ECE 451

Automated Microwave Measurements Laboratory

# LabVIEW tutorial 1

The goal of this tutorial is to be able to write a simple virtual instrument (VI – similar to a program in other programming languages) that accepts the inputs (frequency, power level etc.) from the user, processes them, communicates with the measurement equipment, retrieves the measured raw data from the equipment, analyzes them and presents them to the user in a meaningful form. Using this program, a student should also be able to save the data into a file for later usage.

The concept of LabVIEW programming resembles that of a program flow chart. A box represents each instruction or I/O operation. Boxes are in turn connected with data flow paths (wires). One exceptionally useful aspect of LabVIEW is the "Show Context Help" under the "Help" category. This will create a window that will provide a description of any element that you run your mouse over. Anything in the tutorial that you do not understand can usually be explained with this.

**Before starting the program, all instruments should be turned on and connected through the appropriate bus.**

Open *National Instrument LabVIEW (*32-bit). Select *Blank VI.* You will see the blank **Front Panel** window and Block Diagram of your new program, as shown in Figure 1 . You can switch between **Front Panel** and **Block Diagram** windows by pressing *Ctrl-E.*

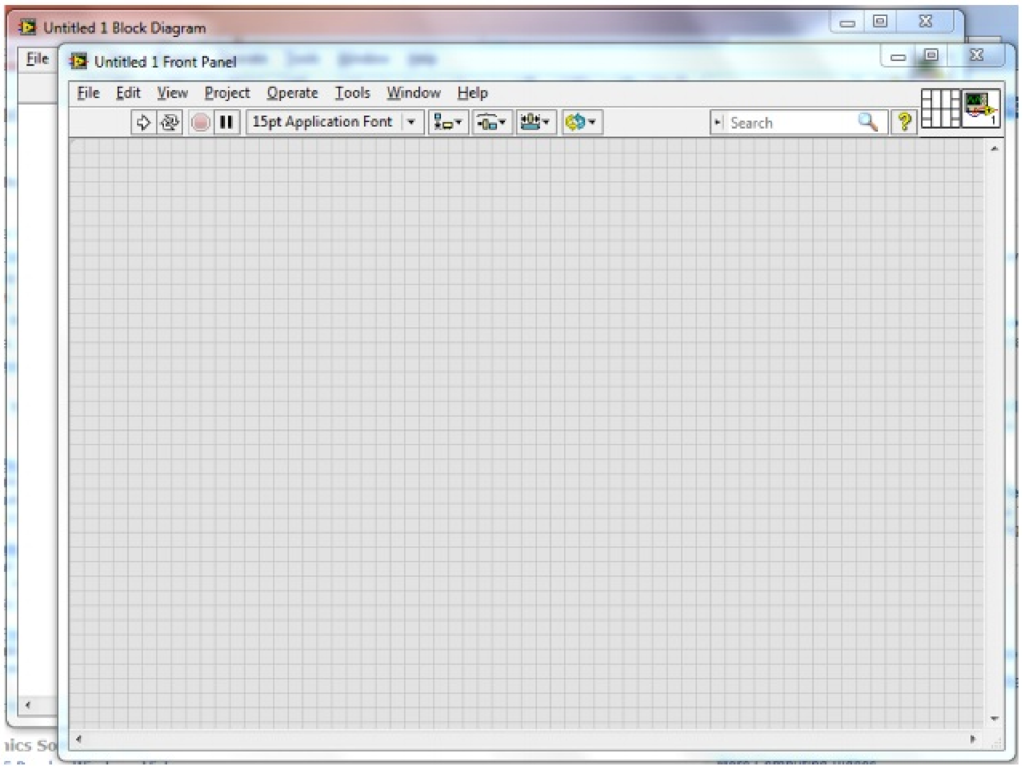


Figure 1: LabVIEW Front Panel and Block Diagram Windows

<u>*VISA Resource*</u> box must be created on the **Front Panel** to communicate with each instrument. To do that, right-click on any blank space of the **Front Panel** and select *Modern>I/O>VISA Resource.* A combo box titled "VISA Resource Name" will appear. Place it anywhere on the **Front Panel**. Change its title to represent a device

you want to communicate with, e.g. Source, or DVM (Digital Voltmeter). From the drop-down listbox, you will be able to select the desired GPIB address that corresponds to the actual device, as shown in Figure 2. LabVIEW automatically detects all the devices connected to GPIB bus, and offers them in the drop-down listbox. Choose the correct address based on which device you want to control. Refer to section  for information on getting the address of the devices we use in this tutorial.

