# AC 2008-3: INTEGRATION OF PROGRAMMABLE LOGIC CONTROLLER PROGRAMMING EXPERIENCE INTO CONTROL SYSTEMS COURSES

**Thomas Cavicchi, Grove City College**

Thomas J. Cavicchi received the B. S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1982, and the M. S. and Ph. D. degrees in electrical engineering from University of Illinois, Urbana, in 1984 and 1988, respectively. He is a Professor of Electrical Engineering at Grove City College, Grove City, PA, where he teaches year-long courses on digital communication systems, digital and analog control systems, and the senior labs (including co-teaching the senior capstone design projects). He also has recently taught courses on signals and systems and electrical engineering for nonelectrical engineering majors. He is the author of Digital Signal Processing (New York: John Wiley & Sons, 2000) and Fundamentals of Electrical Engineering: Principles and Applications (Englewood Cliffs, NJ: Prentice-Hall, 1993). He has taught graduate classes on digital signal processing and digital spectral analysis, and has conducted research on DSP and ultrasonic diffraction scattering for medical imaging. Dr. Cavicchi is a member of Sigma Xi.

# Integration of Programmable Logic Controller Programming Experience Into Control Systems Courses

Department of Electrical and Computer Engineering
Grove City College
Grove City, PA 16127

**Abstract**

The two-semester senior electrical engineering course in control systems includes a segment on the programmable logic controller (PLC). The PLC is a valuable educational venue because it includes a variety of aspects that can prepare the budding engineer for the real world. Some of these are: the learning how to use and the features of massive industry-standard application programs (including digging through lengthy online manuals), the exposure to highly versatile and complex hardware that is ubiquitous in industrial automation, learning to work with other colleagues in a situation of limited resources (only one PLC, though there are multiple copies of the software, including an emulator), a new kind of programming—ladder logic in RSLogix500—that at the senior year can be refreshing now that the main programming languages in coursework are all too familiar (and at times frustrating), use of local area networking both for programming and running the PLC, learning and using a highly developed human-machine interface (RSView32), experimenting with the concept of simulation of the real world (via the debug file in RSEmulate), demonstrating in real time one's work to the professor and being ready to answer questions about that work, appreciating the versatility of the PLC to do control and measurement of analog and digital systems, implementation of classroom/textbook concepts such as PID control in a real-world system with relative ease (RSLogix has a PID instruction), creation of professional-looking technical reports and the satisfaction of completing, as a result of much work, a successful project such as temperature control—and the realization that one is then just a few steps away from being able to succeed in a more elaborate project typical of post-graduate work either in industry or graduate school. The present paper expands on these ideas and briefly presents the three projects assigned to students for learning to use a PLC.

## Background

At Grove City College, the senior year has a two-semester course sequence on control systems (typical student class size is 15 – 25 students). The main topics covered are modeling, lead/lag Bode design[1], the programmable logic controller (PLC)[2,3], digital control[1,3,4], state-space representation[1], state-space pole placement[3,4], optimal control[3,5], and analytical robotics[6], with introductory presentations on fuzzy logic[3] and neural network control[3]. Within the senior lab course, there are five-week lab sequences on signal processing[7] (using SigLab data acquisition), digital communication systems (two five-week sequences), microwave/waveguide systems, analog communication systems, microprocessors, and analog/digital control servo systems. Within the control system lecture courses, the PLC is taught in lecture and experienced hands-on by the students in three projects described later in this paper (one in the fall semester and the other two in the spring semester).

Some of our graduates have communicated back to us that they have jobs working with PLCs and students show a lot of interest in PLCs in class because of their practical, hands-on nature and because of the programming interest and challenge they offer. Our PLC is an Allen-Bradley SLC 5/05 and for programming we use Rockwell Software applications: RSLogix500, RSLogix500 Emulate, RSLinx, and RSView32. The PLC is connected to the college Ethernet LAN. The department has several PC workstations set up with the licensed software and they can be accessed after hours via Remote Desktop from the students' dorm rooms.

Our usage of the PLC began with an automated greenhouse temperature control system that was a senior design project. The greenhouse is a plexiglass/wood scale model about 4'x4'x2'. RSLogix500 ladder logic programming was used to program the SLC 5/05 to open/close louvers, turn on/off circulation fans, turn on/off exhaust fans, and turn on/off heaters to control the temperature, which is measured by three RTD sensors (a fourth sensor outside the greenhouse measures the ambient temperature). The set point is entered via RSView32 and real-time trend plots are generated as well as an animated state-of-system graphic.

**Instruction**

The software is taught to the students in lecture and a hardware demonstration is presented using the classroom projection system and Remote Desktop to a PC with the software. The lectures provide instruction on many important aspects for learning to understand and use PLCs. The following summary of topics covered provides a well-rounded view of both PLCs in general and SLC500 programming in particular:

- A brief historical context of the PLC is given
- The Rockwell Software CD "An Introduction to Industrial Automation" is presented
- The unique concepts of basic ladder-logic programming are reviewed
- The three-stage sequential operation of PLC programs is discussed
- The equivalence between ladder logic and digital combinational logic and state machines is emphasized to help students connect this unfamiliar programming method to more familiar materials they have learned in prior courses
- The proper channel to use is selected (e.g., Ethernet versus the internal connection to a software emulator, or an RS232/DF1 serial connection with a hand-held terminal or direct serial communication between the PC and the PLC). The Ethernet connection naturally provides an opportunity for briefly discussing IP addressing
- The status of all channels is displayed in RSLinx
- Diagnostic information including faults is included in the Processor Status window
- The various input/output modules in the PLC rack and associated data files are described
- Software associated with the power supply module performs a "budget" calculation concerning whether the power supply has sufficient current capacity to power the system; the students are told that such a budget is crucial to calculate when designing/selecting power supplies for their senior projects, which are in their early stages at this point in the academic year.
- Addressing syntaxes for input and output data files are presented.
- The usage of data files both for data and for device configuration are discussed.

