Proceeding of the 2004 American Control Conference
Boston, Massachusetts June 30 - July 2, 2004

An Educational Testbed for Design and Implementation of Computer Control Software

M. Moallem
mmoallem@engga.uwo.ca
Department of Electrical & Computer Engineering
University of Western Ontario
London, Ontario, N6A 5B9
Canada

Abstract— **There is a growing need in industry for engineers who can perform software design and system integration for various applications in embedded control. While courses offered in the Electrical & Computer Engineering disciplines cover such topics as microprocessors, digital and analog hardware, control theory, and programming languages, there are few courses that focus on integrating these subjects for designing embedded systems. This paper describes a laboratory testbed developed for a new course on Embedded Computer Control offered at the University of Western Ontario, Canada. The embedded controller performs command, control, and user interface tasks required to operate a low-cost prototype of a thermal system. Furthermore, its network connectivity allows users to tune system parameters, start and stop running the system, and observe the status of the plant via the Internet.**

Index Terms— **Computer control, real-time control, command and control systems, embedded computing, modular design, network enabled systems.**

I. Introduction

Embedded computing is an enabling technology dealing with the engineering of computer systems used in such areas as command and control, communications and multimedia, and information systems. Emerging technologies such as Micro Electromechanical Systems (MEMS), wireless ad-hoc networks, smart homes and work-spaces, intelligent highways, medical mechatronic devices, etc, would require use of even more embedded computers. The opportunities seem to be endless and limited only by creative designs and novel technologies.

There are certain challenges in developing efficient embedded computer systems. First, due to their nature, the design of embedded applications often require multidisciplinary and cross disciplinary skills. Secondly, many embedded applications are real-time, meaning that the correctness of computations is as important as their timeliness. Thirdly, modern embedded applications have sophisticated functionality, requiring execution of concurrent tasks and intensive algorithms.

With regard to control systems, Boasson [1] analyzed software design issues for complex control systems, emphasizing the need for highly modular and expandable software architectures. Stewart *et al.* [8], [9], developed a real-time operating system, called *Chimera*, for reconfigurable sensor-based control systems and introduced the concept of port-based objects (PBO)– a software frame-work that supports the design and implementation of sensor-based control systems. Real Time Operating Systems (RTOS) have been utilized in several control applications to implement modular architectures (see e.g., [6], [5]).

With the ever increasing role of embedded computing in today's applications, there is a growing need for introduction of new courses in Electrical, Computer, and Software Engineering disciplines. In this regard, such courses should focus on integrating design techniques with concepts introduced in earlier courses. In this paper, we present details of a laboratory project designed to accompany a course on embedded and real-time computer control. The project is designed to provide an opportunity for students to integrate topics covered in the course with previously taken courses in programming and computer science, signal processing, controls, microprocessors, and electronics.

This paper is organized as follows. Section II describes the laboratory setup, course structure, and the capstone project. Section III discusses design issues in developing software for the system. In section IV, we present some results on incorporating the system into a course project and feedback received from students.

II. Laboratory Setup

The objective of the project is to introduce procedures for designing embedded computer controlled systems. The students are required to design the software and build a sensor-based feedback control system, using a Proportional Integral Derivative (*PID*) controller with existing components. Two main references were selected for this purpose, i.e., the textbook by Wolf [11], which focusses on embedded systems design techniques, and a technical report by Wittenmark *et.* al [10], which discusses practical issues such as discretization, anti-aliasing filters, selection of sampling rates, and integrator windup.

The system is depicted in Figure 1 and is comprised of a tube, consisting of a DC fan (used for cooling desktop

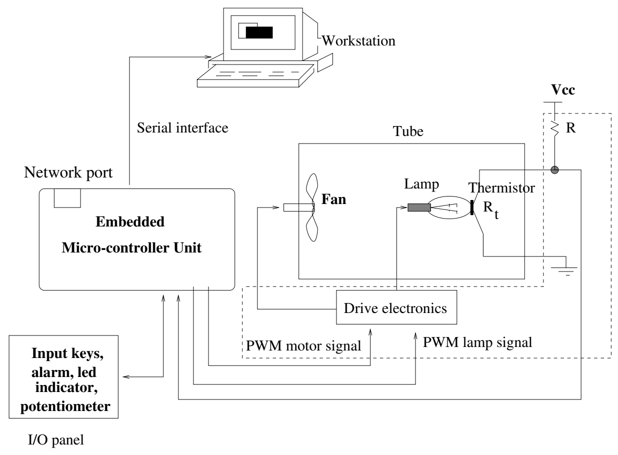


Fig. 1. Components of the temperature control system.