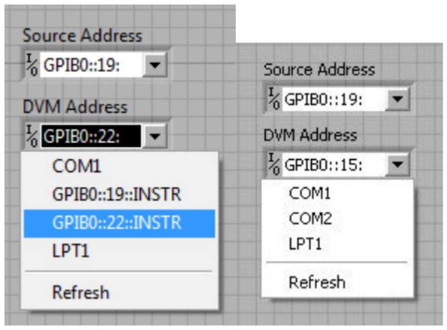


Figure 2: Selecting the GPIB address using VISA Resource box.

Simultaneously with creating the *VISA Resource* box on the **Front Panel**, an equivalently named box was created on the **Block Diagram**. This box will be used to provide the identifier of the device to all the other device control boxes on the **Block Diagram** (used to *Open* and *Close* communication, *Init* (initialize) the device, *Set parameters*, *Assert trigger* and *Read* values), as will be seen shortly. The quick way to copy something is to click on it, then holding "Ctrl," drag it away. This produces a second copy. Add another two *VISA Resource* boxes to the **Front Panel**, label them "DVM Address" and "Source Address" (like in Figure 2) and on the **Block Diagram** connect each to its own *Open* box (found in *Instrument I/O>VISA>VISA Advanced* menu).

The output of *VISA Resource* box should be connected to the "VISA Resource Name" input of the *Open* box (uppermost input on the left-hand side of the box). Next, add two *Write* boxes (found in *Instrument I/O>VISA submenu*), and connect the corresponding resource name inputs with outputs. You can also change the labels of *Open* and *Write* boxes to remind you of their functions, as shown in Figure 3

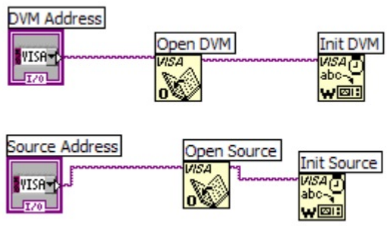
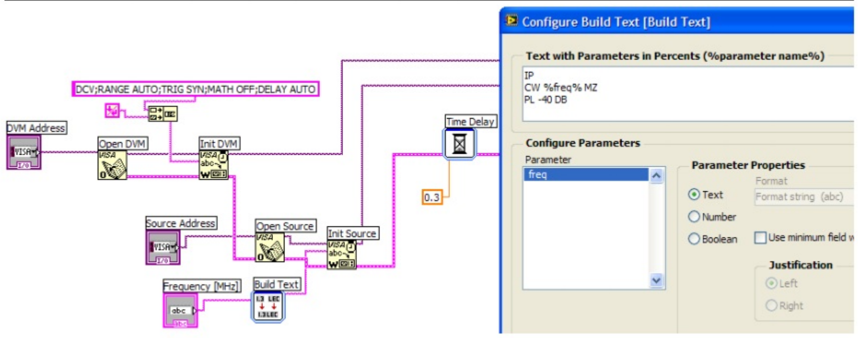


Figure 3: Blocks that are used to initialize the devices.

In order to actually initialize the devices, we need to send them appropriate strings. For initializing the DVM, create on the **Block Diagram** a *String Constant* (found in *Programming>String* palette), fill it with appropriate text for initializing the DVM (as shown in Figure 4), create an *End Of Line* constant (in the same palette), append it to the *String Constant* using the *Concatenate Strings* box (again in the *String* palette), and connect the appended string as an input to "write buffer" terminal of our "Init DVM" box (see Figure 4).



Text with Parameters in Percents (%parameter name%)

```
IP
CW %freq% MZ
PL -40 DB
```

Configure Parameters

Parameter: freq

Parameter Properties: Text [✓] Number Boolean; Format: Format string (abc); Use minimum field w; Justification: [✓ Left] Right

Figure 4: Completed block diagram for initializing the devices.

