Embedded Systems Courses at RIT

Roy S. Czernikowski
Department of Computer Engineering
Rochester Institute of Technology
rsceec@rit.edu

James R Vallino
Department of Software Engineering
Rochester Institute of Technology
J.Vallino@se.rit.edu

Abstract

A three-course sequence of cross-disciplinary real-time and embedded systems courses has been introduced at RIT[•]*. We are teaching these courses in a studio-lab environment teaming computer engineering and software engineering students. The courses introduce students to programming both microcontrollers and more sophisticated targets, use of a commercial real-time operating system and development environment, modeling and performance engineering of these systems, and their interactions with physical systems.*

1. Introduction

Embedded computers are now ubiquitous, often in common products where they are invisible to the user. These embedded processors provide special purpose functionality not found in general-purpose applications familiar to desktop computer users. The standard computing curricula concentrate primarily on general-purpose desktop applications and do not provide students with the opportunity to gain the necessary skills for engineering software in real-time and embedded systems.

2. Real-time and embedded systems at RIT

In Rochester Institute of Technology's computer engineering program, senior projects often focus on real-time and embedded systems, but there was no formal instruction in the engineering of these systems. The software engineering program had an embedded systems application domain comprising three courses: two standard operating systems courses offered by computer science and a concurrent programming course from computer engineering. None of these courses directly addresses issues in developing real-time or embedded software; they were chosen because they were the closest courses relevant to the domain. We decided that the best way to address these shortcomings in the real-time and embedded domain in both the computer engineering and software engineering curricula was to adopt a cross-disciplinary approach. The presence of students from both programs created a unique opportunity for synergy at RIT. The computer engineering students possess knowledge of electronics and control systems along with software development skills at the lower-levels. The software engineering students possess significant knowledge of how to engineer complex software systems including the design and modeling of those systems. Developing software for real-time and embedded systems is where the skills of these two groups intersect.

In July, 2003, we started work on the laboratory and the development of a three-course sequence. Each of these upper-division courses is four academic quarter credit hours and meets for ten weeks of classes having a pair of two-hour studio sessions per week. In the studio-lab environment each class session mixes lecture material with hands-on exercises and projects in a flexible format. These courses are cross-listed in the software engineering and computer engineering programs. Registration is initially controlled with the goal of having an even mix between students from the two programs. To the extent possible we ensure that all project teams have a member from both computer engineering and software engineering. The students will bring together expertise from two domains and apply a common engineering approach for solving real-time and embedded system development problems. To this point, we have offered the first two courses in the sequence several times. The third course is currently being offered for the first time in the Spring 2005 academic quarter. The remainder of this paper describes our laboratory facilities, the syllabus for the three courses we developed and some initial results of the internal and external evaluation of the program.

[•] Sections of this paper will also be presented at the Frontiers in Education 2005 Conference in October 2005.

Our funding came from the award of a National Science Foundation Course, Curriculum and Laboratory Improvement Adaptation and Implementation grant. We identified the School of Computing and Software Engineering at Southern Polytechnic State University and the Department of Computer Science and Engineering at Arizona State University as the collaborating institutions that would provide course materials for adaptation into the courses we developed.

3. Laboratory hardware facilities

The studio lab developed for these courses consists of twelve student stations and an instructor's station. The instructor's station is configured with classroom control software that enables the capture, control and display of any of the student stations on the classroom video projector. Each student station is positioned to allow a pair of students to work together. Each station has a modern personal computer for software development and a 486-based single board computer as a target system. We are using a Diamond Systems [1] pc-104 board with timers, A/D converters, D/A converters, and digital I/O capability for the target systems. See Figure 1.