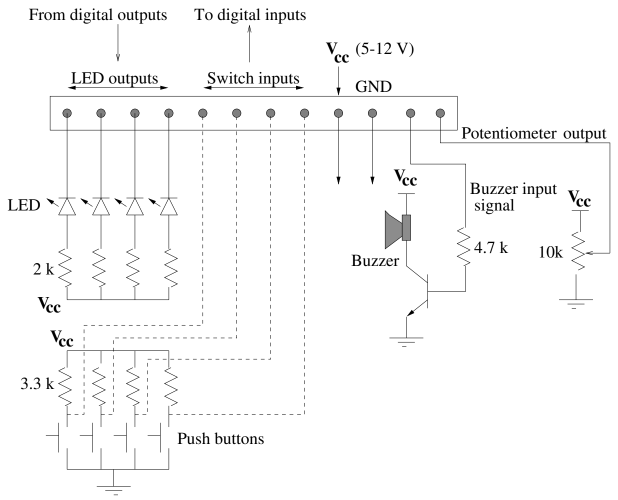


Fig. 2. Circuit diagram of the I/O panel board.

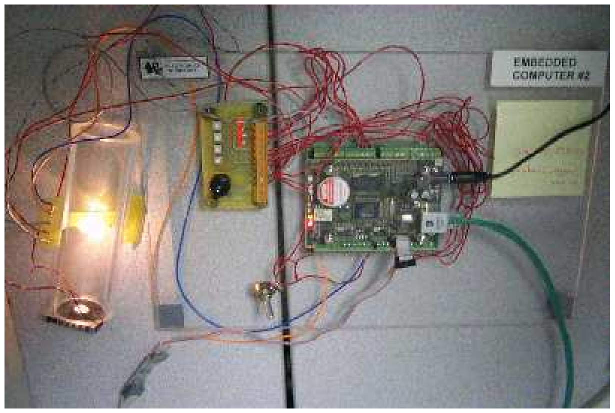


Fig. 3. Embedded system hardware components.

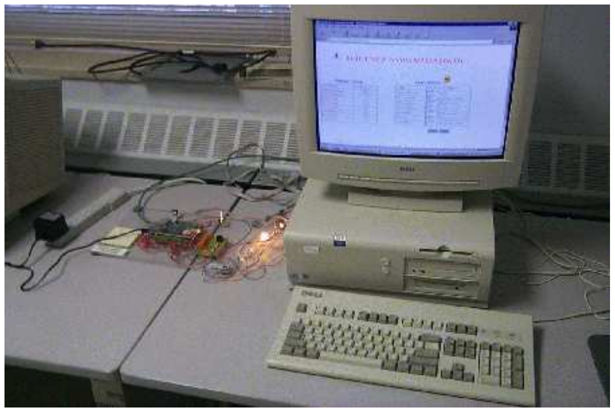


Fig. 4. Embedded system controlled and monitored using an Internet browser.

CPU's), a lamp (for heating the tube), a 10 $k\Omega$ thermistor for measuring temperature, an I/O panel consisting of four push button switches, a buzzer, a potentiometer, and a microcontroller development kit (from *Z*World* [12]). Figure 2 illustrates the circuit diagram of the I/O panel, consisting of push button switches, Light Emitting Diodes (LED's), a buzzer, and a potentiometer. The potentiometer can be used for different purposes. It can be used as a set-point for temperature, or can be combined with push button switches for tuning the *PID* controller gains. The push button keys are used as digital inputs. Using these keys, the user may turn the system on or off, or set the maximum speed of the fan, or change the PID gains. Similarly, the LED outputs may be used to display the status of switches, and the buzzer can be used to inform the user if the temperature has gone above or below certain limits.

The BL2000 Microcontroller Unit (MCU) from *Z*World* [12] was used, which provides digital and analog I/O, and an Ethernet port on a single-board computer unit. The Ethernet model was selected for remote monitoring and supervisory control. The programming environment uses *Dynamic C*, which is an integrated C compiler, editor loader, and debugger for embedded control and communication applications. The PC workstation in Figure 1 is used as a development host that is connected to the target embedded system by a serial RS232 interface.