Initializing the source is a slightly different procedure, since we want to be able to define the sweeping frequency at runtime. To do that, we first create a <u>*String Control*</u> on the **Front Panel** (found in *Modern>String & Path* menu) and label it “Frequency”. Next, by double-clicking on it, we find its corresponding box on the **Block Diagram**; its output will be used as an input to the <u>*Build Text VI*</u> that we will also insert (from *Express>Output* menu). By double-clicking the <u>*Build Text*</u> icon (to conserve screen real estate as in Figure 4, right-click on the *Express VIs*, namely *Build Text* and *Time Delay*, select “View as Icon” option), we can define its functionality, as shown in Figure 4. We now connect the output of “Frequency” box to the “freq” input of <u>*Build Text VI*</u>; and its output, in turn, to the “write buffer” input of “Init Source” box.

We now want to add a time delay of, say, 300 ms, in order for our source to have time to stabilize its output. We do that by inserting the <u>*Time Delay VI*</u> (from *Express>Exec Control* menu), as in Figure 4, and creating the <u>*Time constant*</u> (a quick way to do that is to right-click its “Delay Time” terminal, and to select *Create>Constant* from the shortcut menu). Since we don’t want our time delay to be executed before we send the initialization data to the source, we need to be able to control the flow of our program. One handy way to do that is by using the “error out” and “error in” terminals. Error data flow is indicated by a thick pink wire in the figures. In the version you are using, these thick wires are yellow, black, and white instead.

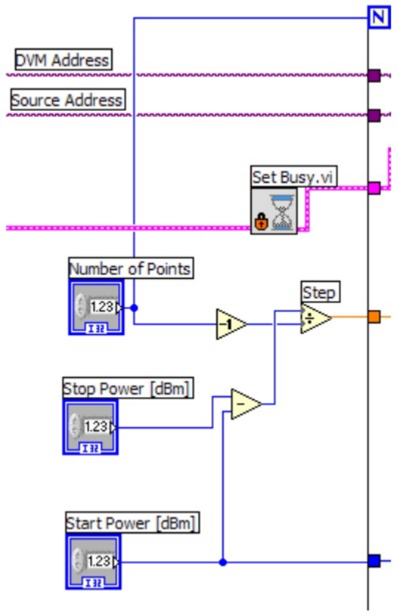
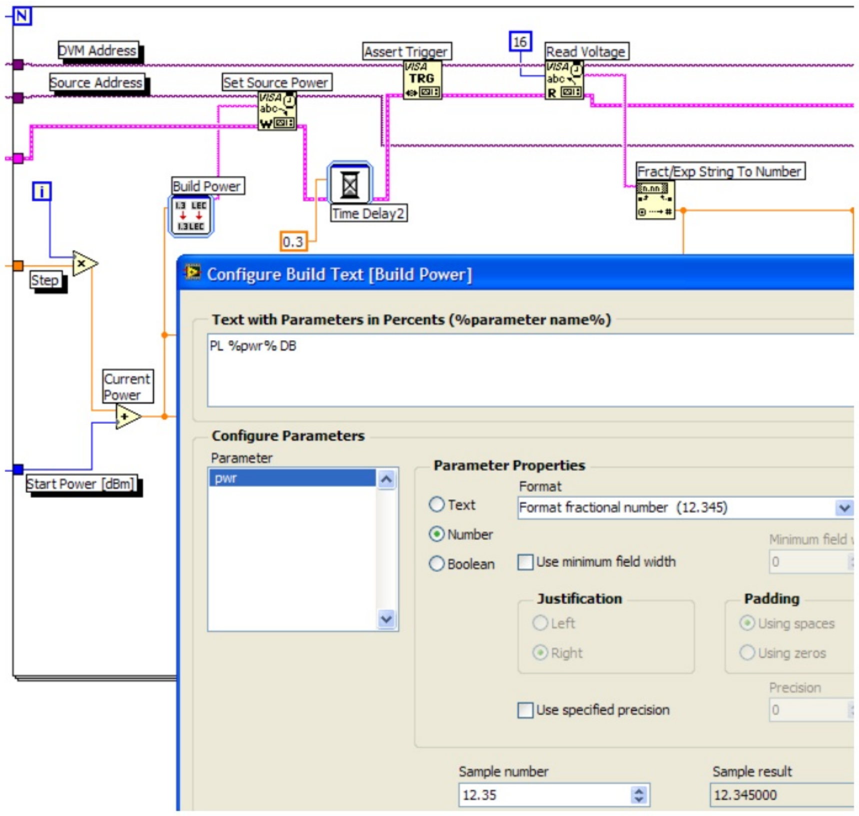


Figure 5: Inputs to the “for” loop.

