LabVIEW Part 1

Introduction

LabVIEW is a standard data acquisition and analysis language within the scientific community. It is a highly effective tool for experimental control and data collection, and necessary for further experimental physics work in physics lab.

LabVIEW is unique among computer languages in that it is a *graphical* programming language. A LabVIEW program is more like a flowchart than like a typical Python or C++ program. This takes some getting used to, and it has some distinct disadvantages; but it does allow for rapid development of laboratory-specific applications.

There are two windows for each LabVIEW program: the "front panel" and the "block diagram". The block diagram is the program, the front panel is the user interface to the program. The menus and tools available to you within LabVIEW depend on which window is currently active. You can switch between the two windows at any time by pressing ctrl-e. Generally when starting a new LabVIEW program, one will first place the inputs (buttons, numeric controls, output displays, etc) on the front panel; then go to the block diagram and "wire" these elements to the desired programming elements such as various types of loops, data-collection routines, equation blocks, case and sequence structures, etc.

Something to keep in mind regarding LabVIEW is that different colors, widths, and textures of "wire" carry different types of information. Floating-point numbers, for example, are thin orange lines while integers are thin blue lines. Arrays of floating-point or integer numbers are thicker orange or blue lines, and so on up to "Dynamic Data Type" (DDT) lines which can carry complex bundles of information and are represented by fat blue-grey braided lines.

There are many different types of tool available in LabVIEW, and it's important to use the right one for the job. You use a tool that looks like a pointer for grabbing an object and moving it around, and another tool that looks like a spool of wire for connecting different terminals with wires, and so on. The easiest way of dealing with tool selection is usually to set it on automatic. This way, LabVIEW observes what you're doing, where you're clicking, and selects the right tool for you automatically, most of the time.

Right-clicking is a very important part of the whole LabVIEW experience, too. Right-clicking on an object —or a part of an object— calls up a context-specific menu which nearly always has the option you want.

Finally, there is extensive help built in to LabVIEW. If you open the context-specific help window, it will show you basic information and links to more detailed information for whatever you are currently pointing at, which is helpful even if you've been using LabVIEW since 1989.[1]

Procedure

Thermometer, in gruesome detail

1. Start LabVIEW, and open a blank vi.[2]

2. Right-click on the front panel and thumbtack the resulting pop-up menu to your screen. You'll be using it frequently... Choose the numeric tools, then a thermometer display from within that sub-menu. Drag the thermometer to your front panel, and label it "Celsius". Drag a second thermometer to your front panel and label it "Fahrenheit". Switch to the block diagram (ctrl-e) and you'll see the two thermometer displays there also. The triangles on the left side of the thermometer displays are the numeric inputs for the theromometers: any number-type wire that is attached there in the block diagram will result in that number being displayed on the thermometer in the front panel.

3. Now let's provide numbers for those thermometers. The LM35 is a temperature sensor in a TO-92 package that, given a $+5V$ supply and ground connection, provides a signal voltage of $10\text{mV}/^\circ C$. (The datasheet is available on the course website.) Wire up an LM35 on your ELVIS II board so that it has the necessary power supply and the signal voltage goes to one of the AI+ connections. Wire the corresponding AI− to ground.[3]

   Right-click on the block diagram, tack the pop-up in place, select Input / DAQ Assistant, and place the DAQ Assistant block on the diagram. A pop-up window will open and give you opportunity to explain to the DAQ Assistant that you wish to acquire signals, analog signals, and they should be voltage signals.[4] You can then choose the channel to

[1] Yep. Possibly even 1988. I don't remember for certain.

[2] LabVIEW programs are called "Virtual Instruments", or .vi's for short.

[3] Always remember that it takes *two* connections to measure voltage. Voltage is the electrical potential *difference* between two points.

[4] Not temperature as you might expect: the temperature options here are for specific types of thermometer such as thermistors and RTDs.

which the sensor is attached. The next window allows you to set the minimum and maximum voltage that LabVIEW may expect on this input (0V and 5V, respectively). You should also set the acquisition mode to be "1 sample on demand". Press "OK": after a slight delay the DAQ Assistant block will re-form itself to reflect its new task.

