NAWCWD TM 8415

A Tutorial on the use of GPIB and Spreadsheet Software Macros for Data Acquisition in the Evaluation of Photovoltaic Device Performance

by

Scott K. Johnson
for the
Chemistry and Materials Division

JULY 2003

NAVAIR WEAPONS DIVISION
China Lake, CA 93555-6100

FOREWORD

This report was written by an Engineer and Sciences Development Program (ESDP) employee during his tour of the Chemistry and Materials Division of the NAVAIR WD Research Department.

This document presents a straightforward process for automating data acquisition using a general purpose interface board (GPIB) and software with GPIB-compatible devices (e.g., multimeters, electronic loads, etc.), and which uses Microsoft Excel to capture the resultant data. Using the equipment and processes described in this document one can centrally control several devices automatically for the purpose of data acquisition and/or experiment control. This method is a cheap alternative to expensive software and hardware, such as LabVIEW™ and analog-to-digital (A/D) data acquisition boards.

This document was reviewed for technical accuracy by Joe Roberts and Mike Seltzer.

R. A. NISSAN, Head
Chemistry and Materials Division
10 July 2003

NAWCWD TM 8415, published by the Technical Information Division, 15 copies.

CONTENTS

This page intentionally left blank.

ABSTRACT

We present a lightweight and straightforward way to automate data acquisition (DAQ). This approach uses a general purpose interface board (GPIB) and software with GPIB-compatible devices (i.e., multimeters, electronic loads, etc) and Microsoft Excel to capture data. (Devices with a GPIB hardware interface are equivalent to the HPIB and IEEE-488 hardware interfaces.) One can centrally control several devices automatically for the purpose of data acquisition and/or experiment control. This method is a cheap alternative to expensive software and hardware (i.e., LabVIEW™, A/D data acquisition boards, etc.), although certain considerations should be taken into account when considering your hardware and software requirements (see Appendix A-1). The following works well for simple experiments and we give an example that generates IV (i.e., current-to-voltage) curve data for a solar cell.

INTRODUCTION

The following is a step-by-step guide to use GPIB-compatible devices, the GPIB interface software, and Microsoft™ Excel to gather data and control experiments automatically. We assume the GPIB card and software has already been installed on the machine you are using. (Our system has an x86 CPU, Microsoft Windows 98, and Microsoft Office 98.)

Two examples will be discussed: first, a simple example that collects voltage readings from a multimeter, and then the *IV Curve* example.

SIMPLE EXAMPLE

For this example a single GPIB-compatible multimeter is attached to the GPIB board with a GPIB cable.

GPIB DEVICE ADDRESSES

Most GPIB-compatible devices allow you to set the GPIB address for that device manually. Usually this is done with three or four binary switches located near the GPIB port for that device. These switches are the base 2 representation of the device address. For example, if your switch was set to `0110`, the address for that device would be `6`.

Setting Up Excel For DAQ Through GPIB

Next we open up Excel and the Visual Basic (VB) editor; go to *Tools* → *Macro* → *VB Editor*. Once in the VB editor it is necessary to import the GPIB VB interface files for the GPIB board. With our National Instruments GPIB board these files were located in the National Instruments directory (C:\Program Files\National Instruments\GPIB\NI488\LangInt\VBasic\ Ver5). The two files you will need to import are VBIB32.bas and NIGLOBAL.bas. To do this, in the VB editor go to *File* → *Import File...* and select the files you want to import.

Then add a *Module* and *Procedure* for the Basic code you will be writing. To do this go to *Insert* → *Module* and then again to *Insert* → *Procedure....* You will be asked to name them. This will add a blank subroutine to the *Module* you just created.

GPIB VB Interface

National Instruments provides a low level interface, or driver, for their GPIB boards. There is also a *C* interface provided.

We use a small subset of the methods defined in VBIB32.bas. They are as follows:

`ibdev`	Opens an unused device
`ibclr`	Clears a specific device
`ibin`	Checks for presents of device
`ibloc`	Put device in *local operation mode*
`ibonl`	Place device online or offline
`ibrd`	Reads data from device
`ibwrt`	Writes data to device

More information on the parameters and details of these methods will follow. The *status word* `ibsta` will also be used frequently (see Appendix A-4). Each bit of the status word has information about the current system status, see GPIB manual (Reference 2), page A-1 for more information.