Now we’re ready to move on to measuring the data. On the **Front Panel**, add three <u>*Numeric*</u> inputs (*Numeric>Numeric Control*), and label them as shown in Figure 5. These will serve as inputs to our “for” loop that will be doing the measurements. To have the data represented as integers (as opposed to default of double-precision real numbers), right-click each of the *Numeric* boxes on **Block Diagram**, select *Representation>I32* (actually, in this case, any integer type would do). Apply several math operations, as shown in Figure 5, to obtain the step size for our sweep. Add a large "for" box (*Programming>Structures>For Loop*), and connect "Start Power", "Step", "Number of Points", and the two instrument addresses to the left-hand side of the "for" box – those will serve as its inputs. A nice programming practice is to set the cursor to "busy" during measurements; this is done by inserting the *Set Busy* box (*Programming>Dialog & User Interface>Cursor*).

The "for" loop has two significant objects automatically created: "**N**" holds the total number of repeats, and "**i**" holds the current iteration of the loop (ranging from 0 to N-1). We use "**i**" to calculate the current value of power to be sent to the source. In each iteration of the "for" loop, we first build the string to be sent to the source, by using *Build Text VI* (labeled "Build Power") whose behavior is depicted in Figure 6.



Configure Build Text [Build Power]

Text with Parameters in Percents (%parameter name%)

PL %pwr% DB

Configure Parameters

Parameter: pwr

Parameter Properties

( ) Text [✓ Number] ( ) Boolean

Format: Format fractional number (12.345)

☐ Use minimum field width — Minimum field: 0

Justification: ( ) Left [✓ Right]

Padding: [✓ Using spaces] ( ) Using zeros

☐ Use specified precision — Precision: 0

Sample number: 12.35

Sample result: 12.345000

Figure 6: Measuring data in the "for" loop.

Next, we write the string to the source, wait 300 ms for the output to stabilize, trigger the DVM by using *Assert Trigger* (found in *Instrument I/O>VISA*), read up to 16 digits of voltage by using *VISA Read* (again found in *Instrument I/O>VISA*), and convert the string that was read to a number by using *Fract/Exp String To Number* (found in *Programming>String>String/Number Conversion*). This process is shown in Figure 6.

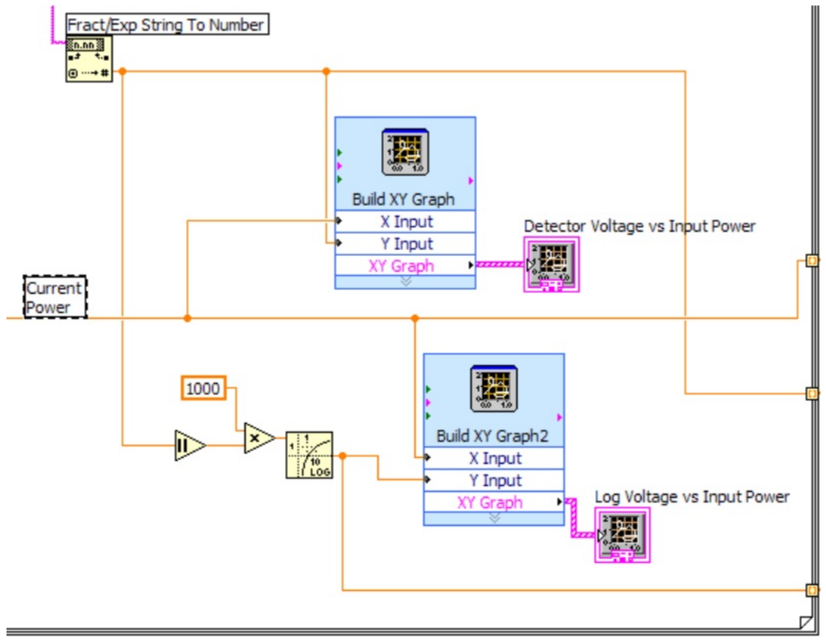


Figure 7: Plotting data in real time.

