How Do I Debug my VI?

Contents:

Inevitably there are going to be some cases where you think you've just about finished your block diagram and you think everything has been wired correctly, but then when you go to run your VI something catastrophically fails, or maybe the output is off by 1 value, or your VI never seems to terminate. These are instances when you want to debug your VI in order to attain the functionality that you intended.

Highlighting Execution

Highlighting execution is a method of debugging that allows you to trace the flow and values of data through the VI block diagram. When execution is highlighted, you will see the computed values at every step in the execution from ever function block as shown in Figure 1. Click here for the Running VIs and Highlight Execution Tutorial.

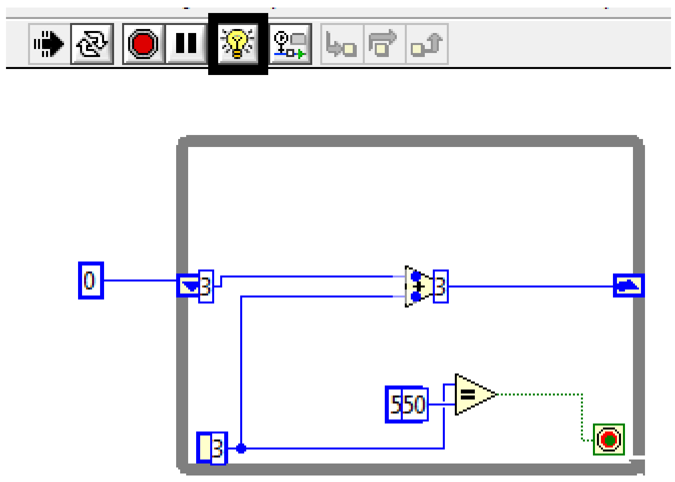


Figure 1

The Probe

Using probes is another useful way of diagnosing a bug in your VI. A probe is simply an indicator that can be placed on any wire to monitor the values that the wire holds. To create a probe, right-click the wire you want to probe and select the "Probe" option as shown in Figure 1.

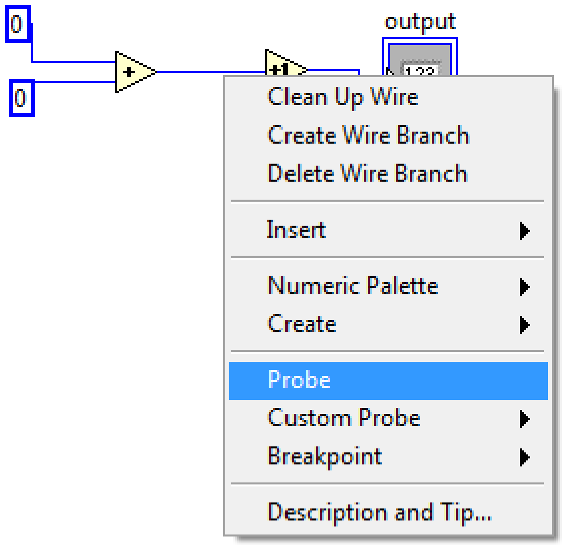
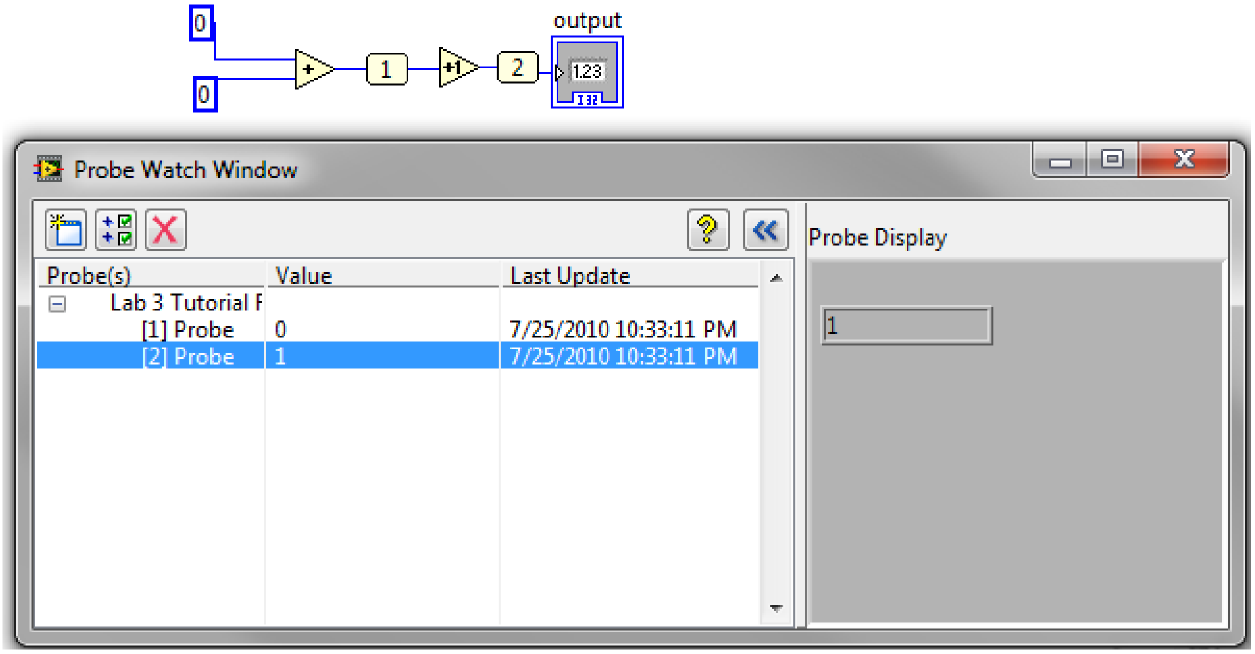


Figure 2

You will notice that when you create probes, a "Probe Watch Window" will appear which shows the value that each probe detects as shown in Figure 2. Remember that the number on the probe is NOT the value that probe is reading but is the probe number.