VB Script

Now we are ready to write a Basic script that takes voltage measurements from a multimeter. (A Fluke 8840A multimeter is used for these examples.) Excel is used as both an input and output mechanism for data collection. The Basic code reads the desired number of data points from a specified cell on the spreadsheet and then writes those data points to a different specified location. The following refers to the file *SIMPLE.bas,* which is in Appendix A-3. Here is a brief step-by-step explanation of the Basic code contained in *SIMPLE.bas.*

First, set the input/output locations for the Excel spreadsheet, see below.

```
Set DATA_POINTS_LOC = Range("A1")
Set DATA_COLLECT_LOC = Range("A2")
```

In the Excel spreadsheet, cell "A1" will be where the user enters how many data points they want to be taken. Likewise, cell "A2" will be the starting point of the data output. (See VB documentation, manuals, and references for more information on Graphical User Interfaces (GUIs) and other VB features.)

Next we bring the multimeter online with the following code:

```
Call ibdev(0, 6, 0, 15, 1, 0, DEV_ID%)
```

The call to `ibdev`, which is defined in the VBIB32.bas file, takes several input parameters and one output parameter. The first parameter is the GPIB board address, 0, in this case. (Reference 2, the GPIB User Manual, has more information on this topic.) The next two arguments are the *primary* and *secondary* addresses for the GPIB-compatible device that you are trying to bring online, in this case, the multimeter. The Fluke multimeter has a GPIB address that has been set to 6. The fourth argument is the device timeout in seconds, which we have set to 15. Different devices will have different timing issues and possibly many different solutions; this argument may help in some of these situations. (The GPIB manual also has many suggests for resolving timing problems.) The next two arguments are binary flags for different operating modes. The first is the *END* message toggle and the second is the *end of string (EOS)* toggle. The GPIB manual also contains more detail about these modes. The above configuration seemed to work fine for our purposes. The last argument is the output argument, passed by reference (see VBIB32.bas), which assigns an integer to `DEV_ID`, allowing for future writes and reads to specific devices. Reminder: VB allows the following characters to specify specific types in their variable identifiers: `% - Integer, $ - String, # - Double, & - Long`, and others.

Once the multimeter is online it can accept *read* and *write* commands. The `ibwrt` call writes the command string, `"F1 S1 TO"` to the multimeter:

```
Call ibwrt(DEV_ID%, "F1 S1 TO")
```

This command does three things: `F1` sets the multimeter to *DC Volt* mode, `S1` sets the *Reading Rate* to medium, and `TO` sets *Continuous Trigger* mode. These commands are unique to the Fluke multimeter, although an HP multimeter we used seemed to have the same command set. Your instruction manual contains a complete command set for your device.

Next we gather the input from our specified input location on the Excel sheet.

```
DATA_POINTS% = DATA_POINTS_LOC.value
```

Now we are ready to start reading data from the multimeter. [*The code segment below has been numbered for easy reference.*]

```
1  Readbuf$ = Space$(20)
2  Set cel = DATA_COLLECT_LOC
3  I% = 1
4  While I% < DATA_POINTS% + 1
5  Call ibrd(DEV_ID%, Readbuf$)
6  rd$ = Left(Readbuf$, (ibcnt% - 2))
7  cel(I%, 1).value = CDbl(rd$)
8  I% = I% + 1
9  Wend
```

Lines **1** through **3** initialize some variables before starting to loop. Line **5** makes the interface call to `ibrd`, which takes two arguments: the first is the device identifier of the device from which you wish to read, and second is the buffer that will hold the data being read. Line **6** takes the first `ibcnt - 2` characters in the buffer `Readbuf`. The variable `ibcnt` is defined with the *GPIB VB Interface,* and is the size of the buffer that you have just read. In certain devices, the last character in these strings is an unreadable marker; you will need to get rid of it if you want to cast that string into a *double* or *integer* type. This unreadable marker can differ from device to device and should be kept in mind if you are getting formatting errors after you have read data from a device. Lines **7** and **8** output the data to the Excel spreadsheet and increment the loop variable.

Now that we are done taking data we can return the device to *local* mode operation and take the device offline.

```
Call ibloc(DEV_ID)
Call ibonl(DEV_ID, 0)
```

The *GPIB* software has good examples for both the *C* and *VB* interfaces, for many different applications. The *GPIB User Manual* (Reference 2) is also a good source of documentation.