- The specification and operation of timers, counters, and the control data block for the PID command are considered, as well as a detailed study of the RTD sensor specifications booklet including 2-, 3-, and 4-wire connections.
- The following important tools for successfully programming and using the PLC are presented:
    - uploading/downloading of programs
    - cross referencing
    - single-stepping/single scanning of programs for debugging (including the somewhat tricky testing of programs that have timers),
    - title/rung comment documenting
    - program version labeling with version notes
    - symbol/description variable labeling
    - navigational/search features
    - desktop workspace customization
    - extensive reporting options
    - backup/archiving options
    - custom data monitoring/multipoint monitoring in which selected variables can be monitored simultaneously and in some cases changed in value in a special small window
    - database management
    - rung- and project-verification
    - input/output forcing as a data-table-overriding feature
    - the multiple means of obtaining help (including annotations of the help files)
    - drag-and-drop of instructions and addresses and mouse-driven rung construction show the students that the software is modern.
    - Emulation: The emulator is presented as an essential means of debugging programs as well as a way that industrial engineers can avoid shutting down systems during software development as they mimic the real-world system in software. In the classroom setting, the hardware is turned off at night to force students to use the emulator for program development and to avoid resource contention.
    - Online edits: Another way that PLC engineers can reduce shut-downs of their system is via online edits (actually adding and/or modifying programming rungs while the program is running), which are demonstrated in real time in lecture.
    - The "Histogram"—poorly named—is discussed and used to show real-time waveform time plot generation.
    - Using DDE/OPC functionality, numerical values of variables displayed in Excel files update live according to their changing values within RSLogix500 in a demonstration that students find quite interesting.
    - Report generation is discussed and illustrated live; within RSLogix500 there are many options for creating professional-looking project reports.
- Of course, the special features of ladder logic that allow one to read, understand, and write a program are described in detail. One theme that shines through the discussions is the high level of organization and documentation that is desirable and possible in RSLogix500, which is worth emphasizing given the large number of variables, devices, and even PLCs that are present in real-world industrial applications of PLCs.

- Following a detailed presentation of RSLogix500 programming, attention is then turned to the human-machine interface software RSView32. This application allows real-time monitoring and changing of variable values within the SLC 5/05 in a highly graphical, user-friendly environment by means of data tags. Features of the application such as trending (multi-variable real-time temporal plots), data logging, alarm setup, display, and acknowledgement, and animation of variable values (e.g., representations of operating gages) are demonstrated live in the classroom and these generate a lot of student interest. For students using LabVIEW, learning RSVIew32 (as well as SigLab and Simulink) can prove helpful for getting started. Just as was done for the RSLogix500 software, many details of programming are covered in real time in class, such as:
    - the Program Manager
    - channel and driver specification
    - scan class selection
    - the tag database and tag monitor
    - activity and data log setup
    - the method for adding tags and testing communications
    - creation of graphics
    - buttons/their functionality
    - trends
    - animation
    - window navigation and printing

With the major programming issues having been discussed in detail, an example program is presented. The program is entered into ladder logic a rung at a time as the logic behind it is discussed, wherever possible with student participation. The project presented is a full-step stepper motor controller. The joke is made that this stepper motor controller is a rather expensive one, but everyone realizes that the selection of this program brings together several important aspects of PLC programming. These include:

- the operation of timers
- bit- versus word-length instruction
- scan order and scanning operation of the PLC
- examination of stepper motor switching tables and converting them into simple logic and then implementing that logic in rungs
- use of solid-state isolated relays and need for power buffering between weak control signals and the powerful equipment they control
- online editing to change the direction of the motor via rung replacement.

One interesting side topic is the use of protective diodes. In Fig. 1a, in the steady-state ON condition, $i = 5\ V/R_L$ and $v_L = Ldi/dt = 0\ V$.

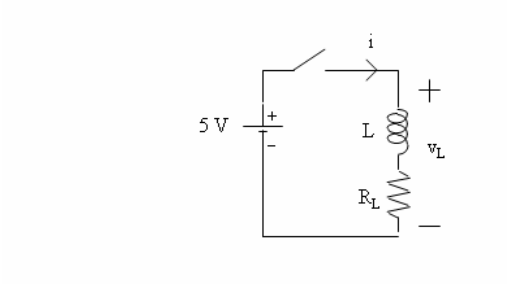


Fig. 1a Lossy inductor driven by 5-VDC voltage source.

When the switch is opened, i immediately drops from 5 V/$R_L$ to zero, so that di/dt and thus $v_L$ are huge negative. The large spike across the electronic switch (solid state relay) could be harmful to it. A solution shown for the stepper motor switch network is shown in Fig. 1b.

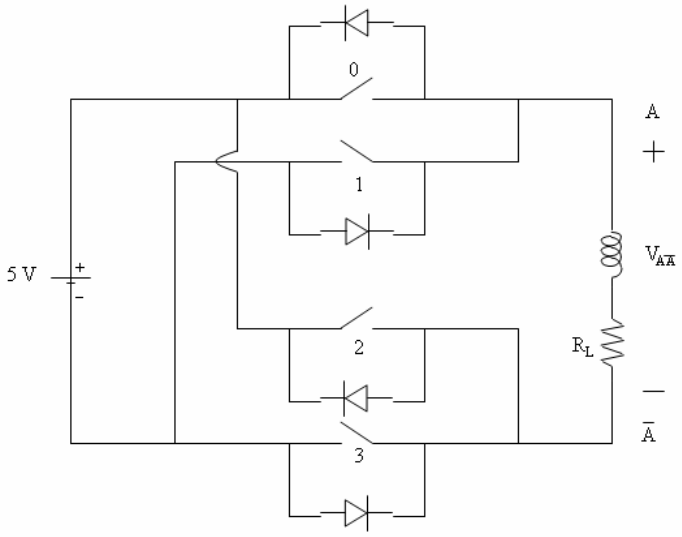


Fig. 1b Stepper motor switching circuit with protective diodes.

