A versatile platform for programming and data acquisition: Excel and Visual Basic for Applications

Dr. Harold T. Evensen, University of Wisconsin, Platteville

Hal Evensen earned his doctorate in Engineering Physics from the University of Wisconsin-Madison, where he performed research in the area of plasma nuclear fusion. Before joining UW-Platteville in 1999, he was a post-doctoral researcher at the University of Washington, part of group that developed automation for biotechnology. His recent research includes collaborations in energy nanomaterials.

A versatile platform for programming and data acquisition: Excel and Visual Basic for Applications

We have switched to a new software platform to for instrument interface and data collection in our upper-division Sensor Laboratory course. This was done after investigating several options and after several meetings with our industrial advisory board. This change was motivated by a campus-mandated change in operating system, plus expiring software licenses. We decided on the Visual Basic for Applications platform (VBA), which resides in the Microsoft Office suite (in particular, MS Excel). This meets the recommendations of our advisory board, which strongly urged that we use a "traditional" programming language, as opposed to a graphical one, in order to provide our students the broadest possible applied programming "base." Further, it has the advantage that the platform is widely available and the required add-ins are free, so that graduates (and other programs) are able to use this as well. Finally, this approach has the added benefit of extending the students' prerequisite computer programming into VBA, which is useful beyond the realm of instrument control and data acquisition. This paper will describe the resources that were collected – and modified – by the author, for the apparently novel combination of VBA, 64-bit programming, and access to both serial/USB and GPIB instrumentation, and provides examples of implementation. The basic principles of VBA for non-experts will also be given, as well as strengths and drawbacks of this approach. We will also report on the first offering of the redesigned course and remark on future improvements.

Introduction

The University of Wisconsin-Platteville is an undergraduate, regional university of just over 7,500 on-campus students. Its 41 majors include seven accredited engineering programs, including Engineering Physics (EP), which typically produces 12 to 18 graduates per year, 90% of whom directly enter the workforce. The program has five faculty members, whose teaching responsibilities are split between introductory physics (mostly for engineering majors) and upper-division courses for EP majors.

Sensor Lab (ENGRPHYS 4210) is a required, two-credit laboratory course for EP majors that has two two-hour class periods per week. It is typically taken in a student's 3rd or 4th year. This course builds on prerequisite courses in Modern Physics, Circuits II, and C++ Programming. Students learn the "physics behind the sensor," and the effect this has on a sensor's implementation. They also learn about circuits used to interface with both digital and analog sensors. In addition, they learn Visual Basic (building on their programming skills developed in the C++ prerequisite) and use this to interface the computer to the sensor by using a DAQ (data acquisition) device.

Typical class sizes are 12 to 16, with one to two sections per year. The course begins with a lab orientation and an exercise that introduces students to temperature measurement using an analog temperature sensor,[1] a serial DAQ[2] and programming with Visual Basic 6 (VB6): students construct a sensor, calibrate it, and write a VB6 program that uses the DAQ to read the sensor's output and displays the measured temperature.

After the class completes this introductory procedure, the course is run as a round-robin laboratory, where student teams spend three two-hour class periods at a sensor station and then rotate to another experiment. At these stations, students explore different sensors (strain gages; proximity sensors; LVDTs; MEMS accelerometer & gyroscopes; rotary encoders, to name a few). Depending on the sensor, tasks may include interfacing to digital or analog outputs; creating a program in VB6 that utilizes the DAQ's inputs & outputs; and determining the characteristics of the sensor. Finally, the last two to three weeks are given over to an open-ended final project in which students are to "use a sensor to solve a 'problem' for a customer." Figure 1 shows some of the stations in the laboratory used by this course. Several of these stations implement VB6 to control the apparatus and to acquire data from benchtop instruments.

Figure 1. The Engineering Physics Sensor Lab.

Exiting seniors and EP alumni regularly identify this course as one of the strengths of the Engineering Physics curriculum. Their comments indicate that the goals of the course are being met. These goals include:

- Learning about a variety of common sensors and their implementation, including sensor terminology.
- Developing an electronics skill set to pull information from the sensors;
- Developing laboratory skills, such as experiment design and implementation, troubleshooting, and reporting;
- Learning and applying a new programming language (VB6) to communicate with external equipment (the DAQ), interpret sensor signals, and "make decisions" based on the sensor output.