In Figure 1, the motor and lamp are driven by pulse-width-modulated signals to control the temperature in the tube. These signals are on the order of a few tens of Hertz, which can be generated by using the digital output bits and blocking delay functions (i.e., *waitfor()* statements in th environment used). The drive electronics consist of two buffers and two CMOS transistors working as ON/OFF switches. Figure 3 shows different hardware components of the system put together. A rough estimate of the budget required for setting up the system is US$200.

The temperature is regulated by turning on either the fan or the lamp based on the voltage developed across the thermistor. The I/O panel and a web browser provide user interfaces to the system and can be used to monitor the temperature of the system, tune *PID* controller gains, and start or stop operation of the system from either the I/O panel or a web browser. The available TCP/IP libraries allow for Ethernet communications and include libraries for

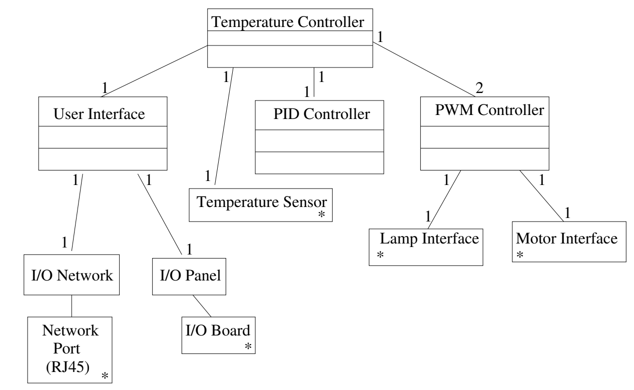


Fig. 5. A Unified Modeling Language (UML) class diagram showing composition of subsystems.

sending email and running *http* servers. As a requirement of the project, the system should send email messages to a user under certain conditions, e.g., to report a failure condition such as a high tube temperature. Figure 4 shows the system under control from an internet browser.

III. Design of the Control System Software

Coding a feedback control algorithm that works properly is often not sufficient. Software design must also take into account such issues as functionality, user interface, safety and reliability, upgradability, power consumption, and system costs.

The first step in the design process is to have a set of requirements. Since some components of the system are already given, the design steps involving specifications and architecture (in particular, hardware architecture) can be performed quickly. The class diagram illustrated in Figure 5 shows components of the software architecture in the Unified Modeling Language (UML) form. The *Temperature Controller* class is comprised of *User Interface*, *PID Controller*, and *PWM Controller* software classes. Special classes representing hardware components are denoted by asterisks. The main module, i.e., *Temperature Controller*, is responsible for initializing other modules, establish links between modules, and run the system tasks.

A. PID Controller

Following [10], a modified *PID* control algorithm that overcomes the conventional *PID* controller problems such as integrator wind-up, differentiation of command signal, etc, is given in the Laplace domain by (see Figure 6)

$$\begin{aligned} U(s) &= K(bU_c(s) - Y(s) + \frac{1}{sT_i}(U_c(s) \\ &- Y(s)) - \frac{sT_d}{1+sT_d/N}Y(s)) \end{aligned} \qquad (1)$$

where K, b, T_i, T_d, and N are controller parameters, and $U(s)$, $U_c(s)$, and $Y(s)$ denote the Laplace transforms of u, u_c, and y, respectively. In order to implement the control algorithm on an embedded computer, equation (1) has to be

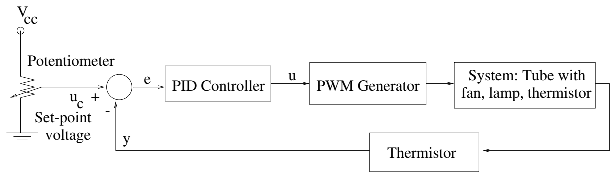


Fig. 6. Block diagram of the temperature feedback control system.

discretized. Denoting the sampling period by T, and using backward differences to discretize the derivative term and forward differences for the integral term, we have

$$u(kT) = P(kT) + I(kT) + D(kT) \qquad (2)$$

where k denotes the k-th sampling instant and

$$\begin{aligned} P(kT) &= K(bu(kT) - y(kT)) \\ I(kT) &= I((k-1)T) + \frac{KT}{T_i}e((k-1)T) \\ D(kT) &= \frac{T_d}{T_d+NT}D((k-1)T) \\ &- \frac{KT_dN}{T_d+NT}(y(kT) - y((k-1)T)). \end{aligned} \qquad (3)$$

