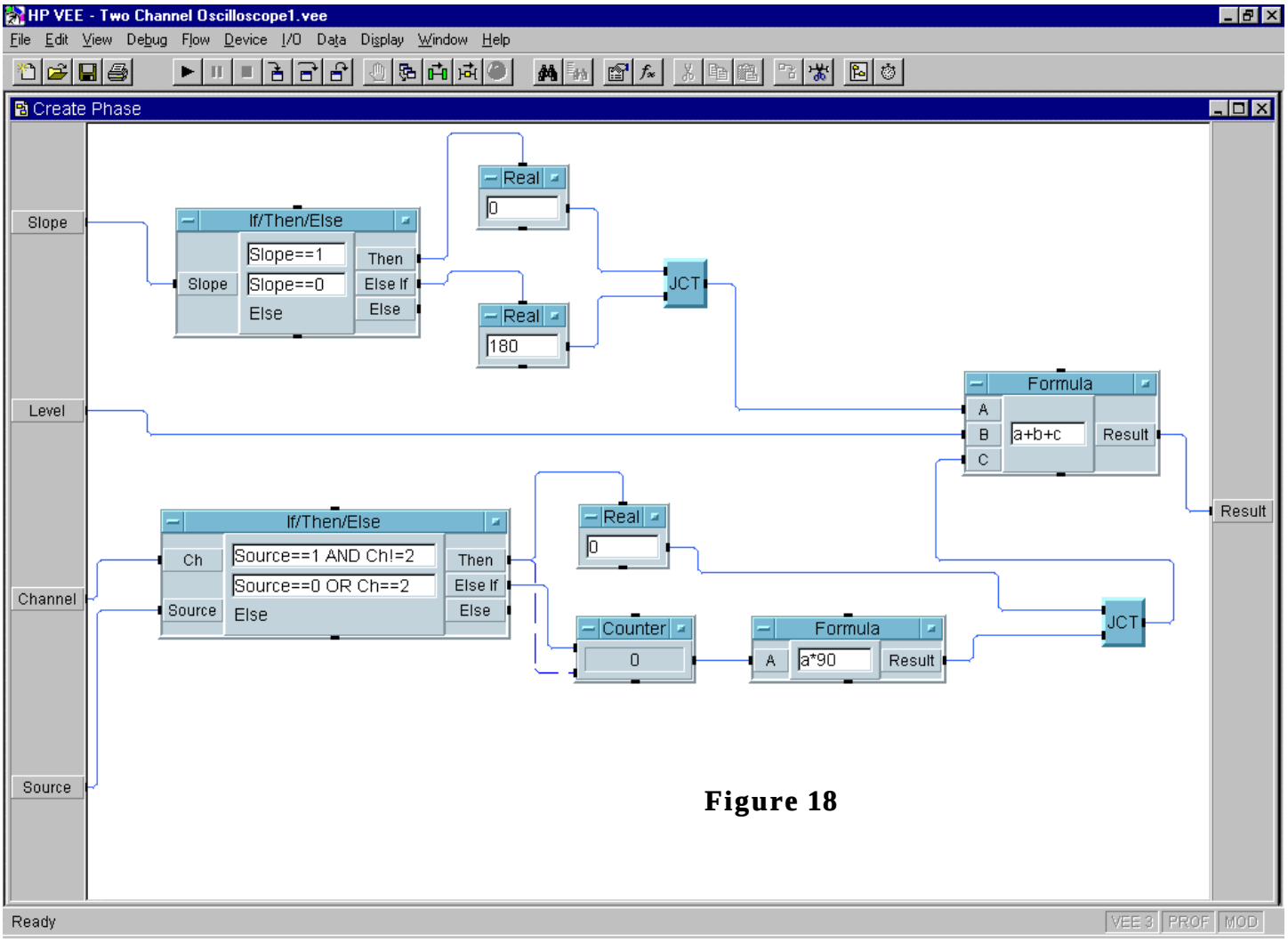


Figure 18

This object (Create Phase, figure 18) creates the phase of the waveform based on the control settings. The slope switch shifts the phase by 180 degrees. If the source is off or channel 2 ("A") is selected the waveform needs to roll so a counter object is activated and the output is used to roll the waveform.

