PHYS 5061 Lab 6
Programmable Instruments

Introduction

This lab introduces the computer control of bench lab instruments for data acquisition and uses a programmable digital multimeter as part of a measurement of the voltage-current characteristic curves of diodes.

Computer-control of many electronic instruments such as multimeters, digital oscilloscopes, function generators and many other more specialized instruments is useful and commonly available. These instruments often provide capabilities beyond the data acquisition capabilities of multi-purpose DAQ modules that plug into a computer or into a USB port, as you have used so far. Dedicated bench instruments usually offer greater flexibility, faster speed, or higher resolution than most DAQ modules. Consider that the NI-USB-6211 can only perform A/D conversions at rates up to 250,000 samples per second, while a digital oscilloscope may be able to capture signals at rates of 100 million samples per second or more, although with some sacrifice in resolution. And a function generator can provide faster, higher quality waveforms than the digital-analog output capabilities of the 6211. Each approach has its advantages and limitations: a suite of benchtop programmable scope, generator and multimeter will easily cost several thousands of dollars compared to probably less than a thousand dollars for a reasonably capable PC-based DAQ module. A sophisticated experiment in a lab is likely to use both kinds of capabilities.

For many years the General Purpose Interface Bus, GPIB, has been a widely used to interface an array of standard laboratory instruments to personal computers. This scheme is also referred to as IEEE-488 (after the standards set forth by the IEEE). An enormous variety of common (and not-so-common) laboratory instruments have been available equipped with a GPIB interface to allow computer control of their operation. Lots of instruments with GPIB interfaces are around, and it still can be found on new instruments. But alternatives to GPIB to connect instruments to a computer are replacing GPIB. Instruments now often have USB connections or even Ethernet connections available, so an instrument can be placed on the Internet and potentially supply data from very remote locations.

The original GPIB standards went through several revisions. Among other improvements over time was the evolution of a common set of commands for various classes of instruments. Originally each manufacturer defined their own commands to program an instrument, e.g. to set a digital multimeter to measure AC voltage or DC current. Replacing an instrument with a newer one, particularly if from a different manufacturer, generally meant re-writing the software to match the new commands. With other advances, most recently manufactured instruments have

adopted a set of Standard Commands for Programmable Instruments (SCPI) that allows easier changes to instruments in an experiment. The resulting command set has a more complicated hierarchy in exchange for better interoperability.

With the rise of new interfaces (USB, Ethernet) alongside GPIB and the consistent command structure, high level software like LabVIEW can hide the details of exactly how communications between instrument and computer happen. LabVIEW uses a set of protocols and routines called VISA (Virtual Instrument Standard Architecture) to handle low level details about communications using USB, GPIB, or Ethernet as needed between the PC and the instrument. As LabVIEW programmers we rarely need to think much about those details and can focus on designing the commands we want to send and reading the data we receive from the instrument without knowing exactly how it happens for the different interfaces.

Instruments in our lab contain a mix of these possibilities. We will make use of the standard GPIB interface for our preliminary work. Sometimes the PC side of the GPIB interface is on a card inside the computer. Our lab computers now use a USB-to-GPIB converter device that allows a PC to talk to several GPIB instruments through a single USB port.

The connections to instruments are made using standardized cables (the "bus" in GPIB), and the cables can be stacked piggy-back style to connect multiple instruments together, either in a series fashion, where the cable runs from the PC to the first instrument, a second cable connects the first and second instruments, etc., or in a "star" or parallel fashion, where a separate cable runs from the controller card to each instrument. The bus is bi-directional: information flows both from computer to instrument and from instrument to computer, and even between instruments.

Instruments on the bus are identified by their (primary) GPIB address, a number that is in the range of 0-30 (0 is usually reserved for the system controller and 31 is dedicated for special purposes) that can be set either by dip-switches on older instruments or from the front panel controls. Devices on the bus can act as either talkers, i.e. they put information or data onto the bus for use by other devices, or as listeners - devices that accept data or instructions from other instruments or the controller. For most of our purposes the flow of information will be between a specific instrument and the PC. Our LabVIEW program will give instructions to the instrument as to what measurements to carry out (PC as talker and instrument as listener) and then the instrument will pass the results of the measurement to the computer and LabVIEW for further processing (PC=listener, instrument=talker).

This introduction to the GPIB will focus on one instrument for starters: the Keithley 199 digital multimeter (DMM). It uses a pre-SCPI command set - which means the commands are simpler to learn and program than SCPI commands. GPIB commands to instruments come in two forms. Device independent commands are instructions that all devices respond to in a uniform fashion and have predictable

consequences, such as "interface clear" "enable remote operation," or "clear all devices" (to their default power-up state). In contrast, device dependent commands are sent as character strings over data lines in the cable, with appropriate manipulation of the control lines to assure reception by the intended receiving instrument. These command strings are specific to the instrument in use and may do such things as set the DMM for measuring voltages on the 30V scale or measure current with auto-ranging of scales. The repertoire of device dependent commands will vary from instrument to instrument.