Probe(s)	Value	Last Update
Lab 3 Tutorial F		
[1] Probe	0	7/25/2010 10:33:11 PM
[2] Probe	1	7/25/2010 10:33:11 PM

Figure 3

Common Errors

Incorrect Wiring

One source of error may be incorrect wiring. You may have intended one wire to connect to a certain pin but you accidentally connected it to an adjacent pin or the wrong function block. These errors are easily fixable once you determine which wire(s) caused your VI to fail.

Consider the example below in Figure 3 which is intended to compute (9 - 2)/(5 + 3)

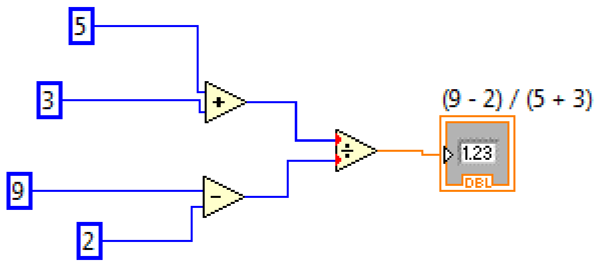


Figure 4

Notice that the way the block diagram is wired, it computes (5 + 3)/(9 – 2) instead of the desired value because the wires to the divide block are not connected correctly. Figure 4 shows how the block diagram should actually be wired.

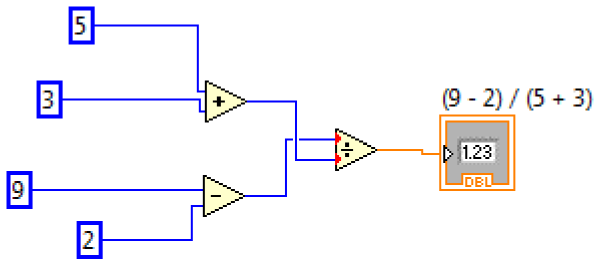


Figure 5

Infinite Loops and Bad Termination Cases

A common source of errors with loop structures is a bad termination case or the conditions for ending the loop are never satisfied. This results in an infinite loop or an execution that never finishes which you will need to abort.

Consider the following example that is intended to sum numbers until the sum reaches 50 and terminates.

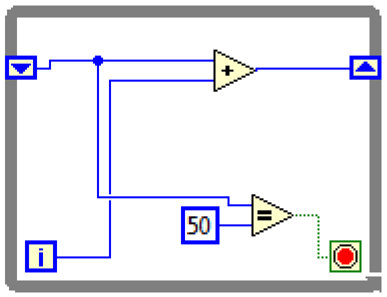


Figure 6

Notice that there are a few errors with this set up. One is that there is no default value for the shift registers. Another more prominent error is that when we sum up the first 10 terms, we never obtain 50 as one of the partial sums, thus, when we execute this loop, the loop will never terminate since the base case signals a termination only when the partial sum equals 50. The fixed version of this VI is shown in Figure 6.

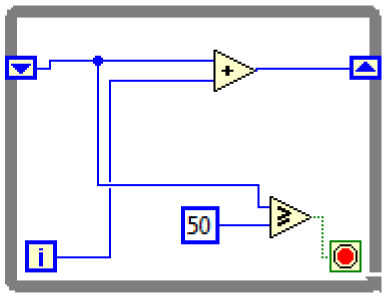


Figure 7

Initializing Default Values

Another less obvious source of error is forgetting to initialize default values, especially when using shift registers. If a default value is not specified for blocks like shift registers, your VI may behave unexpectedly since it is not resetting its default value between executions. For example, consider the following loop structure which is designed to compute the smallest integer n such that the sum of the first n integers is the smallest sum that exceeds 50. See Figure 7

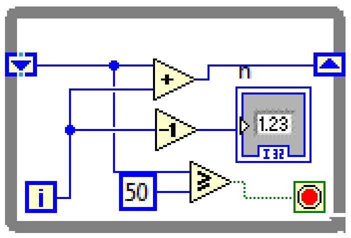


Figure 8

For the first run the value returned by the VI is correct. However, we will notice that on subsequent executions the VI returns an incorrect value. The reason for this is that the shift registers are not initialized and have no default value. Therefore, the value that was stored in the shift registers on the last iteration of the previous execution was used as the initial values causing an early termination. Figure 8 shows how this error can be fixed by adding an initial value.

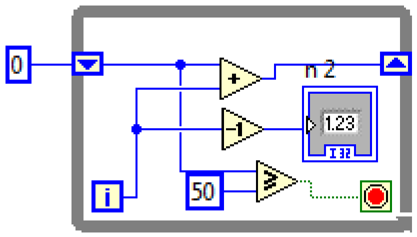


Figure 9

Tutorial Notes

SECOND SET OF REVISIONS COMPLETED 12:42 PM 8-16-10