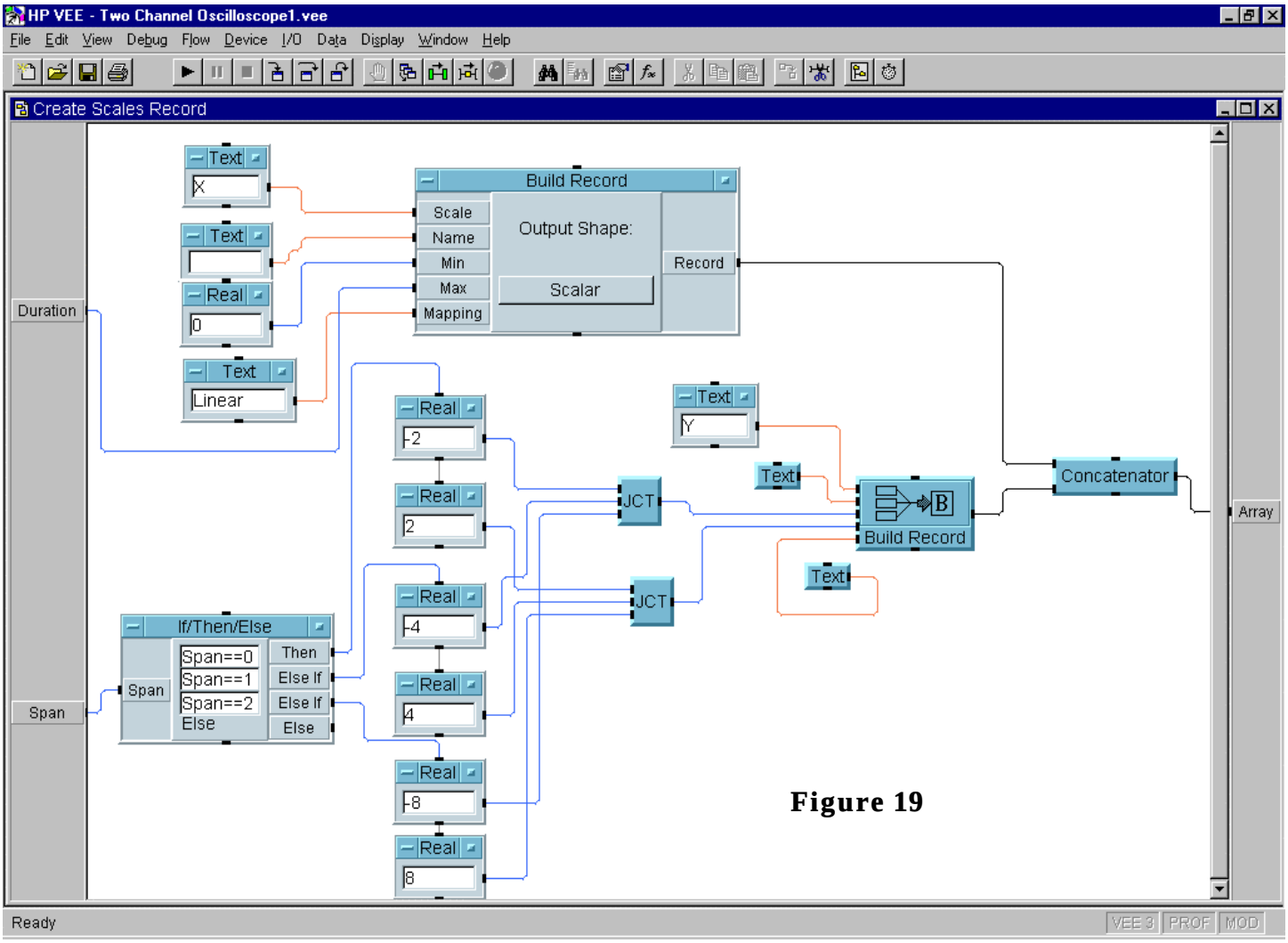


Figure 19

This object simply creates a HP VEE scale record based on control inputs.