Getting started with a GPIB instrument

Chapter 14 of Essick's *Hands-On Introduction to LabVIEW* provides an introduction to using programmable instruments in LabVIEW with the VISA VI's. Much of this chapter is based on a specific fully IEEE-488.2 and SCPI compliant multimeter. Your Keithley 199 multimeter is neither of these, but loosely following Essick's early explorations will get you started. **BUT be aware that the commands Essick uses will not work for your K199 multimeter.** After some preliminary exploration of instrument control, your intermediate goal will be to build a couple very simple VI's that: (1) allow the DMM to be programmed to perform a particular function and (2) read the results of a measurement for further processing or graphing in LabVIEW. The final destination is to put it to work with other DAQ tools to build a VI to make some measurements on a diode.

Read through sections 14.1-6 of Essick. Pay particular attention the steps in using the VISA VI's to communicate with an instrument. Section 14.4 describes common commands all recent instruments understand, but not the Keithley 199!! So look at them briefly, but we won't use them in this lab apart from a chance to see they don't work as described. The same goes for Sec. 14.5 on status reporting. The Keithley has its own scheme for reporting its status, and we will do our best to avoid working with it for now. Sections 14.6-7 describe the SCPI scheme for instrument specific commands. Don't dwell on them. This is not useful to us at this point. It's a fairly complicated tree to form these instructions. We will use the simpler scheme specific to the Keithley 199. Its commands are summarized on just a couple pages, and we will need very few of them for our work.

Make sure that your K199 DMM is connected to your computer through a USB port on the PC and a USB-GPIB adpater. Turn the meter on. Typically during power-up it will reveal its GPIB address, which you should note. Follow Essick's Sec. 14.8 to use the NI Measurement and Automation Explorer (NI-MAX) to see if your DMM appears among the recognized GPIB instruments. (It may not show up by name, but more likely just as a device with the GPIB address that you observed during power-up. Follow through on the process in NI-MAX to send the *IDN? query (identify yourself!) to the meter. The DMM is too old to understand this

command, but, if things are working, you'll get a response that is the most recent measurement completed by the instrument. This comes as a string of characters, with a prefix telling what was measured followed by more characters giving the numerical value. If this is so, then basic communications are functioning. Yay!

Still in NI-MAX, try the "Open VISA Test Panel" tab. This provides the ability to send commands and read responses from the instrument interactively. With the Write (VISA Write) button or tab, try composing the command "F1R1X". NI-MAX provides a default linefeed character as a terminator which you can leave in. (A quick look at the programming guide for the K199 will show that F1 = AC Voltage, R1 = 300 V range. The X is the eXecute command for the K199 telling it to perform the operation. Enter this string of commands and then execute the write operation. With the Read button (or go the the viRead tab in some NI-MAX), set the character count to 40, and execute to read a measurement or two back from the meter. (If you ask for too few characters, you'll get a truncated response and some characters will be left in the K199 output buffer, perhaps recovered on the next read operation, or perhaps lost forever.) Look at the summary of commands for the K199 from its manual and try a few. Hook up some test leads and try to measure the resistance of a few resistors. If communications get hung up, the Clear (VISA Clear) button or tab seems to work well at putting the instrument back in the default state.

Simple VI's

Begin by following in broad strokes Essick's first VI-building exercise in sec. 14.9. (Don't attempt to follow sec. 14.10 and beyond. We'll part company there.) Follow the use of the basic VisaOpen, VisaWrite, VisaRead, VisaClose VI's for instrument I/O Essick describes. Instead of a non-functioning "*IDN?" query to the instrument, use the VisaWrite VI to send a more successful programming command, and then read back a result with the VisaRead. Try it out several times to see if it's durable enough to be reliable under repeated operations. (Mysterious bus errors are not uncommon; we want to avoid anything too fancy in the way of error handling, so our VI's will be very simple and robust–we hope.)

Now save this VI and then use it as a basis for two new and smaller simpler VI's to be saved under new names.

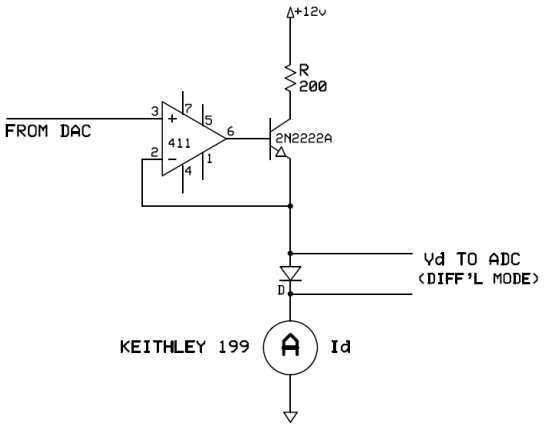
Name the first of these new VI's K199write.vi and modify it to carry out only the write operation to send a command to the instrument. (You'll need to keep the Open, Write, Close VISA VI's). Make the command string a control (if it isn't already) and create a reusable VI with an nice icon and wire up terminals on the icon for inputs (the visa resource, command string, and visa error in cluster) and outputs (at least the visa error out cluster).