If we want to follow our measurements in real time, one possible solution is to insert plots of read data into the "for" loop. To do that, first insert two *Express XY Graph* objects (found in *Express>Graph Indicators* menu) to the **Front Panel**, as shown in Figure 7. Corresponding *Build XY Graph VIs* will automatically be added to the **Block Diagram**. Make sure (by double-clicking on them) that "Clear data on each call" is turned off, since we want graphs of complete measurements, not just single points. Both graphs should have the current value of power connected to their "X inputs". The linear graph will have just the read value of voltage as its "Y input", while we would need to calculate the log value of voltage (in units of dBm) as shown in Figure 7. *Log* is located in *Mathematics>Elementary>Exponential*.

After reading the data and finishing with "for" loop, we would first want to unset the *Busy* cursor, power down our source (e.g. to -75dBm) and close communication with the instruments, as depicted in Figure 8.

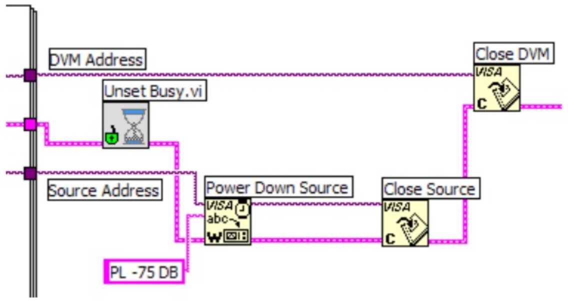


Figure 8: "Cleaning up" after measurement.

Finally, we want to save our measured data to a file, in order to be able to analyze it later. LabVIEW's "for" loop automatically collects all the data coming out of the loop, and creates arrays out of it; the process is called "auto-indexing" (if needed, this behavior could be changed by right-clicking the point where the data leaves the loop - in our case we would want to uncheck auto-indexing for the "DVM Address", "Source Address" and "Error" wires).

To make use of LabVIEW's full potential in working with files, one easy way is to convert the three arrays of data (power, voltage and logvolt) to *Waveform* data types, by using the *Build Waveform* box (found in *Programming>Waveform* palette), as illustrated in Figure 9. Next, we set the array names to represent the type of data measured, by using *Set Waveform Attribute* (found in *Programming>Waveform*) to set the "NI_ChannelName" attributes of the waveforms, also depicted in Figure 9.

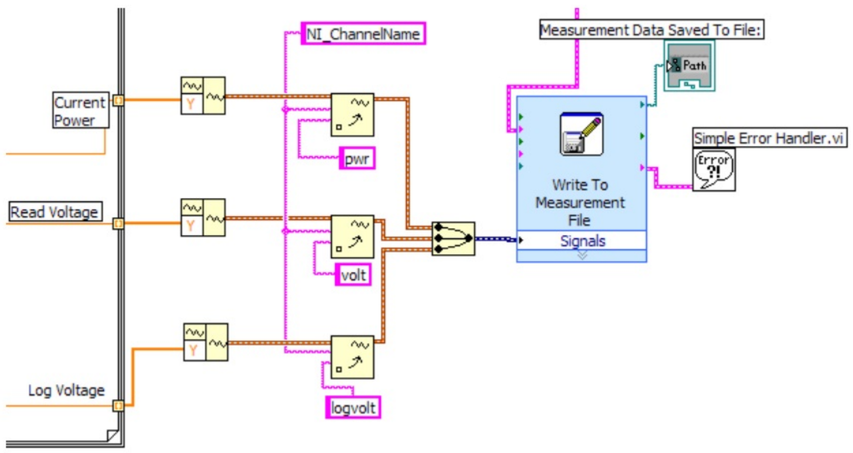
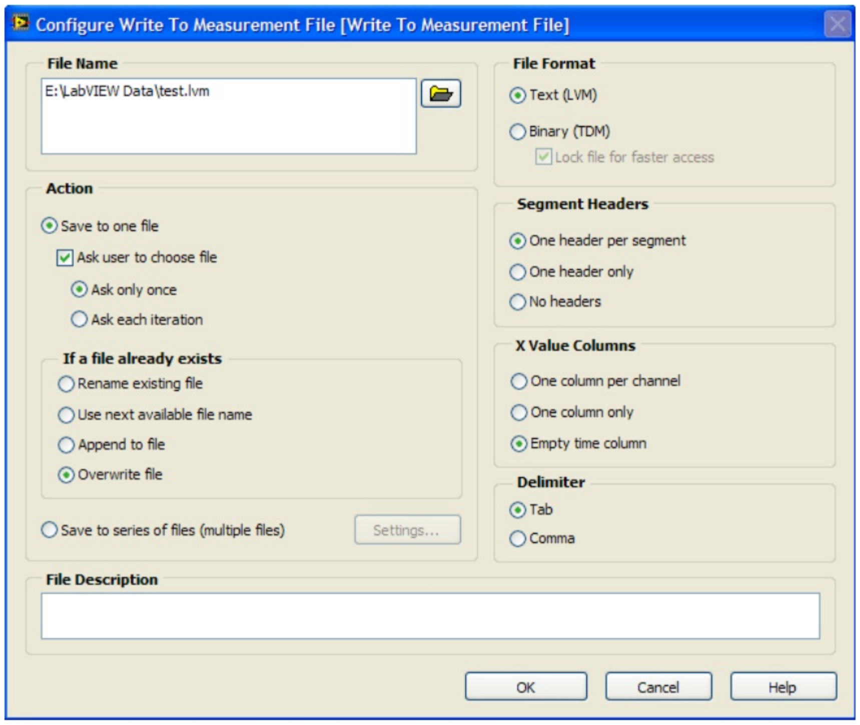