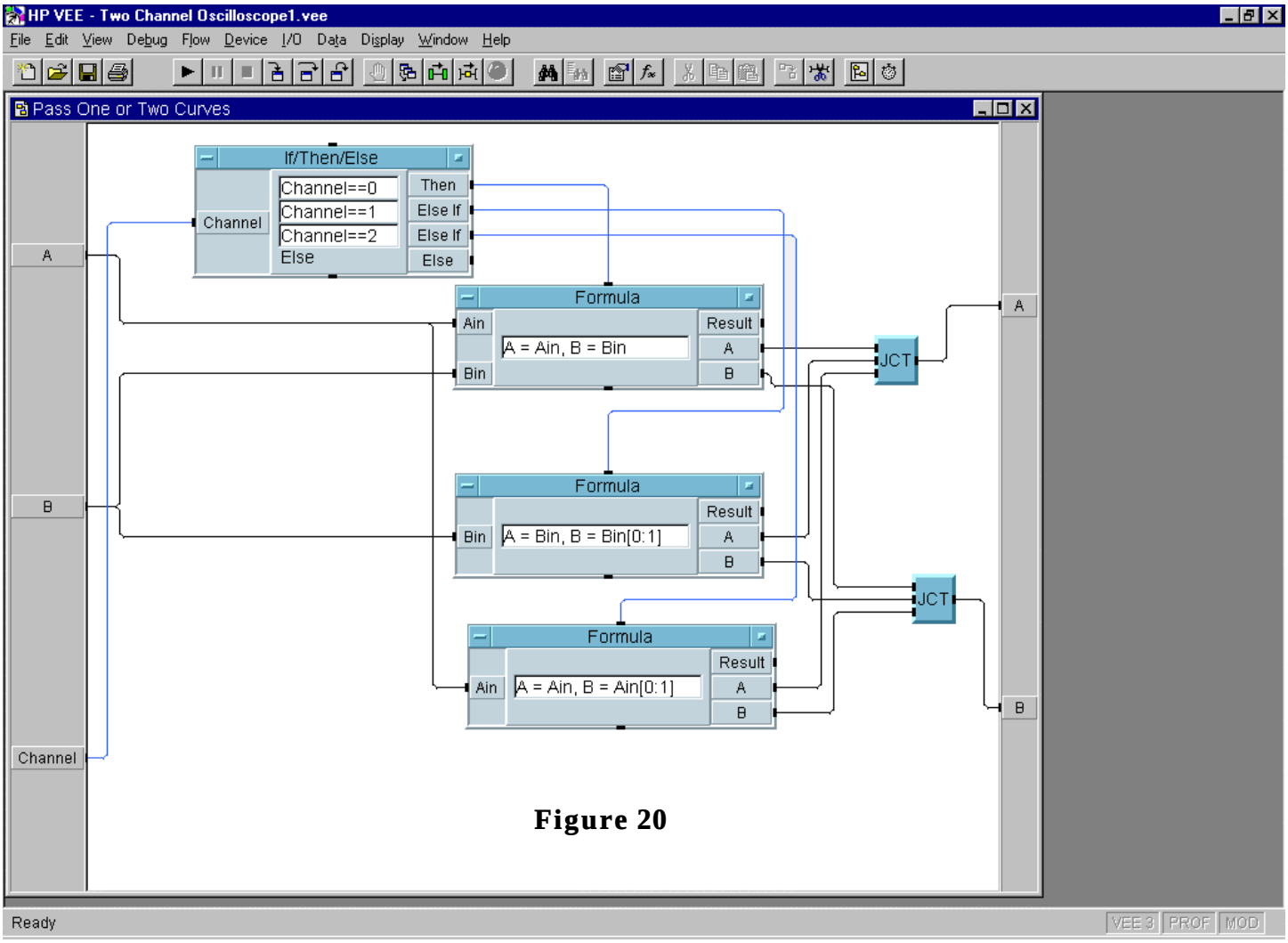


Figure 20

This object (figure 20) is needed because all input pins on a display must contain data. Depending on which channel is selected this routine sends out both waveforms or one waveform and two points. The two points are taken from the waveform passed out. This is done in case the waveform display is auto-scaled. This routine is not required in LabVIEW because LabVIEW displays accept arrays as input. Each element of the array becomes a trace on the display. Basically LabVIEW programmers can vary the number of traces on a display without changing code or writing routines like this.

Object and Line Count Comparison

To obtain a comparison of the effort required to create the comparison program all of the objects and lines were counted. This was accomplished by opening up the program and counting the lines and objects while they were being deleted. The number of objects and lines required by LabVIEW is substantially more than what was required by HP VEE (see tables below).

HP VEE Routine	Objects	Lines
Main	39	30
Hide Slope Level?	4	6
Hide Source Slope Level?	4	6
Make Waves	16	32
Create Phase	10	17
Create Scales Record	19	26
Pass One or Two	6	16
Total	**98**	**133**

LabVIEW Routine	Objects	Lines
Main	118	109
Channel A and or B (Demo)	24	42
Trigger	12	20
Slope	21	44
Total	**175**	**215**

Conclusion

In this article I have attempted to make a fair comparison between HP VEE and LabVIEW. As I stated in the introduction, each has a number of advantages. LabVIEW generally allows more control of display and interface object attributes. The programmer can set and query attributes. A HP VEE programmer can only set certain attributes and none of the attributes can be queried. Most cosmetic HP VEE attributes are set using pop-up menus and cannot be set programmatically. LabVIEW local variables and attribute nodes really break up data flow programming. Tracking the data flow even in the simple example above can be difficult. Historically, LabVIEW has had faster execution times. LabVIEW graphs accept arrays as inputs and may be more suited to multiple curve displays.