Programming Languages in Sensor Lab

The Sensor Lab has used VB6 since its inception in 1998. While the language is aging (extended support for Microsoft Visual Basic 6.0 ended in 2008, though VB6 programs will run on operating systems through at least Windows 8[3]), it remains as a "legacy" language that is still

widely used. Still, it has been replaced in new applications by Visual Basic .NET and Visual Studio.[4] With this in mind, in 2008 the author explored converting the Sensor Lab to replace VB6 with LabVIEW (National Instruments, Inc.), a popular graphical programming language that is used for data acquisition, instrument control, and automation. This conversion process began by creating an "Introduction to LabVIEW" module, to provide students exposure to this environment.

While student responses were mildly positive, the feedback from alumni and employers (at EP program advisory board meetings) was very negative. We were strongly advised by our constituency that the "coding" aspect of the course was very valuable. They saw Sensor Lab as an EP major's first – and sometimes only – "real application" of the programming skills learned in the C++ prerequisite, and even though none were actually programming in C++ or VB6, they saw the skills developed by using these languages as transferable and important for the programming that most of them were doing in their jobs. (LabVIEW, with its graphical programming interface, was seen as powerful but expensive and lacking in both "market penetration" and development of transferable programming skills: they recommended exposing students to it, but not restructuring the course around it.)

We thus kept VB6 in Sensor Lab, knowing that we would eventually need to change. This change was ultimately demanded by our university's conversion to 64-bit Windows 7 (Win7): our IT support person advised that there were issues surrounding VB6's access of the COM ports in 64-bit Win7, and that it may be time to move on. (A survey of several online programming forums showed this to be true. While there may be some workarounds that "fix" this issue, these did not seem worth it, given that the rest of the world is moving past VB6, especially for new applications.)

We considered Microsoft Visual Basic 2008 and 2010, which are freely available in "Express" versions. However, the author – who is not a programmer – found these intimidating and much more powerful (and complicated) than what was needed for Sensor Lab: they lacked the short "learning curve" of VB6 that prevented the programming aspect from overtaking the entire course. Further, these had a similar weakness shared with VB6 and LabVIEW: the full-scale, expensive packages may not be something to which the student has access upon graduation.

A survey of other university courses similar to Sensor Lab showed that LabVIEW and Matlab were commonly used;[5] however the proprietary nature of these platforms makes it very likely that they would also not be available to students after graduation. Another possibility, utilized by some courses, was to use a "smaller" language that is particular to a device, such as BASIC (i.e. for the BASIC stamp from Parallax, Inc.) or Dynamic C (for the Rabbit ® microprocessor from Digi International, Inc.);[6] these are affordable, have a shorter learning curve, and serve to solidify a student's programming fundamentals – though they have even less market penetration than the preceding languages.

Finally, the author considered – and ultimately settled on – Visual Basic for Applications (VBA) within Microsoft Excel.

Visual Basic for Applications: Overview

Visual Basic for Applications (VBA) is an implementation of the Visual Basic 6 language, which is built into most Microsoft Office applications. With VBA in Excel, users can go beyond simple macros and write programs that draw on both the Excel functionality (functions, charts, shapes) as well as those of the Windows Application Programming Interface (API).[7,8]

Implementing VBA in education is not novel; at other institutions, students have been exposed VBA in several ways. Some programs have used it as the primary programming language for underclassmen, for the reasons that Excel (and hence VBA) is nearly ubiquitous and that VBA is an accessible language. Past work has reported the results of this approach in Chemical, Civil, Mechanical, and Biomedical Engineering programs,[9,10,11] as well as for general introductory programming courses for students in engineering[12] and engineering technology.[13]

While this body of work showed that VBA is a viable platform for developing programming skills, the author was unable to find many examples of VBA implementation in an educational laboratory setting, aside from creating "active" instruction sheets[14] (in which case the students were not programming), or as part of a student solution to an open-ended project, both in the literature[15] and from the author's personal experience. (That said, these solutions still relied upon 3rd-party solutions for "collecting data into Excel."[16]) Being able to apply VBA to data acquisition and instrument communication and control would present the same advantages as selecting VBA for a programming language: ease of use, widespread acceptance, power and adaptability.