Running Your Basic Program

To run the Basic code go to back to the Excel spreadsheet and go to *Tools* → *Macro* → *Macros....* A window will pop up; the name you gave your procedure will be listed in this window. Highlight the procedure you want to run and then press the *Run* button.

Error Checking and the User Library

It is highly recommended to do some error checking when making calls that interface the *GPIB* board. We wrote some procedures that include error checking, which can be found in Appendix A-4. The methods are as follows, and will be covered in more detail as we encounter them:

```
Open_Dev    Opens an unused device
Close_Dev   Closes a device
Write_Dev   Writes data to device
Read_Dev    Reads data from device
```

IV CURVE

One way to evaluate the performance of a solar cell is with an *IV Curve,* or *current-to-voltage* curve. These graphs plot the voltage versus the current as the load, or resistance, changes. This data can be useful in a couple of different ways. It gives both the short circuit current, I_{sc}, and the open circuit voltage, V_{oc}. The I_{sc} is when resistance is as close to nothing as possible, and similarly the V_{oc} is when the resistance is very great. The product of these values is the theoretical power maximum. The *IV Curve* can also be used to calculate the *fill factor, FF,* defined below, where P^o_{max} is the theoretical power maximum, and P_{max} is the maximum power of the device.

$$FF = \frac{P_{max}}{P^o_{max}} \tag{1}$$

There are other measures as well, such as *spectral resposivity* and *quantum efficiency,* which can be found in the solar cell literature but will not be discussed here (Reference 4). Figure 1 is an example of an *IV Curve* of a commercial solar cell.

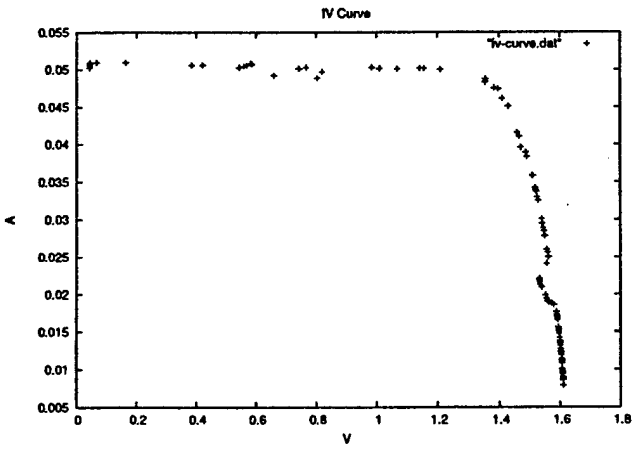

FIGURE 1. IV Curve.

IV CURVE HARDWARE

To generate an *IV Curve* one must have the following circuit (Figure 2).

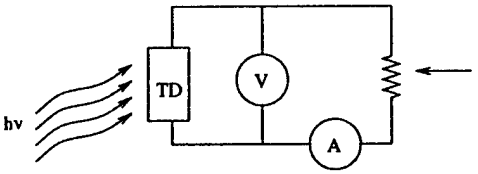

FIGURE 2. IV Curve Generation Circuit. *TD* refers to *Test Device.*

The *TD* is in parallel with a voltmeter as well as an ammeter and an adjustable load, which are in series.

One way to achieve the adjustable load is with a programmable electronic load. This is nice for automation, too. We used an *HP 6050A Electronic Load* (Reference 3). Figure 3 shows the setup with the electronic load.

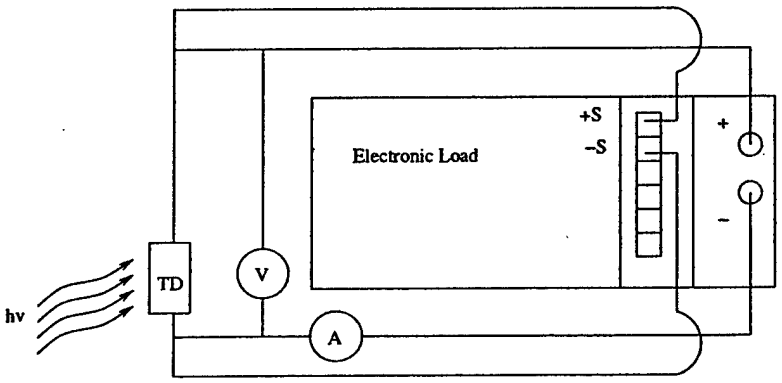

FIGURE 3. IV Curve Generation Circuit w/ HP Electronic Load.