Figure 9: Formatting and saving data.

We then aggregate the three waveforms, by using the *Merge Signals* box (found in *Express>Signal Manipulation*), and feed them to the *Write To Measurement File VI* (found in *Express>Output*), the settings of which should be as in Figure 10.



Configure Write To Measurement File [Write To Measurement File]

**File Name**
E:\LabVIEW Data\test.lvm

**File Format**
[✓ Text (LVM)]
Binary (TDM)
[✓ Lock file for faster access]

**Action**
[✓ Save to one file]
[✓ Ask user to choose file]
[✓ Ask only once]
Ask each iteration

**If a file already exists**
Rename existing file
Use next available file name
Append to file
[✓ Overwrite file]

Save to series of files (multiple files)

**Segment Headers**
[✓ One header per segment]
One header only
No headers

**X Value Columns**
One column per channel
One column only
[✓ Empty time column]

**Delimiter**
[✓ Tab]
Comma

**File Description**

Figure 10: *Write to Measurement File* configuration.

This way, the data is saved in LabVIEW's proprietary text format[1] with the extension .lvm, viewable in a text editor, but not directly importable into other programs, such as Keysight ADS.

In order to communicate with ADS, we would need to write a separate subroutine for saving the data into e.g. CITIfile[2] format, which falls out of the scope of this course because of the relative complexity of that subroutine (compared to the simple *Write to Measurement File Express VI*). However, for the purpose of demonstrating how

[1] `http://www.ni.com/tutorial/4139/en/`
[2] `http://cp.literature.agilent.com/litweb/pdf/ads2004a/cktsim/ck0419.html`

to export data measured in LabVIEW to ADS, a complete subroutine for saving the data in CITIfile format is provided named ECE451_save2citifile. Before adding that sub-VI, place a conditional loop to enable the user to choose which file format to use which can be found in *Express>Exec Control>Case Structure* as shown in Figure 11.