Name the second VI K199read.vi and make it a self-contained and re-usable VI to read a measurement from the K199. Again, design an icon and configure all the needed input and output terminals so that you can easily drop it into a larger VI as a functioning sub-VI. There is one more important step in taking the K199's response and making it useful: the response is a *character string* and is not yet useful in calculations or graphs. We need to strip off the extraneous characters, isolate that part of the string that is the numerical result and convert it into an internal format (e.g. real double) the computer uses to represent numbers, not text. You'll find a useful VI for this among the palette of "string to number" functions (see p. 574-5 in Essick). Your finished K199read.vi should include an output terminal that provides the resulting number (providing the character string at another output terminal is optional, but potentially useful in debugging).

Thoroughly test your VI's - particularly the K199read.vi. You might drop it in a while-loop in a new VI and exercise it a bit and find any speed limitations. DMM's are not intended for very fast operation - this one trades speed for better resolution, so you might find your VI breaks if you ask for results too frequently.

Putting things together: current-voltage characteristics of a diode

Use LabVIEW and the Keithley 199 as the DC ammeter in the circuit shown to carry out a measurement of the current-voltage characteristics of a diode and plot $I(V)$. This will require you to: use your new VI's to work the meter as an ammeter to measure the current; use the ADC capabilities of the USB-6211 to measure the voltage drop across the diode and also draw on your prior experience with analog outputs (AO0) from the USB-6211 DAQ gadget to provide a driving voltage. Read through everything first to have a rough overview of the task before proceeding. Be sure you think about how this circuit works before diving in, randomly applying voltages from the DAC to drive the circuit.



In the circuit shown the op-amp is used to apply a voltage from one of the DAC channels (say AO0) of the USB-6211) to the diode. Since both the DAC analog outputs of the USB-6211 and most op-amps have limited output current, the transistor serves a current amplifier.

Use one of the analog outputs of the USB-6211 DAQ gadget to provide a desired V_{dac} to the op-amp (use an LF411). This op-amp's job is to keep the 2N2222A transistor turned on just enough to put the anode end of the diode at the same voltage as coming from the computer's DAC. (Remember the golden rules for op-amps.) The true voltage drop across the diode will be measured with one of the ADC analog inputs of the USB-6211 operating in differential mode. This means you'll need to configure the corresponding DAQ Assistant VI for *differential inputs* (previously we used NRSE) and wire up the correct *pair* of input lines, e.g ai0 and ai8.

Your task is to make a nicely detailed measurement of the diode's current as a function of voltage and generate a good plot in the end. Do this FIRST for a typical signal diode (1N914 or 1N4148 for example). You'll want to step through a range of applied voltages V_{dac}, and for each one measure the voltage drop across the diode and the current through the diode. Get a detailed $I(V)$ curve, not just a handful of points. You may want to think about the appropriate gain or voltage range to use for the analog input DAQ Assistant to get the best resolution. You already know

that in normal forward biased operation a diode exhibits a voltage drop roughly between 0.5 to 0.7 V and therefore you're not likely to to be needing the full ± 10 V range of the ADC, and a smaller ± 1 V range might be suitable. Similarly it would probably be foolish, maybe hazardous to the diode's survival, to program the DAC to apply voltages much bigger than this.

It is a wise practice to be sure you set the DAC voltage back to zero at the end of your measurements as a final step in your VI.

Caveats: The 200Ω resistor in particular is present to limit currents flowing through, and power dissipated by, the transistor and diode. A safer choice might start with a higher value. If things seem to run hot or out of control, this would be a good change to make. But you also need to be aware that at some point this may be the limiting factor in what current you measure through the diode. It's probably not a good idea to have 100 mA flowing through a signal diode.

Measure $I(V)$ with the diode forward biased. Your LabVIEW VI should save the data you have gathered to a file for later use. Then flip the diode around in the circuit and try to measure the negative (reverse biased) side of the $I(V)$. You might want to extend your voltage range further on the reverse biased side. These reverse currents, remember, are supposed to be very, very small - very possibly too small for the Keithley 199 to resolve.

Once you have completed measurements on a signal diode, try repeating the measurements with a light emitting diode (LED). For the LED, you'll need to re-think the range of voltages you'll try - a red LED will need larger forward biasing to turn on and emit light. Get good data for this diode, too.

Make nice plot(s) of the data and discuss the differences in the curves of the two diodes you measure.

Keep good notes in your lab notebook while you work through the trials and tribulations of getting everything to work. Include printouts of the VI's, document how they work, and the resulting graphs.

Related readings on GPIB in addition to Essick's chapter 14:
Dunlap, *Experimental Methods* pp. 114-119
Preston and Dietz, *The Art of Experimental Physics*, Appendix C.