IV CURVE SOFTWARE

There are many similarities between the *IV Curve* example and the 'simple' example, we covered previously. The *IV Curve* example follows the same general algorithm, setup I/0, open and initialize devices, gather input, collect data, and close devices, except that there are more devices, parameters, and variables in this example. The routines in the USER_LIB.bas are used in the *IV Curve* example and variables are declared explicitly for clarity (see Appendix A-4 and A-5).

Once again, the I/O locations are defined, and now we have two input parameters: the number of data points you want to collect, DATA_POINTS; and the maximum resistance that you want to use when plotting the *IV Curve,* MAX_RESISTANCE. The reason for the maximum resistance parameter will be discussed in greater detail later. Here the inputs are found in column "B"; this allows you to label the Excel spreadsheet for easy reference of input parameters.

```
Set DATA_POINTS_LOC = Range("B1")
Set MAX_RESISTANCE_LOC = Range("B2")
Set DATA_COLLECT_LOC = Range("C1")
```

Next the devices are brought online and initialized. Now that there are three remote devices being used, it is necessary to have three device identifiers. We are now using the call to Open_Dev, which takes all the same parameters as the *GPIB Interface* call, minus the output parameter, which is returned in this case. The only parameter that changes in each call is the second one, which is the *GPIB address*. This will change depending on the addresses of your devices.

```
MULTI_METER_1_DEV_ID = Open_Dev(0, 1, 0, 15, 1, 0)
MULTI_METER_2_DEV_ID = Open_Dev(0, 6, 0, 15, 1, 0)
ELEC_LOAD_DEV_ID = Open_Dev(0, 5, 0, 15, 1, 0)

Call Write_Dev(MULTI_METER_1_DEV_ID, "F1 S1 TO")
Call Write_Dev(MULTI_METER_2_DEV_ID, F5 S1 TO")
Call Write_Dev(ELEC_LOAD_DEV_ID, "MODE:RES")
Call Write_Dev(ELEC_LOAD_DEV_ID, "INPUT ON")
```

After the devices are open, each device is set to the state desired for data collection. There are two Fluke multimeters in use. MULTI_METER_1 will be the voltmeter, and MULTI_METER_2, the ammeter. Recall, for the Fluke multimeters "F1" is the command to set the multimeter to measure DC Volts, and "F5", DC Amperes. For the *HP Electronic Load* two commands are sent, one sets the device mode to *Constant Resistance* and the next turns "on" the input.

Next, the input parameters are retrieved from the Excel spreadsheet, along with the calculation of the resistance increment interval, RESISTANCE_INC, and some other initialization. Here the resistance increment is calculated by dividing the maximum resistance by the data points you want to sample. This gives a consistent increase in resistance for each data point. By specifying the MAX_RESISTANCE it allows you to tune what range of resistances you want to collect. For this particular electronic load module, the maximum specified resistance is 20kΩ, which is used to calculate the V_{oc}. For your solar cell, however, you may only need 200Ω to approach the V_{oc}. For example, let DATA_POINTS equal 10, and MAX_RESISTANCE equal 20kΩ. Then RESISTANCE_INC will be 2kΩ. If your solar cell approaches the V_{oc} around 200Ω no *IV Curve* will be recorded because all of the data is taken at the V_{oc}. If the MAX_RESISTANCE is 200Ω, then the RESISTANCE_INC will be 20Ω allowing for points to be recorded from 1Ω to 200Ω at 20Ω intervals, which will generate meaningful data. The starting resistance is also initialized to 1Ω.

```
DATA_POINTS = DATA_POINTS_LOC.value
MAX_RESISTANCE = MAX_RESISTANCE_LOC.value
RESISTANCE_INC = MAX_RESISTANCE / DATA_POINTS
RESISTANCE = 1#
Set cel = DATA_COLLECT_LOC
```

Then the first data point is collected, the I_{sc}, or *short circuit current.* The resistance is set to the lowest possible resistance the *HP Electronic Load* can handle, 0.033Ω; this is done by adjusting the resistance range and then changing the resistance. (In VB, the ampersand is string concatenation, and does the type conversion automatically.)

```
Call Write_Dev(ELEC_LOAD_DEV_ID, "RES:RANG " & 0.033)
Call Write_Dev(ELEC_LOAD_DEV_ID, "RES " & 0.033)
Sleep 500
AMPS = Read_Dev(MULTI_METER_2_DEV_ID, 2)
cel(1, 1).value = 0
cel(1, 2).value = AMPS
```