In the PLC code, switches 0 and 3 open before 1 and 2 close. Thus with $V_{AAbar} << 0$ V, there is, via diodes 2 and 1, already in place a voltage-limiting conduction path to discharge the inductor energy. With contact potential of 0.6 V per diode, $V_{AAbar}$ is thus safely limited to –(5 + 1.2) V, and all associated solid state relays are protected. Figures 2a and 2b show, respectively, the situation before and after addition of the diodes via oscilloscope traces of a PLC-driven stepper motor waveform. Notice that the spikes are of the polarity that Lenz's law predicts ($v_{self} << 0$ upon opening of switches 0 and 3), with exceptions when there was probably some contact bounce that was captured. Students often use H-bridges in their senior projects, and this discussion helps them see why the application circuits include such diodes.

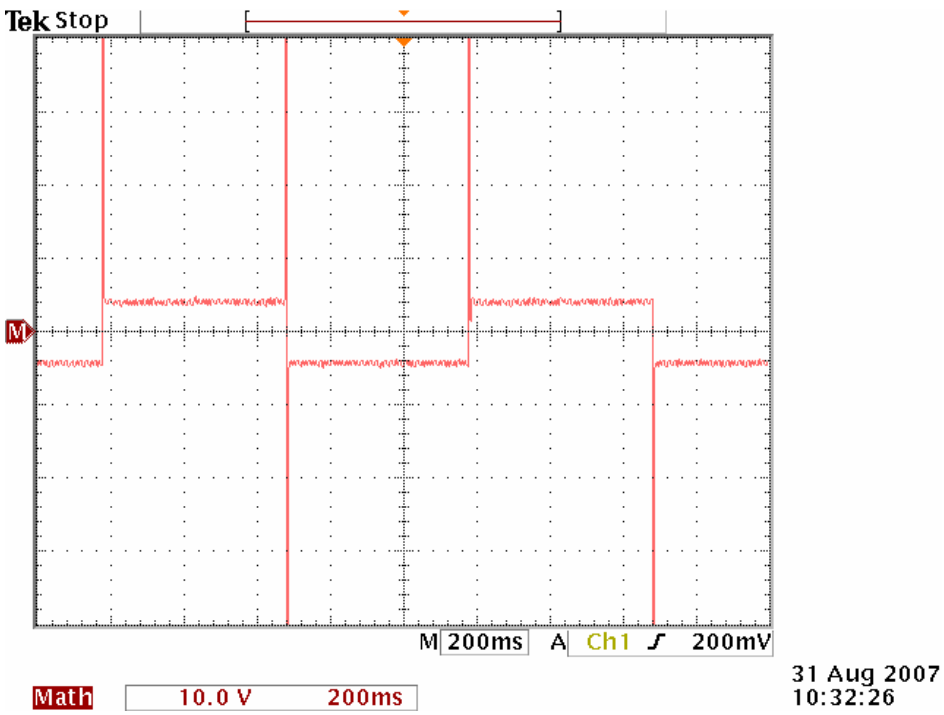


Fig. 2a Actual PLC-driven stepper motor coil voltage waveform before addition of diodes.

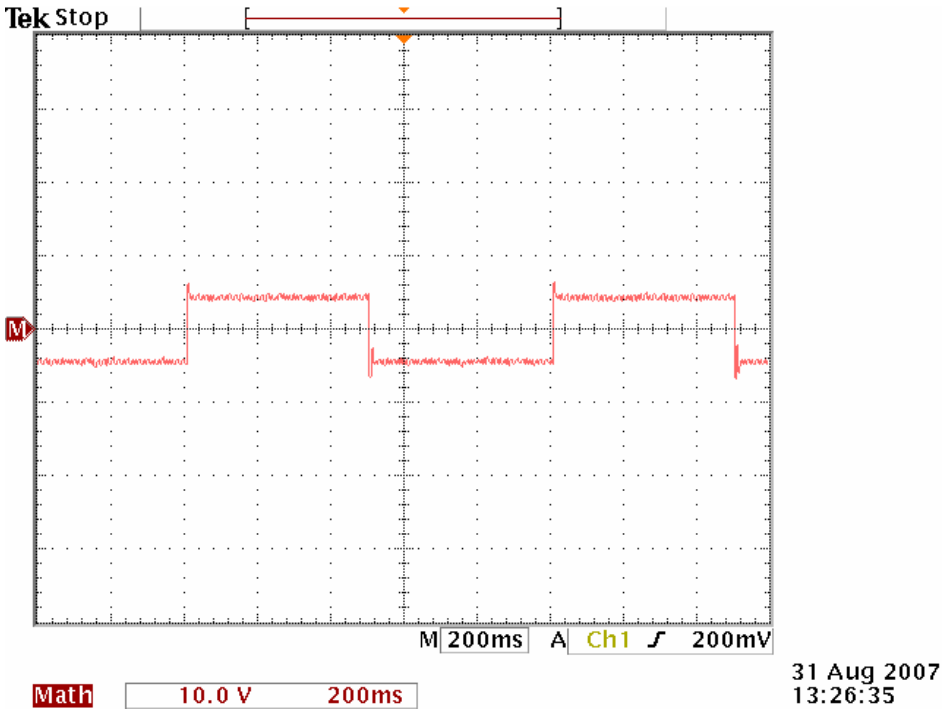


Fig. 2b Actual PLC-driven stepper motor coil voltage waveform after addition of diodes.

Another interesting problem in this project is the limitation on "Histograms" (time waveform plots). If the stepper motor speed is low, the binary coil waveforms appear exactly as expected, but if the motor speed is increased (by means of a timer preset setting), the "Histogram" waveform pulsewidths may become distorted. The problem usually arises from inadequate sampling or from Windows processes interfering to result in irregular sampling—the latter of which may be more likely in cases where sometimes the plotted waveform is normal and sometimes it is distorted. Observation of the oscilloscope waveforms shows that the actual binary coil waveforms still appear exactly as expected, despite the distorted waveshapes presented in the "Histogram" window. In other environments such as Matlab's Real-time Windows Target, the kernel takes over the PC, over Windows, and so the desired sampling interval is always maintained (though of course nothing can correct for too long a sampling interval).