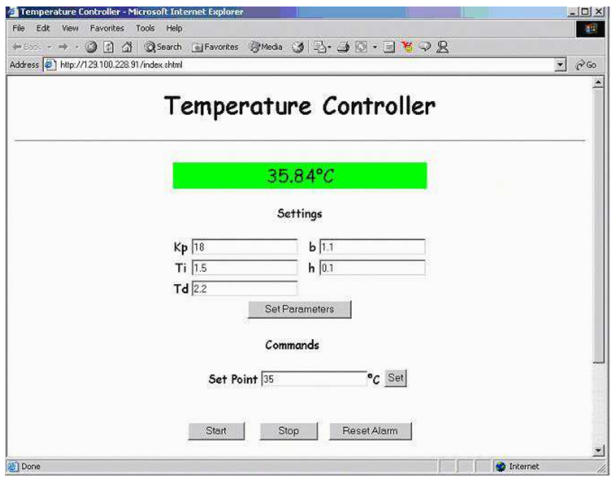
In order to develop modular embedded software in C, one has to use Abstact Data Types (*ADT*). An *ADT* is a precursor to objects and consists of a data structure and a set of methods that operate on the structure. Towards this end, an appropriate data type for the *PID Controller* module in Figure 5 as follows

```
typedef struct {
 float K, b, Ti, T, Td;
 unsigned short N; /* parameters */
 float i, ilast; /* integral term */
 float y, ylast; /* output */
 float d, dlast; /* derivative term */
 float uc;       /* input */
 float u;        /* output */
 float elast;    /* error term */
 float ulow, uhigh;/* low/high inputs
           for windup compensation */
} pid_t;
```

The routines that operate on the `pid_t` data type are used for initializing, updating, and calculating the *PID* controller and take the following general form

```
void initPid(pid_t *p, ...);
void updatePid(pid_t *p, ...);
void calcPid(pid_t *p, ...);
```

The *PID Controller* module interacts with the *Temperature Sensor*, *PWM Controller*, *User Interface*, and *PWM Controller* modules. Thus, appropriate parameters (indicated by $\cdots$) have to be passed to the above routines from these modules.


Temperature Controller - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://129.100.228.91/index.shtml
Go
Set Parameters
Set
Start
Stop
Reset Alarm
Done
Internet

Temperature Controller

35.84°C

Settings

Kp 18 b 1.1
Ti 1.5 h 0.1
Td 2.2

Commands

Set Point 35 °C

Fig. 7. Web browser interface to the embedded system.

B. Other Modules

The *PWM Controller* obtains its data from the *PID Controller* and sends *PWM* pulses to either the fan or the lamp. The *PWM Controller* is implemented through software by sending high and low digital values to a digital output channel. The *User Interface* module in Figure 5 is comprised of two modules: (1) The *I/O Panel* module, which scans the I/O panel for new inputs from the push button switches and the potentiometer, and provides outputs to the LED's and the buzzer, and (2) The *I/O Network* module, which provides an interface to the network port. The TCP/IP protocol suite allows application layer protocols for sending email, transferring files, using the File Transfer Protocol (FTP), and displaying web pages, using the Hyper Text Transfer Protocol (HTTP). The Simple Mail Transfer Protocol (SMTP) is used to send email messages from a client to a Mail Transfer Agent (MTA) (see e.g., [3]).

As a requirement of the course project, the embedded system should send emergency messages to a user if the temperature sensor reads values above or below some critical limits. A typical interface to the *http* server task running on the embedded processor is shown in Figure 7. Using an Internet browser, a user can monitor the tube's temperature, start and stop the system, change the set-point temperature, and tune the *PID* controller gains.

The *Temperature Sensor* module consists of a thermistor in series with a resistor as shown in Figure 1. The voltage v_t across R_t in Figure 1 is read by an A/D converter and can be used to obtain the temperature of the thermistor as follows

$$T = \frac{\beta T_0}{\beta - T_0 ln\left(\frac{v_t R}{(V_{cc}-v_t)R(T_0)}\right)}. \tag{4}$$

where $R_t(T_0)$ is a known resistance at temperature T_0, and β is a constant.