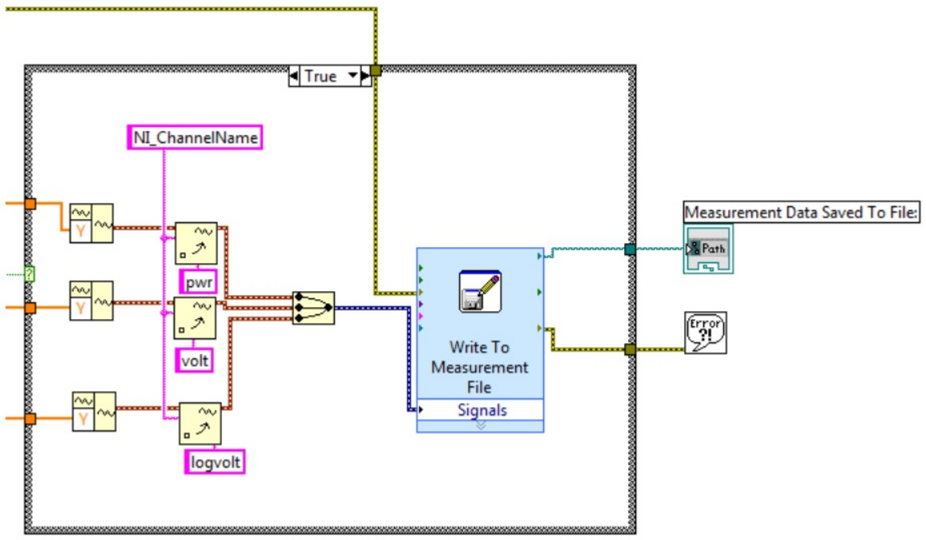


Figure 11: Inserting a conditional structure around the LVM generation code.

Now, switch to the false case by pressing one of the arrows to the right of the box that says “True”. Place the aforementioned subVI by right-clicking and selecting “Select a VI”, which then opens a dialogue box from which the ECE451_save2citifile VI can be found. Create a <u>*Boolean*</u> control to the conditional structure by right clicking the left wire end of the green question mark box on the left of the structure and selecting *Create>Control*. Wire the sub-VI as shown in Figure 12.