This paper presents modules, references, and code that enable one to use Excel with VBA as a means of communicating with serial-based instruments as well as those connected via USB or GPIB. All of the components are freely available, and Excel/VBA thus becomes a viable means of communicating with – and controlling – the outside world.

Visual Basic for Applications: Basics in Excel

Several books and websites exist that show the basics of VBA in Excel.[17] Here we present an overview of how to get started.

1. First, enable the "Developer" tab in Excel (see Figure 2 below). This is done by checking the appropriate box under "Excel Options," and enables access to VBA programming and code.

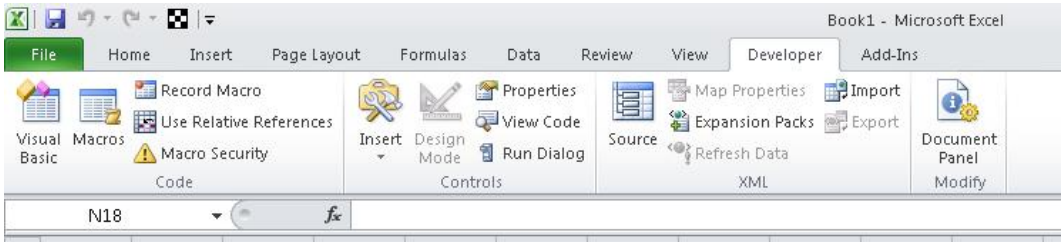

Figure 2. Developer tab in Excel 2010 (from Microsoft Developer Network; http://msdn.microsoft.com/en-us/library/ee814737(v=office.14).aspx .

2. Selecting "Visual Basic" (leftmost button in Figure 2) switches to the VBA view, which shows the code windows. The "Project" window (Figure 3 below) shows the worksheets in the workbook, as well as any modules associated with the workbook. (Modules are containers for code that can be used by the entire workbook; right-clicking on the VBA Project allows one to insert or import a new module.)
3. By double-clicking the module, a coding window appears (Figure 3). Here, one may create functions – which return a value – or a subroutine, which can follow more complex instructions. Once created, these can be called within Excel or by other subroutines and functions.

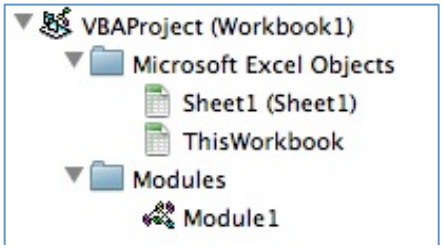

Figure 4. Project window in VBA. Modules are "containers" for code that can be accessed by the entire workbook.

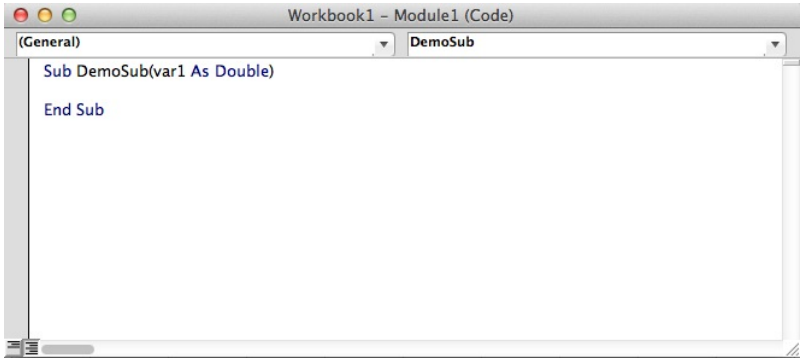

Figure 3. Coding window for a module.

4. VBA can be used to collect and act upon user input by way of a UserForm, which can be added via the Project window (right-click on the Project). UserForms can have their own code and actions, and are also able to use the routines in the Modules. In essence, the UserForm is provides a means of gathering information from the user, and acting on that

information. A screen shot of a sample UserForm used to test a DAQ is shown in Figure 5 below; the Sensor Lab uses UserForms to operate several stations, as will be shown below.