C. Execution of Tasks

Concurrent programming allows for separation of modules into separate tasks and is a key element in modular software development. Analysis of schedulability of tasks is helpful in evaluating performance of the system and can provide guidelines for maximal utilization of the processor. For a priority-based preemptive kernel, the Rate Monotonic Analysis [4] and its modifications (e.g., [7]) can be used to determine schedulability of a task set. For the case where a non-preemptive kernel is used, one has to consider the structure of the code. The cooperative multitasking kernel is used to implement periodic tasks using the following construct [12]

```
main ( ) {
/* variable declarations */
 for (; ;){
   costate Task_1 {
      waitfor(delay(period of task 1);
   /* code for task 1 */
      }
   costate Task_2 {
      waitfor(delay(period of task 2);
   /* code for task 2 */
      }
            ......
   costate Task_N {
      waitfor(delay(period of task N);
   /* code for task N */
      }
   }
 }
```

where *costates* are blocks of code that can suspend themselves, by executing *waitfor( )* statements, *yield* execution to other tasks, or *abort* their execution. In order to provide a timing analysis for the system, let us denote the periods and the worst case execution times of the tasks by $T_1, T_2, \cdots, T_N$, and $e_1, e_2, \cdots, e_N$, respectively. A timer interrupt is used to update a global variable that keeps track of the time. The *delay()* functions in the code listing above use this global variable to identify whether the corresponding task has to be executed or not. Suppose that the timer interrupt runs every T_{tick} seconds and it takes e_{tick} seconds to execute the interrupt code. Let us denote the maximum time it takes for a *for (; ;)*{ ... } loop to be executed by t_x. Then, t_x can be obtained by solving the following equation

$$t_x = \sum_{i=1}^{N} e_i + \frac{t_x}{T_{tick}} e_{tick}. \tag{5}$$

Taking the deadline of each task equal to its period, a necessary condition for all tasks to meet their deadlines is that the period of each task be greater than the worst case execution time t_x of the loop, i.e.,

$$T_j > \frac{\sum_{i=1}^{N} e_i}{1 - \frac{e_{tick}}{T_{tick}}}, \quad \forall j = 1, \cdots, N. \qquad (6)$$

The schedulability condition (6) can be used as a guideline to guarantee that all tasks meet their deadlines. Qualitatively, reducing the execution times and increasing the periods would help satisfy the inequality condition given by (6). Students are required to measure execution times of the tasks, perform a quantitative analysis of the schedulability of the task set, and take appropriate measures if the schedulability test fails, e.g., by changing the tasks' periods or optimizing code for speed.

IV. Discussion of Results

In general, the students' responses to the project were quite positive. Some students from the Electrical Engineering option, who had less background in programming and were not paired with students in the Computer Engineering option, had some difficulty in developing code and testing their programs. The group sizes were typically two with occasionally three students per group. However, majority of students maintained that the material for doing the project was well laid out and that the project had made the course interesting by integrating concepts from microprocessors, programming, control, and electronics.

All the groups managed to finish their projects after six lab hours (two lab sessions). The preliminary lab exercises were helpful in this regard as they were designed to guide the students towards implementing the building blocks required by the project. A flowchart for these experiments is shown in Figure 8 which include writing simple programs to perform cooperative multi-tasking, perform digital and analog I/O, obtain characteristics of the temperature sensor, and run the motor and lamp using Pulse Width Modulation (PWM).

The graduate students taking the course were required to implement the more challenging network interface and email programs. Figures 3, 4, and 7 illustrate systems implemented by students. Other recommended projects could focus on adding an *ftp* server task for streaming sensor data to a host workstation, and the use of wireless technology for remote control and monitoring.

V. Conclusion

Control systems are one of the main application domains for embedded computing. With the complexity of applications growing rapidly, it is very important that engineers designing control systems be educated so that they are aware of the design procedures and challenges in embedded computing technology.

An experimental testbed for design and implementation of embedded control software was developed to serve as the laboratory component of an undergraduate-graduate course on Embedded Computer Control. The course material and project integrate concepts taught in several other courses including microprocessors, control systems, programming and software development, and electronics. The low-cost temperature control platform provides students with a hands-on environment to develop fundamental skills in the design and implementation of embedded computer systems. The opportunity for students to experiment with a network enabled system allows them to get familiar with state-of-the-art technologies such as network appliances and their potential applications in industrial measurement and control.