**The Student PLC Projects**

Three projects have been assigned to students: A half-stepping stepper motor controller, a lead screw/limit switch control device, and a closed-loop heating/cooling temperature control system.

The full-step stepper motor controller presented in lecture is a good basis from which students can create the code necessary to drive the stepper motor with half-stepping waveforms, yet the requirements for half-stepping do provide a moderate programming challenge to the students. Details such as the fact that "A bar" is no longer necessarily the logical complement of A because both "A" and "A bar" may be required to simultaneously be zero, the determination of the speed in rpm from a knowledge of the degrees per full step and parameters within their program (speed was not addressed in the lecture demo), the fact that some instructions are word-level rather than bit-level, and so on force students to think through the given problem carefully. A wide variety of approaches has been used by students to achieve half-stepping, including bit-shifting, direct writing of the switching table into code for each time interval, logic/range testing for each switch individually, and use of the sequencer output (SQO) instruction. Similarly, counting up/counting down are implemented various ways, such as running a counter up or down, subtracting the preset from an only up-counting counter, or just using a different set of rules on the upward-counting variable from that for the down-counting variable to determine which solid-state relay switches must be closed. Students are required to plot out "Histograms" of the four stepper motor coil waveforms for both clockwise and counterclockwise operation. They must also predict by calculation and verify by measurement the achieved speed of the motor. They are required to meet or exceed a specified speed, which is possible only if their code is reasonably efficient. For example, certain ways of writing code that involve integer division of preset times lead to failure of the motor to spin rapidly (i.e., at low timer preset values) or even work at only a few different speeds (associated with the different selectable time bases) if the code depends on a timer value rather than a counter value. During the fall 2007 semester, a student found a bug in his code when his code caused the motor to run too slowly.

For the second project, a DC motor-driven lead screw moves a small block along the lead screw as the latter rotates; see Fig. 3. Upon pressing of a pushbutton, the block is driven to a limit switch and upon hitting it, the motor must be reversed (same speed in the opposite direction) and the block is to be returned to its original position; there is not a second limit switch at the original

position. The starting position of the block may be varied from run to run so that the time from start to limit switch is unknown to the programmer; the block must always be returned to the original position for that run. A timer is used within the PLC program to determine how long to move the block back after the limit switch is hit. That part of the programming is reasonably straightforward.

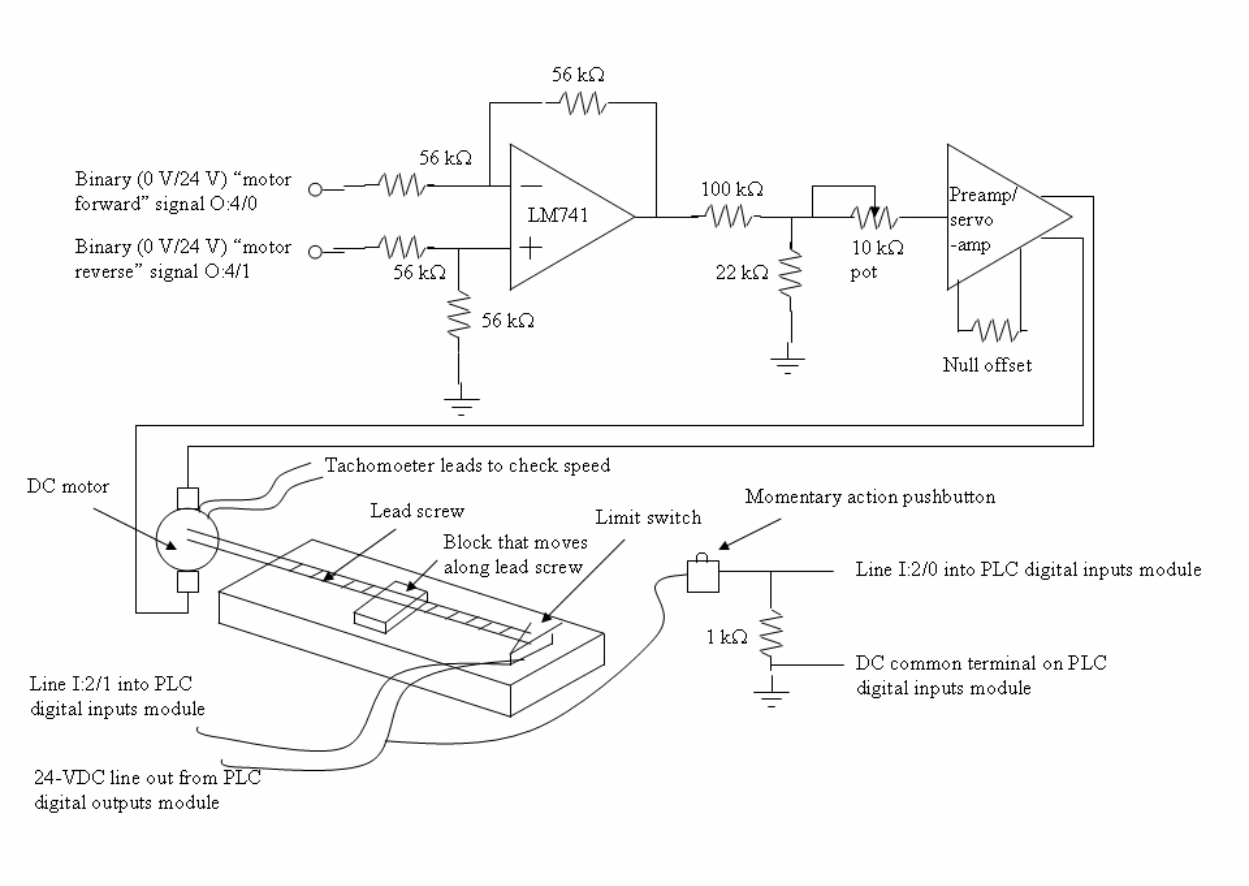


Fig. 3 Circuitry for PLC project with pushbutton-initiated traversal of block along lead screw to end, sensed by limit switch, upon which block is returned to arbitrarily selectable starting position.

