Assignment 9

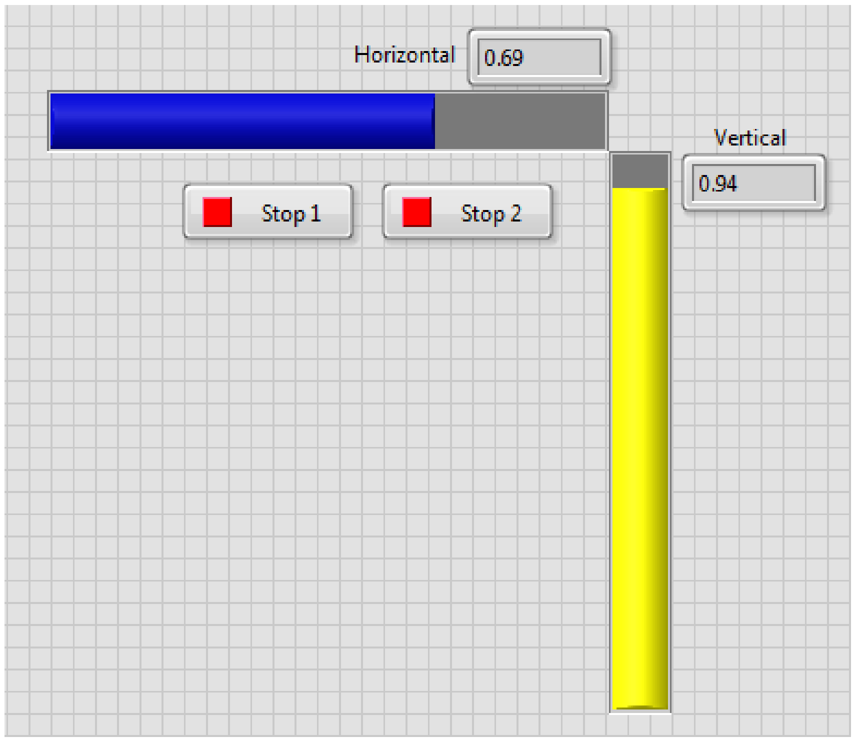
(Due Monday 31 October)

Parallel Loops

A strength of LabView that appeals to many programmers is its ability to have multiple loops running simultaneously and asynchronously. This is possible in (some) text-based code as well, but it's almost certainly not easy to do. LabView, in contrast, makes this a simple, straightforward process.

Open a blank VI, go to the Block Diagram, create two While Loops, each with a Stop button controller. Inside the first loop, create a dynamic floating point operation that increments over the range 0—1 in steps of 0.015. Have it reset to 0 when it reaches 1. This can be done with a Shift Register or a Feedback Node, depending on your preference. Set the Wait (ms) time for the first loop iteration at 80 ms. Configure the second While Loop with a 0.02 increment and a 50 ms wait. You should now have two asynchronous loops that will run in parallel.

Create indicators for both numerical operations. Go to the Front Panel. Replace the first indicator with Numeric: Horizontal Progress Bar and the second indicator with a Vertical Progress Bar. Right-click on the horizontal indicator and select properties. Make the Height:Width aspect ratio 30:300, and set the scale at 0—1. Also display a numerical indicator. Do the same for the vertical indicator except make the aspect ratio 300:30. Arrange the corners of the progress bars as shown (the Silver palette is used here):



Run the VI and observe that the loops can be stopped independently.

Control Loop

It is not possible to use wires to share data in parallel loops, but data can still be exchanged between them. One way to do this is with Local Variables. Right-click on the vertical indicator icon in the Block Diagram and create a Local Variable. Also create a Local Variable for the horizontal indicator. Right-click on each icon and select "Change to Read". Remove the two Stop Button controls.

Create a third While Loop in the Block Diagram and place the two Local Variables inside it. Develop logic to stop this loop when the horizontal and vertical indicators *both* exceed a value of 0.95. This will require the use of two comparisons together with the Boolean AND function.   Place a Boolean control button on the Front Panel and label it "Master Stop". Put its Boolean icon inside the third loop and insert an OR function. Wire the AND output to one terminal; wire the Master Stop control to the other. The output of the OR operation is wired to the conditional stop of Loop 3. This third loop with the comparison structure will be the Master Loop; when it stops it should force the other two loops to also stop.

Create an indicator on the output of the OR operation. Name it Master Boolean. Right-click on its icon and select "Hide Indicator". This will hide it on the Front Panel only. Right-click on the icon and create two Local Variables and make them both Read operations. Place the Local Variables in the first two loops and wire them to the conditional stops.

If the VI is setup correctly, it should automatically stop all loops if the two sliders converge at the corner, i.e. when their values are both simultaneously greater than 0.95. The VI should also stop whenever the Master Stop is pressed.

When the VI is re-run after the convergence criterion is met, the Master Boolean control will still be TRUE and the VI will immediately halt. This glitch can be fixed by forcing the Master Boolean to FALSE at the start of the program. Create a third Master Boolean Local Variable, but keep it as a Write operation. Wire a FALSE constant to it and place it outside the While Loops. There is still a problem because there exists an ambiguity due to data flow: it is unknown whether this variable will be set before the Master While Loop runs or after. This is an example of a race condition. Implement data-flow logic to unambiguously set (write) the Master Boolean to FALSE *before* the Master Loop starts running. Note that writing to local variables causes slower execution than data propagating on wires and through tunnels, so the program must account for the varying speed of data flow when dealing with this race condition.