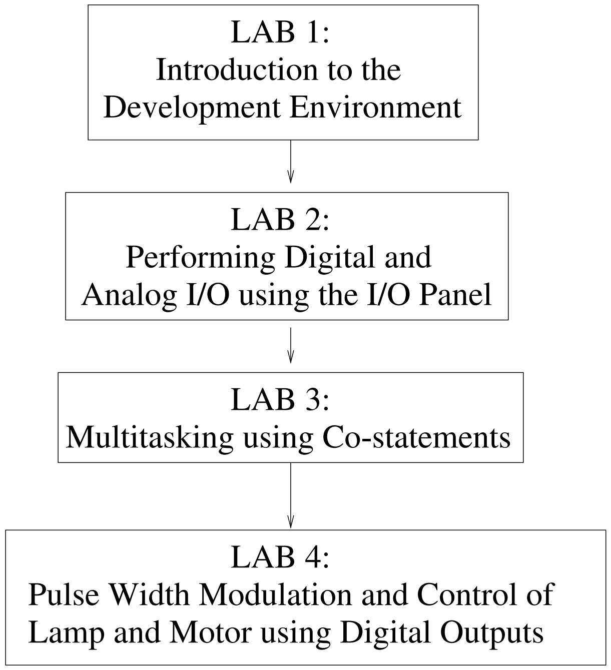

LAB 1:
Introduction to the
Development Environment
LAB 2:
Performing Digital and
Analog I/O using the I/O Panel
LAB 3:
Multitasking using Co-statements
LAB 4:
Pulse Width Modulation and Control of
Lamp and Motor using Digital Outputs

Fig. 8. Flowchart of preparatory experiments.

Acknowledgement

The author would like to acknowledge help provided by teaching assistants, K. Kang, H. Bassan, and P. Yuan, in developing several of the laboratory experiments, and B. Kettlewell, G. Aarsten, and E. Porter of the Electronic Shop at the University of Western Ontario, for building the electronic components of the system. The feedback and constructive comments by students has also been valuable in developing the laboratory.

References

[1] M. Boasson, "Control Systems Software," *IEEE Transactions on Automatic Control*, Vol. 38, No. 7, pp. 1094-1106, 1993.
[2] X. Feng, S.A. Velinsky, and D. Hong, "Integrating Embedded PC and Internet Technologies for Real-Time Control and Imaging," *IEEE/ASME Transactions on Mechatronics*, Vol. 7, No. 1, pp. 52-60, 2002.
[3] M.T. Jones, "Embed with the Mailman," *Embedded Systems Programming,* pp. 45-52, October 2001.
[4] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real–Time Environment," *J. Assoc. Comput. Mach.,* Vol. 20, No. 1, pp. 46-61, 1973.

[5] M. Moallem, “Electron Beam Position Monitoring and Feedback Control in Duke Free-Electron Laser Facility,” *IEEE Transactions on Industrial Electronics,* Vol. 49, No. 2, pp. 423-432, 2002.

[6] M. Moallem, R.V. Patel and K. Khorasani, “Nonlinear Tip-Position Tracking Control of a Flexible-Link Manipulator: Theory and Experiments”, *Automatica*, 37, pp. 1825-1834, 2001.

[7] L. Sha, R. Rajkumar, and S.S. Sathaye, *Proceedings of the IEEE, Special Issue on Real-Time Systems,* Vol. 82, No. 1, pp. 68-82, 1994.

[8] D.B. Stewart, D.E. Schmitz and P. K. Khosla, “The Chimera II Real-Time Operating System for Advanced Sensor-Based Robotic Applications,” *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 22, No. 6, pp. 1282-1295, 1992.

[9] D.B. Stewart, R.A. Volpe and P.K. Khosla, “Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects,” *IEEE Transactions on Software Engineering,* Vol. 23, No. 12, 1997.

[10] B. Wittenmark, K.J. Astrom and K-E. Arzen, *Computer Control: An Overview,* Technical Report, Department of Automatic Control, Lund Institute of Technology, Sweden (downloadable from www.control.lth.se/ kursdr/ifac.pdf).

[11] W. Wolf, *Computers as Components: Principles of Embedded Computing System Design*, Morgan Kaufman, San Francisco, 2001.

[12] Z*World, Inc., Davis, CA, *http://www.zworld.com/company/*.