After the resistance has been set, the circuit is allowed to settle and then the amperage is read from the ammeter. The data are then output to the spreadsheet. Now a data collection loop can commence. The code segment below has been numbered for reference.

```
1   I = 2
2   While I < DATA_POINTS
3      Call Write_Dev(ELEC_LOAD_DEV_ID, "RES:RANG " & RESISTANCE)
4      Call Write_Dev(ELEC_LOAD_DEV_ID, "RES " & RESISTANCE)
5      VOLTS = Read_Dev(MULTI_METER_1_DEV_ID, 2)
6      If VOLTS < 2 Then
7          AMPS = Read_Dev(MULTI_METER_2_DEV_ID, 2)
8          cel(I, 1).value = VOLTS
9          cel(I, 2).value = AMPS
10       End If
11       I = I + 1
12       RESISTANCE = RESISTANCE + RESISTANCE_INC
13    Wend
```

The basic procedure is as follows:

1. Set the resistance *[lines: 3, 4]*

2. Read from voltmeter *[line: 5]*

3. Check for erroneous data *[line: 6]*

4. Read amps and output *[lines: 7, 8, 9]*

5. Update loop counter, I, and RESISTANCE *[lines: 11, 12]*

Finally, the V_{oc} is calculated, the last data point output, and the devices are closed.

```
Call Write_Dev(ELEC_LOAD DEV_ID, "RES:RANG " & 20000)
Call Write_Dev(ELEC_LOAD_DEV_ID, "RES " & 20000)
Sleep 500
VOLTS = Read_Dev(MULTI_METER_1_DEV_ID, 2)
cel(I, 1).value = VOLTS
cel(I, 2).value = 0

Call Close_Dev(MULTI_METER_1_DEV_ID)
Call Close_Dev(MULTI_METER_2_DEV_ID)
Call Close_Dev(ELEC_LOAD_DEV_ID)
```

From here the raw data can be manipulated and displayed with Excel.

IV CURVE: COMMENTS

The example just given could have been done with just one multimeter. In that case, the program would switch the multimeter between reading volts and amperes. This would be no problem, although it would be slower. There are many other ways one might set up this experiment, it just depends on the availability of hardware and what you need.

Other calculations for the example above, for example *fill factor*, may be calculated from the raw data collected using Excel. This can also be controlled using a VB script or can be done manually after you collect the data. How much computation to automate, either by writing procedures in VB, or using calls to Excel is totally up to the user and can be customized accordingly.

CONCLUSION

A cheap alternative to expensive software and hardware (i.e., LabVIEW and A/D data acquisition boards) works well for simple experiments. This approach uses a GPIB board and software, with GPIB-compatible devices (e.g., multimeters, electronic loads, etc.) and Microsoft Excel to capture data.

ACKNOWLEDGMENTS

I would like to acknowledge the assistance of Sam Edwards, NAVAIR WD, Public Works, who explained the basics of the *IV Curve* data collection setup and donated some of the hardware used for the *IV Curve* generation. Thanks also to the creative folks behind the **Google** search engine for helping me find good VB documentation and help.

REFERENCES

1. Fluke, 8840A Multimeter, Instruction Manual, May 1984.

2. National Instruments, GPIB User Manual for Win32, June 1998.

3. Hewlett Packard, Electronic Load Mainframes, HP Models 6050A, Operating Manual, May 1993.

4. Field, H., UV-VIS-IR Spectral Responsivity Measurement System for Solar Cells, presented at the National Center for Photovoltaics Program Review Meeting, September 8-11, 1998.

APPENDIX

A-1. GPIB vs. A/D Boards

There is a good comparison between internal (e.g., PCI, ISA, etc.) A/D boards vs. using external instruments with GPIB, or RS232. The comparison is provided by Keithley™ and can be found at the following web address:

- `http://www.keithely.cl/misc/across/Selector_Guide.pdf`

A-2. Timing

There are many ways to achieve timing in VB, all with varying degrees of accuracy and complexity. See VB literature for more information. One simple way is to do the following. First, define a GetTickCount method:

```
Public Declare Function GetTickCount Lib "kernel32.dll" () As
Long
```