4. The data output of the DAQ Assistant block is in the form of a DDT. We really don't need all the meta-information contained in a DDT; we just need the single floating-point measurement from the LM35 sensor. Right-click the output triangle and choose Signal Manipulation Palette / From DDT. Place this converter on the block diagram, and in the ensuing pop-up select "single scalar" as the desired output.

5. The output of the "From DDT" block will be a floating-point value, which is what we want to display: but it's scaled wrong. Since the output of the LM35 is 10mV/°C, we need to multiply by 100 to obtain the actual temperature. Go back to the block diagram palette (or right-click to call it up again) and select Programming / Numeric / Multiply (it looks like an op-amp with a × on it) and place it near the From DDT output. Wire the data to one terminal of the × block. Right-click the other terminal of the × block and select "Create Constant". Enter "100" in the constant, then wire the output of × to the input of the Celsius thermometer.[5]

6. The equation to convert °F to °C is

$$F = C * 1.8 + 32$$

We can build this equation by using a collection of multiplication and addition blocks —it's not hard for such a simple equation— but there's another way of doing equations that is more convenient when the calculations are more complicated. In the block diagram palette, choose Programming / Structures / Formula Node. Use this to draw a box on the screen. Right-click on the left edge of this formula box and select "Add Input". Label the input "C". Similarly, create an output on the right edge of the formula box labeled "F". Now type the formula "F = C*1.8 + 32;" in the box. (Don't forget the semicolon — it's required.) Draw a wire from from the Celsius wire to the "C" input, and another from the "F" output to the Fahrenheit temperature display.

[5]You can also set this scale factor while setting up the DAQ Assistant, back on the page when you were setting the minimum and maximum voltages. Remember that next time, it's a useful trick.

7. Take a minute or two to clean up your diagram. LabVIEW programs can be difficult to decode, sometimes, and neatness pays off. You can shift things a pixel at a time by selecting them and using the arrow keys, or further by using shift-arrow combinations, and of course you can also just drag them around. There are also some alignment tools at the top of the window, which work on multiple-object selections. Your block diagram should look something like figure 1.

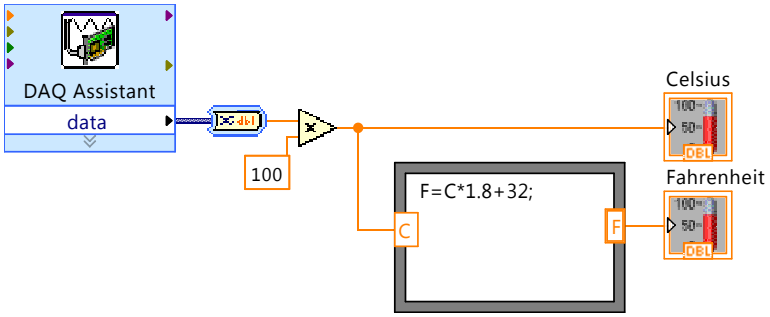

Figure 1: Block Diagram for our temperature sensor virtual instrument

8. Try your program! Return to the Front Panel window, and press the "play" arrow at the top left. Is the resulting display reasonable? It won't be exact: that requires a better calibration of the LM35, but it should be ballpark. Right-clicking on the thermometer displays will give you (among other options) the option of viewing a numeric display as well as the thermometer. This isn't as cool, but it's more useful.

9. Now let's put this routine in a loop so that the temperature display is continually updated. In the block diagram palette, choose Programming / Structures / While Loop, and draw a "while box" around your existing block diagram. Items in a while box run repeatedly until the exit condition is met. The exit condition, in this case, is the little stop sign in the bottom right of the while loop. Right-click on the input to this stop sign, and choose "create control". This will put a "stop" button on your front panel. Now when you run the program, the temperature display will be updated until you press "stop".