Many students have a little difficulty with using the debug file properly within RSEmulate, in which the emulator and not the hardware is used. The debug file is a model of the outside world providing sensor data to and being affected by the PLC and its peripherals—in this case, the limit switch and the motor/lead screw. Despite being warned to base action only on actual outputs and that the only connection between the PLC main program and the debug file is to be inputs/outputs, students initially have a tendency to, in their debug file, use internal variables within their PLC ladder logic file, which are not available in the real world. They must view the

PLC ladder logic file / emulation debug file interface as the same as that between the PLC and the real world. The PLC output (motor drive signal) is the input to the debug file and the PLC inputs (the pushbutton and limit switch states) are the debug file outputs. Thus in the debug file, upon start, the pushbutton switch is to be held high for about one second, mimicking how long a person holds down a pushbutton. The PLC receives this input (in emulation) and turns on the forward motor output. The debug file senses the forward motor output signal high and begins a timer, with its preset set to a time that one would expect the block to typically take to move from its original starting point to the limit switch, e.g. 10 seconds. Upon conclusion of that traveling time interval, the debug file asserts the limit switch for about 1 second (by running another timer) and the PLC senses that and reverses the motor direction (meanwhile, with its own timer, the PLC program has effectively measured the preset of the debug file timer). In this way, one has been able to try out the PLC ladder logic program in which the real world to which the PLC interfaces is well modeled by the debug file.

The final PLC project is a PID-controlled temperature control system, which makes use of the PID instruction in RSLogix. A water-filled metal cylinder with platinum RTD sensor is attached to a heating bar, which has a voltage-controlled heating element, and a small computer fan whose voltage is continuously variable is used for cooling (one can actually hear the changes in the signal controlling it, as the fan speed changes). For the particular heating/cooling elements, the time constants for heating and cooling are respectively about 15 minutes and 5 minutes. A schematic diagram of the hardware is shown in Fig. 4.

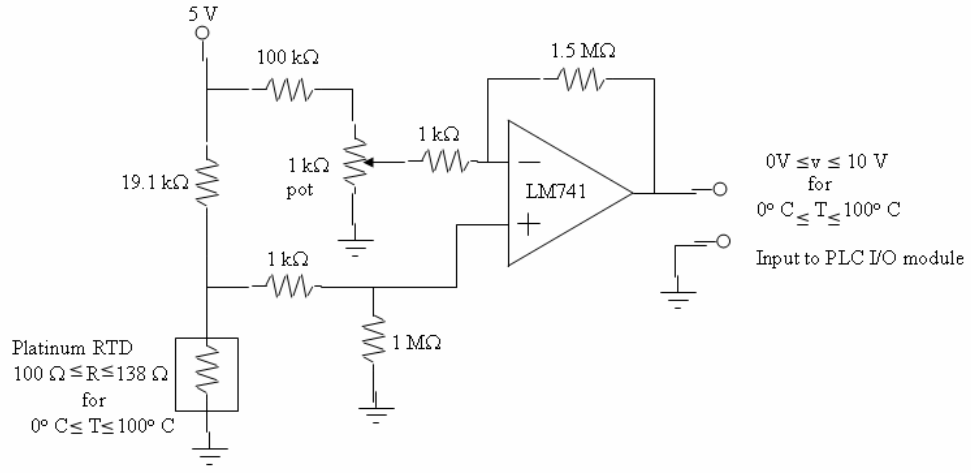


Fig. 4a RTD sensor signal conditioning circuit for PLC temperature control system project.