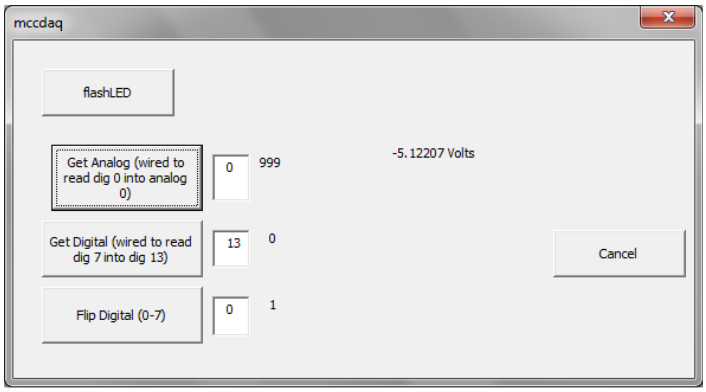

Figure 5. A sample UserForm, used to test a DAQ's digital and analog ports.

Serial port communication with VBA

Communicating with the RS-232 port (the "COM" port) in VBA is fundamental to instrument communication. Many simple devices, such as balances, thermocouple meters, and data acquisition (including our DAQs[i]) can connect to this port. Several advanced devices, including lab bench equipment (multimeters and power supplies) also have RS-232 connections; they often have a simple ASCII command set that can be used for remote operation.

However, Visual Basic – and by extension, VBA – typically performs serial port I/O by way of the Microsoft Comm Control component. This is a part of the Visual Studio package, and is therefore not as widely available as Excel. Fortunately, sample VBA code has been developed and freely distributed that uses the Windows API to perform the I/O operations. The code that was found at several sites online[18] was created for up to four serial ports and 32-bit systems – and therefore not compatible with 64-bit Win7. With very minor modifications, the author adapted this code for 64-bit systems and a greater number of serial ports; the resulting module is posted in the author's university web space.[ii]

This module, which can be easily imported into any VBA project, thus enables Excel with VBA to communicate with any serial device. Sample macro-enabled Excel files posted in the author's university web space demonstrates communication with an ADR 2000 DAQ via the RS-232 port (Ontrak Control Systems, Inc) as well as with a DI-149 DAQ via the USB port (DATAQ Instruments, Inc.). [19]

[i] ADR 2000 series DAQs from Ontrak Control Systems, Inc.
[ii] The module, titled "modCOMM," is 20 pages long – but only a 28 kilobyte .BAS file that can be readily imported into any VBA project.

Of course, much modern instrumentation instead ships with libraries to facilitate communication with a PC. Fortunately, there is enough overlap between VBA and Visual Basic that any existing Visual Basic libraries can be used with little to no modification. For example, to communicate with the USB-1208LS DAQ (Measurement Computing, Inc.) using VBA, one need only import their Universal Library declaration file CBW.BAS (a free download, designed for Visual Basic, which also ships with the DAQ) as a module in the VBA program.[20] With the use of this library, VBA can then be used to control the DAQ without need for the Windows API module described above. The sample worksheet (described above; posted online) illustrates communication with the USB-1208LS DAQ.

Instrument communication via USB and GPIB

Finally, students in Sensor Lab also have used VB6 to take remote readings from benchtop instruments, such as a Digital Multimeter (DMM), which can offer better stability and precision than a DAQ. This communication is commonly done using the Virtual Instrument Software Architecture (VISA). According to National Instruments,[21]

> "VISA is a widely used standard for configuring, programming, and troubleshooting instrumentation systems comprising GPIB, VXI, PXI, Serial, Ethernet, and/or USB interfaces. VISA provides the programming interface between the hardware and development environments such as LabVIEW, LabWindows/CVI, and Measurement Studio for Microsoft Visual Studio."