10. *Optional:* Give the temperature sensor something interesting to read. Pick a resistor that would dissipate ≈ 250 mW when $5V$ is applied to it, and arrange your breadboard so that this resistor is *in contact* with

the LM35. Turn the ELVIS II on and watch the temperature go up... *Throw this resistor out when you're done!*

11. Save your results on a flash drive or network volume: we may do more with this vi later.

Light Sensor, in less detail

1. Build a light sensor with a PIN photodiode and the transimpedance amplifier circuit shown in figure 2. Don't be shy about asking for help from your instructor on this part; it uses things you'll learn later in the course. For now treat this op-amp circuit as magic and don't worry too much about how it works. Connect the output of this circuit to an unused AI connection on the ELVIS II board.

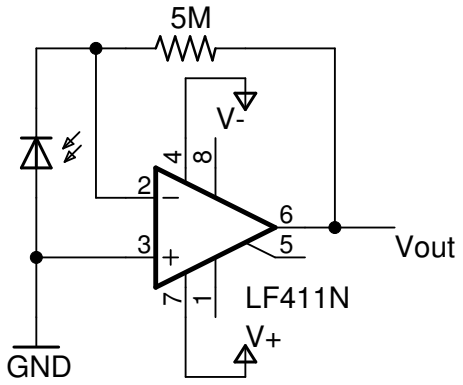

Figure 2: Transimpedance Amplifier. Use an LF411 or TL061 op-amp.

2. Open up a new .vi and this time put a waveform graph on the Front Panel.

3. On the Block Diagram, put in a DAQ Assistant block to read the voltage from your photodiode amplifier, just as you did before with the LM35. This time, though, have the DAQ Assistant measure 10,000 samples at a frequency of 10 kHz. Wire the output of the DAQ Assistant directly to the Waveform Graph. Also have the .vi save the measurement set to a text file for analysis by some other software package.[6] Run your program to test that everything is working so far.

[6] Python!

Check the text file also, making sure that the data is in a readable format.

4. Now that we have a light sensor, let's find something more interesting to measure than just waving your hands over the photodiode. Put in another DAQ Assistant block, but set this one to create a digital output. A continuous pulse train at about 10 Hz would be just about perfect. Use this digital output to drive a red LED (with current-limiting resistance in series) on the ELVIS II board, and arrange the layout of components so that the photodiode is looking at the LED. (See figure 3(A).) Verify that things work as expected: you should see a square wave on the waveform graph.

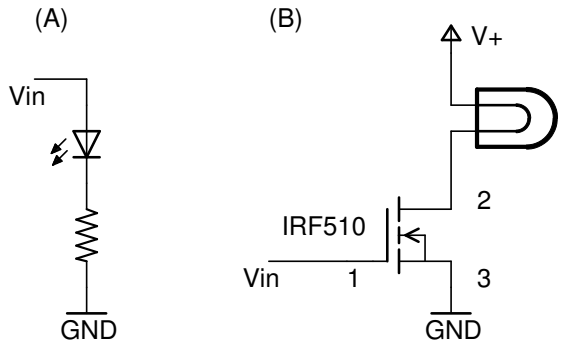

Figure 3: Using a logic signal V_{in} to drive (A) an LED and (B) a light bulb.

5. Now check what happens with an incandescent bulb. Replace the red LED with a flashlight bulb. The digital output on the ELVIS II board is not sufficient to supply the approximately 500 mA required by a flashlight bulb, so use an N-Channel Enhancement-Mode MOSFET as a switch. (See figure 3(B).)[7]

   Observe the light level from the flashlight bulb. How long does it take to turn this bulb on? Is the time to turn off the same as the time to turn on? Can you estimate the "half-life" of the light from the bulb? You may wish to answer the half-life question more precisely by

[7] Either an IRF510 or an FQP20N06 will work fine. They have the same pinout, which is indicated by the numbers in the figure.

analyzing your data with Python or Mathematica or some other tool with curve-fitting capabilities.

Writeup

In your lab report, be sure to include printouts of your front panel and block diagrams for both .vi's.

Clean up

Make sure your circuits are disassembled and parts are put back away, then clean and dust the work area around your station.