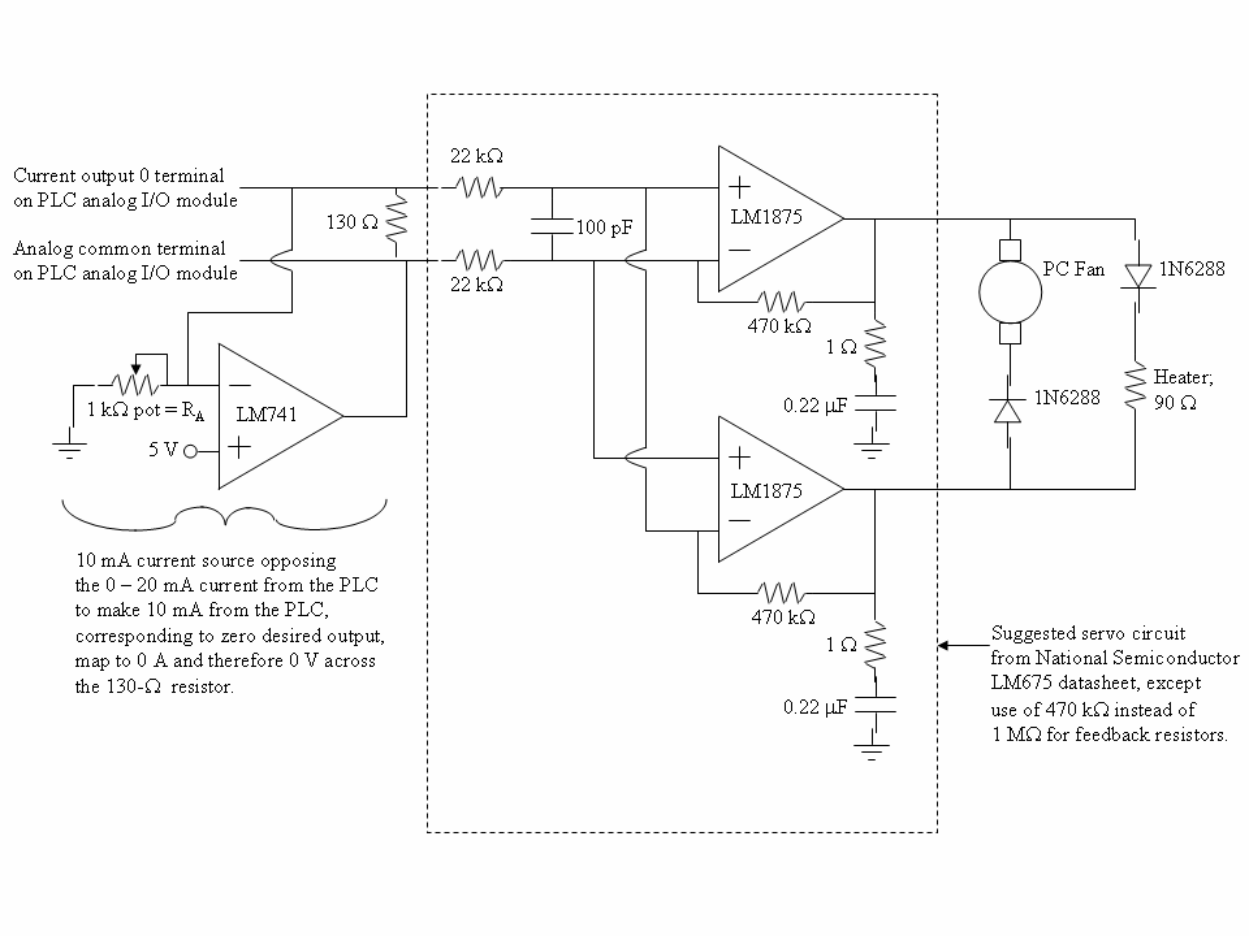


Fig. 4b Heater/fan output circuits for PLC temperature control system project.

Students must do a little digging to properly set up for PID control in RSLogix 500. A control file dedicated to the PID instruction must be created and appropriate integer variables used for the set point and other variables. The PLC instruction has only nonnegative integer output values, yet in order for the final power amplifier circuitry to function with only one PID instruction, bipolar results are required. In addition, the analog output for our analog I/O card is a current output rather than a voltage output. The bipolar requirement was achieved by constructing a current source to subtract off enough current to create negative or positive actuating voltages to be amplified with a bipolar power amplifier. A bias must be added onto the output number from the PID calculation; otherwise RSLogix outputs zero when a negative value would normally be generated.

First the students are assigned to perform proportional-only control with a moderate value of gain such as 10 or 15; in the example below, 10 is chosen to provide a relatively marked performance improvement using PID compared with proportional-only. This gain value provides a modest amount of overshoot and a steady-state error of about 2.0 degrees Celsius for a set point of 35 degrees C (95 degrees F) under proportional-only control. Students are to evaluate the percent overshoot, the settling time, and the steady-state error from their plots. Using RSView32, floating-point temperatures in degrees Celsius are to be displayed both in numerical fields and in real-time trend plots that show desired/actual temperatures and an

appropriately scaled version of the actuating signal all as functions of time. Contrarily, positive integer values for temperature are required in the PID instruction, corresponding to the valid range of 0 to 16383. Students must be able in RSView32 to enter desired temperature in degrees Celcius, print out the plot using a print button they create, and have a "home page" and trend page that can be navigated by buttons.

Next, students are asked to try PID control, which has already been discussed in class. Students must recognize that the default in RSLogix500 is to differentiate the output without negation, whereas conventional PID control has the derivative of the error signal (which negates the derivative of the output). One often omits including the set point in the derivative calculation to remove spikes when the set point is changed suddenly, but having the wrong polarity on the derivative term of the process variable (both seen in the RSLogix documentation and observed in actual system operation and Simulink simulations) is known to increase overshoot or even destabilize the closed-loop system—defeating the purpose of the derivative control term. The particular temperature control system studied is so sluggish that the derivative term has been found to make little difference one way or the other for the usual PID parameter values, but it is important for the student to learn the correct polarity.

Either a seat-of-the-pants or a more systematic approach is done to tune the PID gains. One way of tuning the parameters is to perform open-loop testing and follow the Zeigler-Nichols tuning rules. In this particular application example, the Zeigler-Nichols rules (performed for heating) turn out to result in greater overshoot and oscillation than desired, but at least they give an order-of-magnitude starting place for a parameter search for "best" PID performance. In particular, the integral reset time of 6 minutes and the derivative scaling time of 2 minutes with a proportional gain of 10 were selected instead of the Ziegler-Nichols values. For use of the PID instruction with these tuning parameters, the steady-state error was indeed practically eliminated with only a modest amount of overshoot., as shown in the example PID run presented below.

A typical set of trend plots is as follows. In Fig. 5, an open-loop test is performed to obtain an estimate of the delay and rise/fall times of the system (about 2 minutes and 22 minutes, respectively for heating; negligible delay time and only about a 6-minute fall time for cooling), which are used to generate Ziegler-Nichols initial PID test parameters ($K_c$ = 17 [using the average of rise/fall times], $T_I$ = 4 min., $T_d$ = 1 min.) . The maximum temperature at saturated output for this heating system is approximately 48 degrees C and the minimum is of course the ambient room temperature.