HP VEE is much easier to use than LabVIEW because the programmer doesn't have to wire as much information and can utilize HP VEE's powerful auto line routing. The object and line count comparison in the article above is a good indication of how much faster HP VEE programming is. HP VEE is easier to learn, faster to use and more error-proof. The ability in HP VEE to type in logic and formulas is an enormous time saver and a very powerful tool. LabVIEW also has a formula box but it has very limited capabilities. Another time and irritation saving feature of HP VEE is the amazing tool-morphing capability. Once a programmer has experienced this impressive device it is frustrating to go back to swapping tools manually. HP VEE maximizes readability by allowing easy resizing of objects, keeping all case structures visible and allowing placement of user interface panels. One feature that is not mentioned in the article is that HP VEE files are stored in ASCII format. This can be a great advantage for file transfers between operating systems, editing, and even documentation.

As you have seen through a discussion of features and a comparison of examples LabVIEW and HP VEE offer very different solutions to graphical programming. In making a decision between two options it is easy to focus on just one feature that seems important or one deficiency that seems insurmountable. Considerable time and expense will be invested in the purchase of a graphical programming language and, more importantly, in becoming an expert programmer.

In considering the overall performance, ease of use and capabilities of both programs I must recommend HP VEE over LabVIEW. For the purposes of this article HP VEE version 4.0 and LabVIEW version 4.1 were compared. As new versions are released we can expect improvements in both products but I feel that the underlying premise of HP VEE will continue to be simple straightforward programming and the expert HP VEE programmer will gain even greater abilities to produce powerful software with amazing speed.

Happy programming!