Depending on your application you can use GetTickCount however you like, for example recall our first example, `SIMPLE.bas`. If you wanted to record, approximately, when each voltage measurement was taken starting from *time* = 0, you could do the following. Initialize a starting point,

```
TIME_O& = GetTickCount
```

Then, wherever you wanted a current time, as a Double, use

```
TIME_NOW# = ( (GetTickCount - TIME_O&) / 1000#)
```

A-3. SIMPLE.bas

```
Public Sub Main ()
' Set USER defined I/O locations
    Set DATA_POINTS_LOC = Range("A1")
    Set DATA_COLLECT_LOC = Range("A2")
  Open Device
    Call ibdev(0, 6, 0, 15, 1, 0, DEV_ID\%)
' Write to device its initial settings
    Call ibwrt(DEV_ID\%, "F1 S1 TO")
' Gather input parameters
```

```
    DATA_POINTS\% = DATA_POINTS_LOC.value
' Perform data collection
    Readbuf\$ = Space\$(20)
    Set cel = DATA_COLLECT_LOC
    I = 1
    While I < DATA_POINTS\% + 1
         'take measurements
         Call ibrd(DEV_ID\%, Readbuf\$)
         rd\$ = Left(Readbuf\$, (ibcnt\% - 2))
       ' output data point
            cel(I, 1).value = CDbl(rd\$)
        ' update loop variable
            I\% = I\% + 1
    Wend
' Return device to local mode and take offline
    Call ibloc(DEV_ID\%)
    Call ibonl(DEV_ID\%, 0)
End Sub
```

A-4. USER LIB.bas

(In the source code below, VB statements have been split over several lines for readability only; errors will occur if they remain in this form.)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Open_Dev:
'
' Opens and initializes remote component(i.e. multimeter, etc.)
'
' Input: board_index      _ GPIB board address
'          prim_add       _ primary GPIB address
'          sec_add        _ secondary GPIB address
'          time_out       _ time device waits before timing out,
                            in seconds
'          end_mess_flag  _ end message flag, usually true
'          eos_flag       _ end of string mode flag, usually false
' Output: dev_id          _ device identifier

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Private Function Open_Dev(ByVal board_index As Integer,
                          ByVal prim_add As Integer,
                          ByVal sec_add As Integer,
                          ByVal time_out As Integer,
```

```
                                ByVal end_mess_flag As Integer,
                                ByVal eos_flag As Integer) As Integer

   Dim dev_id As Integer
   Dim ln_flag As Integer

   Call ibdev(board_index,
              prim_add,
              sec_add,
              time_out,
              end_mess_flag,
              eos_flag,
              dev_id)

If (dev_id < 0) Then
    ERROR_MESSAGE ("Can't initialize device, gpib addr = " & prim_add)

End If

Call ibclr(dev_id)

' checking for presence of device
Call ibln(dev_id, prim_add, sec_add, ln_flag)
If ln_flag = 0 Then
      ERROR_MESSAGE ("Device not working, check power, gpib addr = " &
               prim_add)

   End If

   Open_Dev = dev_id

End Function


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Close_Dev:
'
' Returns device to local control and takes it offline
'
' Input: dev_id         - device identifier
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub Close_Dev(ByVal dev_id As Integer)

   Call ibclr(dev_id)
   Call ibloc(dev_id)
   Call ibonl(dev_id, 0)
```

```
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Write_Dev:
'
' Writes command string to remote device
'
' Input: dev_id                  - device identifier
'        buf                     - command string buffer
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub Write_Dev(ByVal dev_id As Integer,ByVal buf As String)
   Call ibwrt(dev_id, buf)
   If (ibsta And EERR) Then
      ERROR_MESSAGE ("Error = " &
                  iberr &
                  "Couldn't trigger device, dev_id = " & dev_id)
   End If


End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Read_Dev:
'
' Read device output buffer
'
' Input: dev_id             - device identifier
'        offset             - buffer offset
'
' Output: value             - the value of the buffer
                              in decimal form as a double
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Function Read_Dev(ByVal dev_id As Integer,
                          ByVal offset As Integer) As Double

   Readbuf = Space(20)
   Call ibrd(dev_id, Readbuf)
   If (ibsta And EERR) Then
      ERROR_MESSAGE("Error = " &
                    iberr &
                    "Couldn't read from device,dev_id = " & dev_id)

   End If
```

```
   rd = Left(Readbuf, (ibcnt - offset))
   Read_Dev = CDbl(rd)