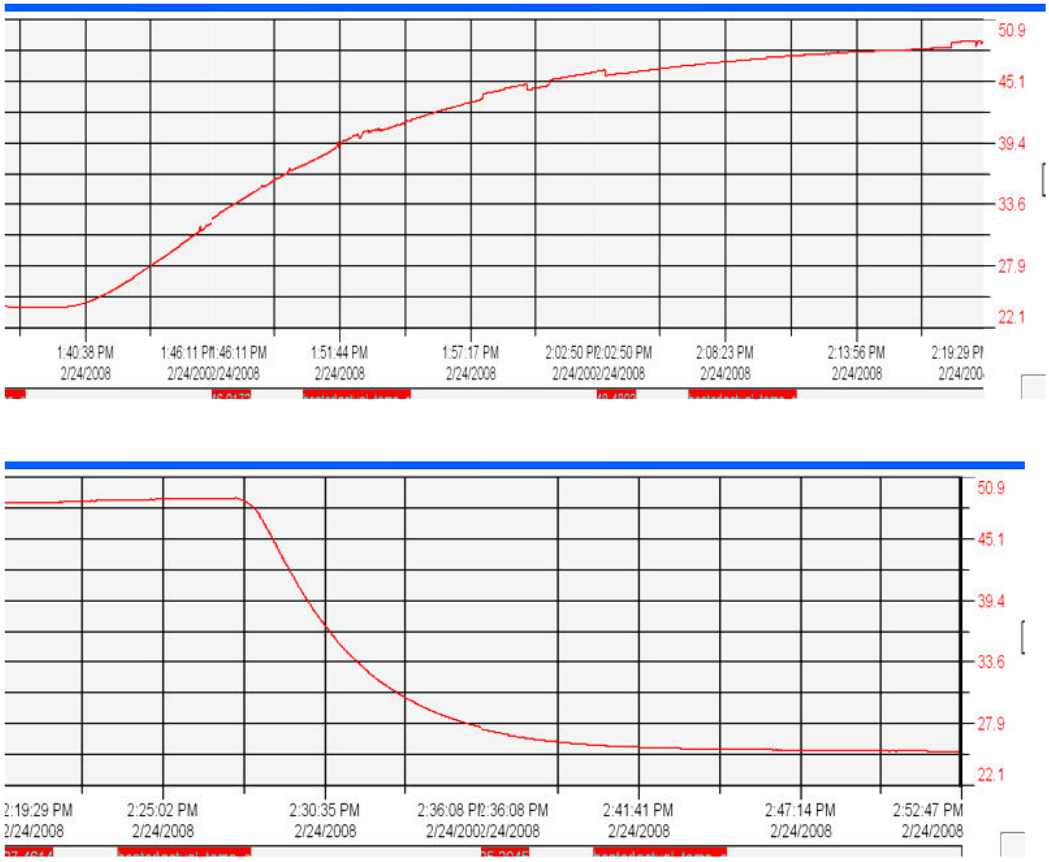


Fig. 5 Open-loop heating/cooling test. Waveform shown is actual temperature in degrees Celcius versus time.

In Fig. 6 is shown a warming cycle for proportional-only control. The final value of temperature is about 33 degrees C, which corresponds to a steady-state error of 2 degrees C. The control signal is the one beginning saturated and then falling below saturation and mildly oscillating; the bar temperature is the rising waveform.

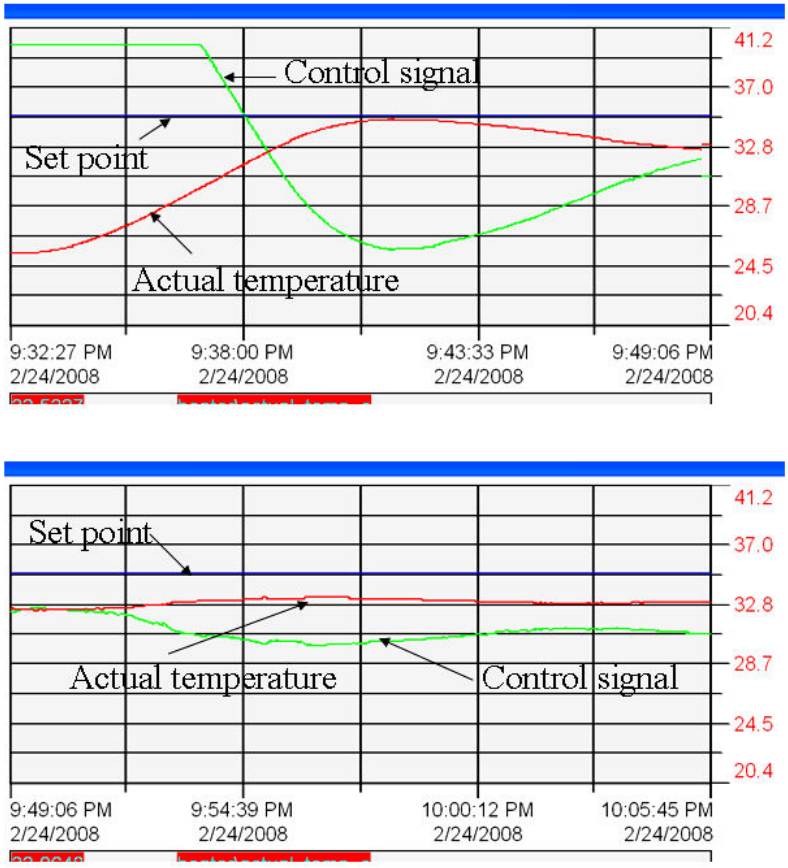


Fig. 6 Heating cycle for proportional-only control ($K_c = 10$) for constant 35-degrees Celsius set point.

With PID control, the steady-state error can be eliminated, as shown in Fig. 7 using the non-Ziegler-Nichols PID parameters discussed previously ($K_c = 10$, $T_I = 6$ min., $T_d = 2$ min.).