Different companies may have their own implementations of the VISA I/O standard (i.e. National Instruments, Inc. and Agilent Technologies, Inc.), and these typically include software libraries and configuration programs. VISA libraries are freely available, and it is a fairly straightforward process to enable VBA to access these useful (and free) VISA commands.[22] An outline of this process is as follows:

1. Download and install the VISA software libraries. These are free from vendors such as Agilent Technologies ("IO Libraries Suite") and National Instruments ("NI-VISA").[23]
2. In Excel's VBA editor, add the VISA COM library reference. This is done by selecting "Tools > References…" (see Figure 6 below), and then checking the box next to the "VISA COM 3.0 Type Library" in the subsequent list.
3. Use the simple (and common) VISA library commands to address the device of interest[24,25] and to send device-specific commands (i.e. standard SCPI[iii] commands, included with the instrument's documentation). Sample code is available for download at the author's web site.
4. Communicating with an instrument requires use of its "VISA address." To find this address, one uses the configuration program that came with the VISA library download. Two examples are the NI-MAX (National Instruments Measurement and Automation Explorer) and the Agilent Connection Expert. These programs will reveal every instrument connected to the machine, and can be used to determine the VISA address. For example, the address of a GPIB-connected device may be "GPIB0::2::INSTR"; for a USB-connected instrument it may be "USB0::0x0699::0x0363::C063380::0::INSTR".

[iii] SCPI = Standard Commands for Programmable Instruments, maintained by the IVI Foundation. See http://www.ivifoundation.org/specifications/default.aspx, accessed Jan. 4, 2014.

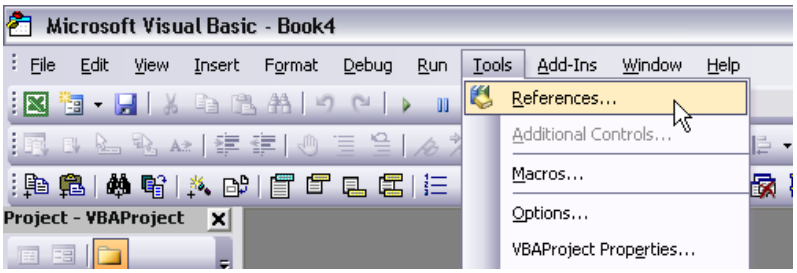

Figure 6. Selecting the VISA COM reference to a VBA project enables the program to communicate with a wide range of instruments through several potential connections.

Thus, VBA can use freely available, standard libraries to communicate with devices over a wide range of interfaces (USB, GPIB, Ethernet). These commands can be integrated into programs to produce a powerful application. The author has successfully converted the Sensor Lab stations, formerly run via VB6, to be run with VBA. The new programs communicate with a DAQ and benchtop instruments, drive stepper motors, and save data to the spreadsheet. A screen shot of a sample UserForm is shown in Figure 7 below; a commented sample program that illustrates these capabilities is available for download at the author's site.

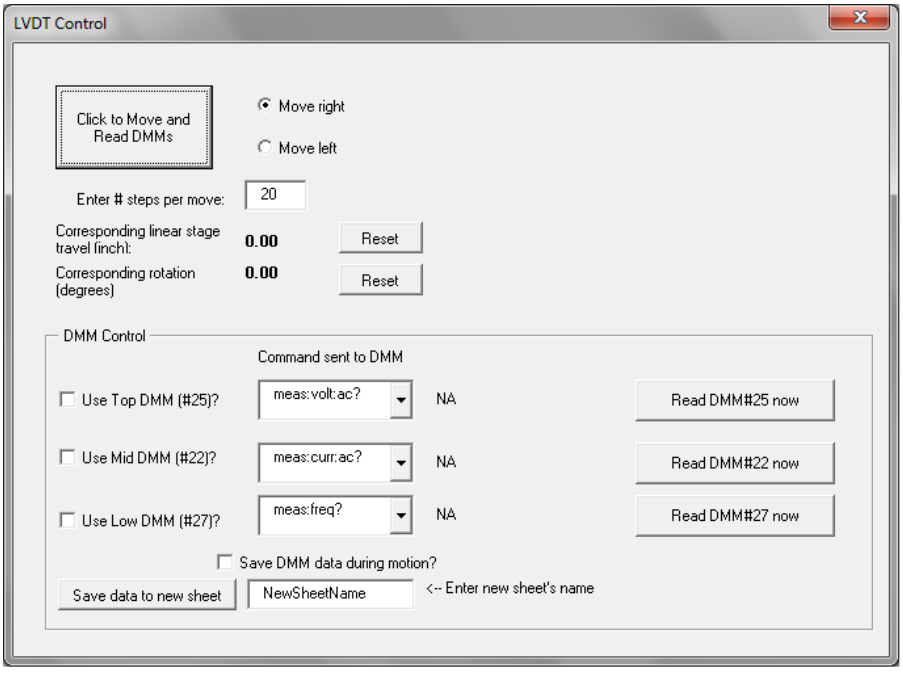

Figure 7. UserForm used to control the "LVDT" station in the Sensor Lab. This form drives a DAQ's digital outputs to move a stepper motor, queries multimeters, and saves the data to a worksheet.