End Function

' Error message routine, calls VB's MsgBox
Private Sub ERROR_MESSAGE(ByVal err As String)
   MsgBox err
   End
End Sub
```

A-5. IV CURVE. bas

```
Public Sub Main ()

' Declare Device id's
     Dim MULTI_METER_1_DEV_ID As Integer
     Dim MULTI_METER_2_DEV_ID As Integer
     Dim ELEC_LOAD_DEV_ID As Integer

' Declare USER I/0 locations as Range
     Dim DATA_POINTS_LOC As Range
     Dim DATA_COLLECT_LOC As Range
     Dim MAX_RESISTANCE_LOC As Range

' Declare other USER variables
     Dim I As Integer
     Dim cel As Range
     Dim DATA_POINTS As Integer
     Dim MAX_RESISTANCE As Double
     Dim RESISTANCE As Double
     Dim RESISTANCE_INC As Double
     Dim VOLTS As Double
     Dim AMPS As Double

' Set USER defined I/0 locations
     Set DATA_POINTS_LOC = Range("B1")
     Set MAX_RESISTANCE_LOC = Range("B2")
     Set DATA_COLLECT_LOC = Range("C1")
```

```
' Open Device(s)
    MULTI_METER_1_DEV_ID = Open_Dev(0, 1, 0, 15, 1, 0)
    MULTI_METER_2_DEV_ID = Open_Dev(0, 6, 0, 15, 1, 0)
    ELEC_LOAD_DEV_ID = Open_Dev(0, 5, 0, 15, 1, 0)

' write initial device modes
    Call Write_Dev(MULTI_METER_1_DEV_ID, "F1 S1 TO")
    Call Write_Dev(MULTI_METER_2_DEV_ID, "F5 S1 TO")
    Call Write_Dev(ELEC_LOAD_DEV_ID, "MODE:RES")
    Call Write_Dev(ELEC_LOAD_DEV_ID, "INPUT ON")

' Gather input parameters
    DATA_POINTS = DATA_POINTS_LOC.value
    MAX_RESISTANCE = MAX_RESISTANCE_LOC.value

' Perform needed pre_run calculations
  ' Calculate resistance increment
  RESISTANCE_INC = MAX_RESISTANCE / DATA_POINTS

  ' Initialize resistance setting (1 Ohm)
  RESISTANCE = 1#

' Perform data collection
    Set cel = DATA_COLLECT_LOC

    ' calculate Isc and output first data point
    Call Write_Dev(ELEC_LOAD_DEV_ID, "RES:RANG " & 0.033)
    Call Write_Dev(ELEC_LOAD_DEV_ID, "RES " & 0.033)
    Sleep 500
    AMPS = Read_Dev(MULTI_METER_2_DEV_ID, 2)
    cel(1, 1).value = 0
    cel(1, 2).value = AMPS

    I = 2
    While I < DATA_POINTS

      ' set load
        Call Write_Dev(ELEC_LOAD_DEV_ID,"RES:RANG "& RESISTANCE)
        Call Write_Dev(ELEC_LOAD_DEV_ID, "RES " & RESISTANCE)

      ' take voltage measurement
        VOLTS = Read_Dev(MULTI_METER_1_DEV_ID, 2)

      ' through out erroneous voltage measurements
```

```
            If VOLTS < 2 Then
                ' take amp meas.
                AMPS = Read_Dev(MULTI_METER_2_DEV_ID, 2)
                ' output data point
                cel(I, 1).value = VOLTS
                cel(I, 2).value = AMPS
            End If

        ' update loop variable and resistance increment
                I = I + 1
                RESISTANCE = RESISTANCE + RESISTANCE_INC
Wend

' calculate Voc(last data point), and output
  Call Write_Dev(ELEC_LOAD_DEV_ID, "RES:RANG " & 20000)
  Call Write_Dev(ELEC_LOAD_DEV_ID, "RES " & 20000)
  Sleep 500
  VOLTS = Read_Dev(MULTI_METER_1_DEV_ID, 2)
  cel(I, 1).value = VOLTS
  cel(I, 2).value = 0

' Close devices
  Call Close_Dev(MULTI_METER_I_DEV_ID)
  Call Close_Dev(MULTI_METER_2_DEV_ID)
  Call Close_Dev(ELEC_LOAD_DEV_ID)

End Sub
```