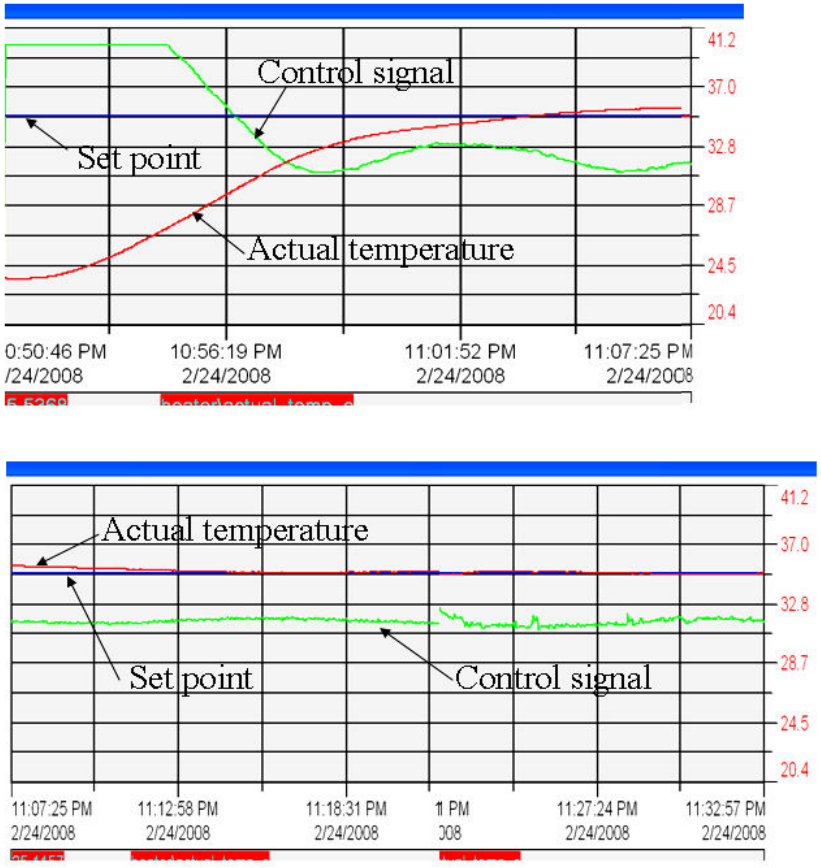


Fig. 7 Heating cycle for PID control ($K_c$ = 10, $T_i$ = 6 minutes, $T_D$ = 2 minutes) for constant 35-degrees Celsius set point.

**Side Benefits of the PLC Projects**

As well as being a multifaceted programming experience, the student work on the PLC forces students to adopt some important behaviors. Students must also plan properly and treat their classmates with respect in a situation of limited resources. Grove City College has purchased several copies of the software but has only one PLC. Students are able to log in remotely at night to lab PCs that have the software on them to develop and test code (using the emulator). As only one student can use a PC at a time, there must be time-sharing and such etiquette as being patient with each other when one is on a machine that someone else wants to use, not locking up the PC under his/her name, remembering to log off, refraining from using remote login for many hours at a time during the day (when there is physical access to the lab PC), scheduling their hardware usage and demonstrations in harmony with each other, leaving the lab when another student is demonstrating her/his project so that there is a level playing field of evaluations, heeding warnings to avoid certain actions that could jeopardize the system, etc. Personal integrity is essential and emphasized, and code is checked for originality. Integrity is

also assessed via the demonstrations, where students must not only demonstrate that the hardware works according to stated specifications but also explain how their code works ("walk the instructor through their code") and answer questions about the code and the assignment. Also, PLC problems may appear on exams (e.g., programming a washing machine cycle or other process control sequencing, concepts or terms presented in lecture, etc.) that expose how well students have been learning and how independently they have been working. Planning and the avoidance of procrastination are also critical in order to avoid "crunch-time" near the due date of the projects. Students are given the assignments at the beginning of the semester and have a very large amount of time (months) more than is necessary to complete the projects. If substantial procrastination occurs among many students, then there can be contention and/or missed deadlines. Thus even though the projects are not team efforts, important interpersonal skills are developed through the assignments. Although all working code is accepted, excessively long code may be penalized; a "Lord of the Rungs" award is given for the most concise and/or effective code.

In addition, students must become resourceful in doing these projects. Although all of the basics are presented thoroughly in lecture, implementation details must be found in the extensive documentation, which has been placed in a folder on the PCs containing the software (in the form of PDF files). In students' work after graduation, often they will be required to learn on their own how to use software. Through the trio combination of Matlab and PLC assignments as well as senior design projects, students have ample opportunity to become resourceful in extracting relevant information from large volumes of documentation. In all these three endeavors, students must use industry-standard software with far more capability than is used in the assignment, as opposed to watered-down, simplified software that sometimes comes in textbooks. Again, alumni periodically note that they have used specific software such as the PLC software in their postgraduate work; having already used such software in an educational setting can clearly be highly advantageous later.

**Student Opinion and Conclusion**

The first semester that the PLC projects were offered, the percentage of student evaluation comments that were negative was very high—71% negative, but improvements were made quickly. For the ten semesters since then, the percentage of student evaluation comments that were positive has been very high—88% positive; these exclude 9 student comments in which students asked for more lab stations with the software, a situation that has since been rectified by purchasing more Rockwell Software licenses. Some of the words that students have used in their course evaluation comments to describe the PLC projects are: cool, great, practical, relevant, informative, challenging, fun, interesting, liked, enjoyed, helpful, educational, worthwhile, a nice break from usual assignments. In addition, an employer survey (2002) listed "PLC troubleshooting" as a technical skill especially desirable for electrical engineers in his/her field. In alumni surveys (2005, 2002 graduates), "PLC background (projects and lectures) was excellent—used PLCs briefly at my time at (student's company name) working with some automated welding robots" and RSLogix was listed by two alumni in their postgraduate work; a current student notes that in his job after graduation he will be using PLCs for dairy processing and he said he is excited about doing the upcoming temperature control project.

## References


1. N. S. Nise, *Control Systems Engineering, 5th Ed.* New York: Wiley, 2008.

2. Rockwell Software, Inc., numerous on-line guides and reference manuals.

3. T. J. Cavicchi, *Lecture notes for ELEE421/422*. Grove City College.

4. C. L. Phillips, H. T. Nagle, *Digital Control System Analysis and Design*, 3rd edition. Englewood Cliffs, NJ: Prentice Hall, 1995.

5. P. N. Paraskevopoulos, *Modern Control Engineering*. New York: Marcel Dekker, Inc., 2002.

6. S. B. Niku, *Introduction to Robotics.* Upper Saddle River, NJ: Prentice Hall, 2001.

7. T. J. Cavicchi, "Experimentation and analysis: SigLab/MATLAB data acquisition experiments for signals and systems, *IEEE Trans. on Education,* Vol. 48, No. 3, Aug., 2005, pp. 540-550.