Summary

Using Excel and VBA for instrument communication was implemented for the first time in the Spring 2014 offering of Sensor Lab. Because it is anticipated that students will be interested in

this "little known" feature of Excel, and motivated to investigate further for their own applications, some time will be devoted to VBA programming. The order of presentation is:

A. Introduction to VBA. Writing simple functions and macros.
B. Subroutines; using controls and buttons in VBA.
C. Creation of UserForms.

At this point, students will shift to constructing a temperature sensor. Once its operation is verified, they will use Excel/VBA with a DAQ to monitor the sensor's output.

D. Serial port communication with VBA
E. Utilizing the DAQ's digital and analog inputs and outputs.
F. Use VBA to drive an action (by way of the DAQ) when a threshold temperature is attained.

It is anticipated that students will find VBA programming more useful and relevant than VB6; it is hoped that this will spur an improvement in programming proficiency. A class survey will be used to assess students' programming competence and whether they expect to use VBA after the conclusion of the course.

Acknowledgment

The bulk of the stations and procedures in the Sensor Lab at the University of Wisconsin-Platteville was created by Prof. W. Doyle St. John, and the author gratefully acknowledges his support over the years.

The University of Wisconsin-Platteville's Curriculum Improvement Fund supported the work reported here.

References

[1] Texas Instruments LM135 series precision temperature sensor; http://www.ti.com/lit/ds/symlink/lm335.pdf accessed Jan. 3, 2014.
[2] OnTrak Control Systems' ADR 2000 serial data acquisition interface; http://www.ontrak.net/adr2000.htm accessed Jan. 3, 2014.
[3] "Support Statement for Visual Basic 6.0 on Windows Vista, Windows Server 2008, Windows 7, and Windows 8." http://msdn.microsoft.com/en-us/vstudio/ms788708.aspx accessed 1/3/2014.
[4] A recent job search at Indeed.com (Jan. 3, 2014) shows 630 jobs requiring VB6; 5,700 listing Visual Basic .NET; 9,500 requiring Visual Studio; 1,600 listing LabVIEW; and 3,600 listing VBA (Visual Basic for Applications).
[5] For example, the University of Washington's BIOEN 317: Biomedical Signals and Sensors Laboratory uses Matlab and LabVIEW (https://depts.washington.edu/bioe/resources/course-syllabi/bioen-317/); U. Illinois' ECE 437 Sensors and Instrumentation (https://wiki.engr.illinois.edu/display/HKNDEN/ECE+437+-+Sensors+and+Instrumentation); U. Wisconsin's ECE 317 Sensors Laboratory (http://aefis.engr.wisc.edu/index.cfm/page/CourseAdmin.ViewABET?coursecatalogid=501&pdf

=True); Stanford's ME220 Introduction to Sensors (http://www.stanford.edu/class/me220/). Sites accessed January 2014.

[6] See, for example, U. Minnesota's AEM 4601 Instrumentation Laboratory (http://www.aem.umn.edu/teaching/curriculum/syllabi/UGrad/AEM_4601_syllabus.shtml); Johns Hopkins' 530.420 Robot Sensors and Actuators (https://www.lcsr.jhu.edu/Education/Courses/530_420_FALL2012). Sites access January 2014.

[7] "Visual Basic for Applications." January 4, 2014. In *Wikipedia: The Free Encyclopedia*. http://en.wikipedia.org/wiki/Visual_Basic_for_Applications

[8] Ribando, R.J., "An Excel/Visual Basic for Applications (VBA) Primer," *Computers in Education Journal,* Vol. VIII, No. 2, April-June 1998, pp. 38-43.

[9] List, George, "Introducing Civil Engineering Analysis Through Programming," *Proceedings of the 2007 Annual ASEE Conference,* Honolulu, HI, June 24-27, 2007.