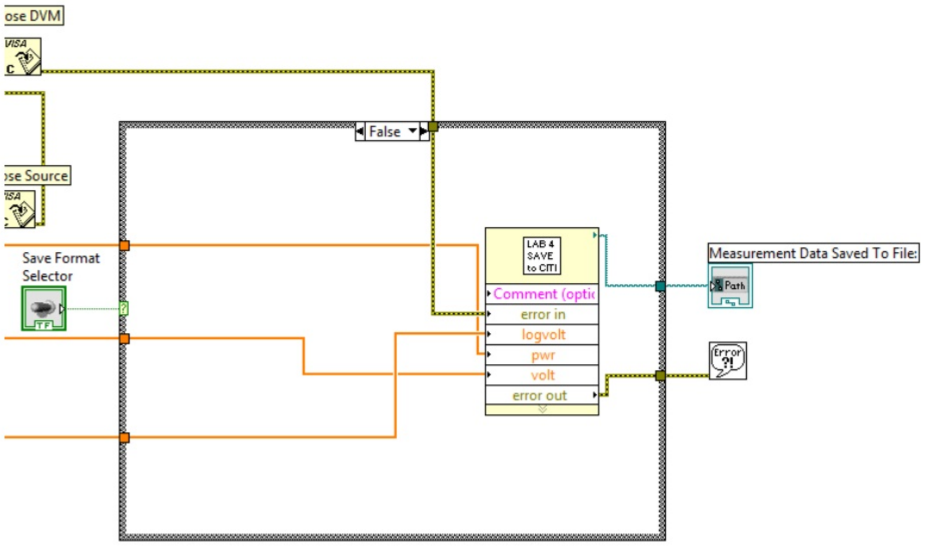
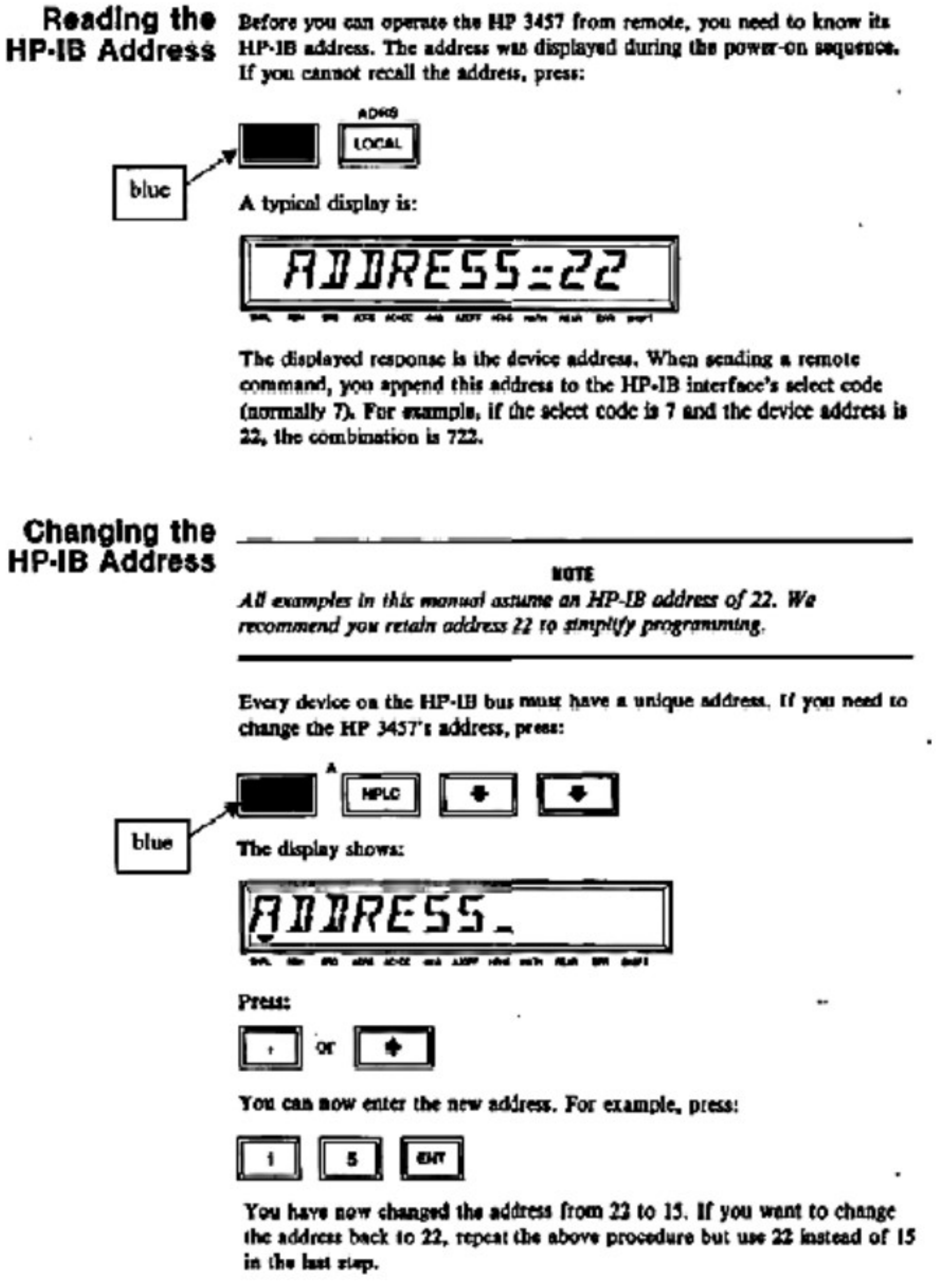


Figure 12: Inserting a conditional structure around the LVM generation code.

We finally insert a <u>*File Path Indicator*</u> to the **Front Panel** (from *Modern>String & Path*) to be able to observe the actual location of the saved file, as shown in Figure 12, and end the dataflow with <u>*Simple Error Handler*</u> (from *Programming>Dialog & User Interface*).

# Appendix – HP3457A Reading and Changing the HP-IB Address



## Reading the HP-IB Address

Before you can operate the HP 3457 from remote, you need to know its HP-IB address. The address was displayed during the power-on sequence. If you cannot recall the address, press:

blue

ADRS

LOCAL

A typical display is:

ADDRESS=22

The displayed response is the device address. When sending a remote command, you append this address to the HP-IB interface's select code (normally 7). For example, if the select code is 7 and the device address is 22, the combination is 722.

## Changing the HP-IB Address

**NOTE**

*All examples in this manual assume an HP-IB address of 22. We recommend you retain address 22 to simplify programming.*

Every device on the HP-IB bus must have a unique address. If you need to change the HP 3457's address, press:

blue

A

NPLC

The display shows:

ADDRESS_

Press:

↑ or ↓

You can now enter the new address. For example, press:

1 5 ENT

You have now changed the address from 22 to 15. If you want to change the address back to 22, repeat the above procedure but use 22 instead of 15 in the last step.