[10] Thomas Jr., Garth E. et al., "Evolution of a Freshman Software Tools Class," *Proceedings of the 2005 Annual ASEE Conference,* Portland, OR, June 12-15, 2005.

[11] Griffin, L., and R. Crockett, "Biomedical Engineering Simulation Using Visual Basic Macros in Microsoft Excel," *Proceedings of the 2007 Annual ASEE Conference,* Honolulu, HI, June 24-27, 2007.

[12] Maase, Eric, "Kangaroo Thinking: Mathematics, Modeling, and Engineering in Introductory Computer Programming for Engineers," *Proceedings of the 2007 Annual ASEE Conference,* Honolulu, HI, June 24-27, 2007.

[13] Myszka, D., "Motivating Students in an Introduction to Computing Course by Requiring Animated Solutions," *Proceedings of the 2006 Annual ASEE Conference,* Chicago, IL, June 18-21, 2006.

[14] Poole, Nigel, "Activating Laboratories using Visual Basic for Applications," Journal of the Higher Education Academy Engineering Subject Centre, Vol 2, No 2 (2007).

[15] Frasch, Lewis G., "Utilizing Student Computers for Laboratory Data Acquisition in a University-Wide Laptop Environment, *Proceedings of the 2002 Annual ASEE Conference,* Montreal, Canada, June 16-19, 2002.

[16] Three examples include: (1) *Collect*, sold by companies including LabTronics, Perkin-Elmer, and Omega, acquires data through the RS-232 port; (2) Windmill Software Ltd. produces *Windmill RS232* (http://www.windmill.co.uk/excel/) to capture data from serial instruments to Excel; (3) TalTech's *WinWedge* software (http://www.taltech.com/winwedge), used in Reference 15, captures data directly to Excel, Access or any Windows application or web page.

[17] Walkenbach, J., "Microsoft® Excel® VBA Programming for Dummies," Wiley Publishing, Indianapolis (2010); *see also* "Getting Started with VBA in Excel 2010," downloaded from http://msdn.microsoft.com/en-us/library/ee814737(v=office.14).aspx on Jan. 4 2013 (Microsoft Developer Network).

[18] The serial I/O module was created by David M. Hitchner; the code was downloaded from http://www.thescarms.com/VBasic/commio.aspx. Another site has similar code (though without the acknowledgment to Mr. Hitchner): http://dev.emcelettronica.com/serial-port-communication-excel-vba . Finally, the code (again without the acknowledgment to Mr. Hitchner) was found as part of a sample code for controlling a Dimetix, Inc. distance sensor at this site: http://www.dimetix.com/appl/_FRMappl.html .

[19] http://www.uwplatt.edu/~evensenh/Hal_Evensens_Homepage/VBA_Communication.html

[20] "Universal Library Help," from Measurement Computing, Inc. Downloaded from http://www.mccdaq.com/GetPDF.aspx?t=/PDFs/manuals/Universal-Library-Help.pdf , January 2014.
[21] "National Instruments VISA," downloaded from the National Instruments, Inc. web site January 2014. http://www.ni.com/visa/ .
[22] Also see the useful video tutorial: "Visual Basic for Excel, Simple Example Program to Control Instruments" http://www.youtube.com/watch?v=Y9-I6GoXg3U , accessed Jan. 4, 2014.
[23] National Instruments' "NI-VISA 5.3:" http://www.ni.com/download/ni-visa-5.3/3823/en/ ; Agilent Technologies' "IO Libraries Suite 16.3:" http://www.home.agilent.com/en/pd-1985909-pn-E2094/io-libraries-suite-162?&cc=US&lc=eng . Both accessed January 4, 2014.
[24] "Controlling instruments with Excel and the bundled Visual Basic for Applications," Agilent Technologies web site, accessed January 4, 2014. http://www.home.agilent.com/agilent/editorial.jspx?ckey=474172-1-eng&id=474172-1-eng&nid=-536900124.0.08&lc=eng&cc=US
[25] "How to get VISA GPIB to work in Excel, Excel VBA, VB6 or Visual Basic 6," National Instruments web site, accessed January 4, 2014. http://forums.ni.com/t5/Instrument-Control-GPIB-Serial/How-to-get-VISA-GPIB-to-work-in-Excel-Excel-VBA-VB6-or-Visual/td-p/1450478