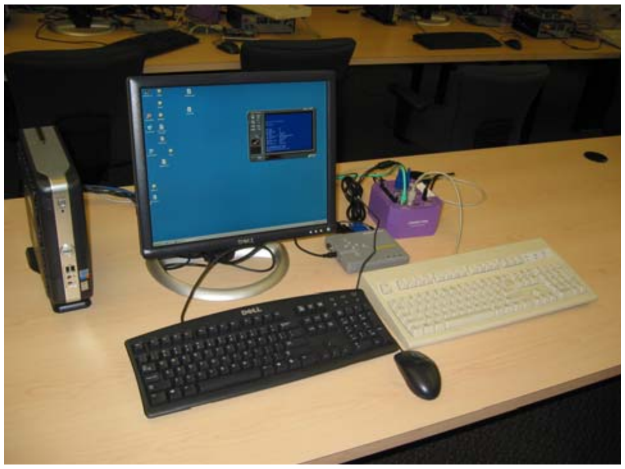
Figure 1 – PC Development environment and Diamond Systems pc-104 board target system showing picture-in-picture target system console.

To reduce the clutter in the student's work area we eliminated the second monitor often attached to the target system. Students can view the output from the target system in a number of ways. For text-based standard output, the target system development software provides a redirected console on the development system. We also have the VGA output converted to S-video and then fed into a USB S-video digitizer. The digitizer's software provides a picture-in-picture display shown in Figure 1. Finally, for projects that are generating VGA graphics output the student can view the full resolution video through the second input channel on the development station's dual-input monitor.

For the experiments involving programming a microcontroller, each station is also provided with a Motorola 68HC12 board, a custom designed interface board on which is mounted the microcontroller board, a custom binary LED-switch board for elementary binary input and output, a signal generator and a power supply.

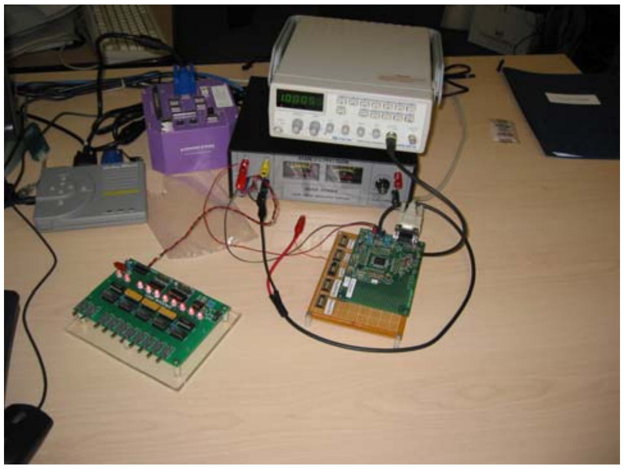
Figure 2 – M68HC12 Microcontroller, interface board, LED-Switch Board, Signal Generator and Power Supply.

The last pieces of hardware to mention are primarily used in the third course in the sequence. This course covers performance engineering of real-time and embedded systems. To motivate the need for system tuning of real-time systems we use the control of physical systems. The two systems we choose for the laboratory are from Quanser Systems [8]. We selected their inverted pendulum and ball and balance beam systems shown in Figures 3 and 4 respectively. In the third course the students also experiment with hardware/software co-design on a Digilent Spartan 3 FPGA board [2] shown in Figure 5. There is one FPGA system at each student station.

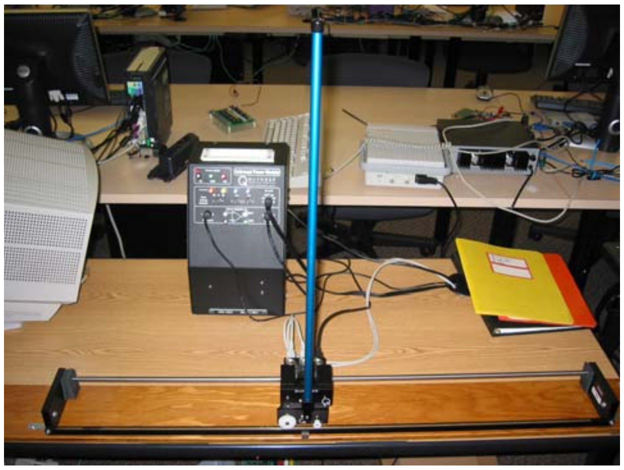
Figure 3 – Quanser System Inverted Pendulum

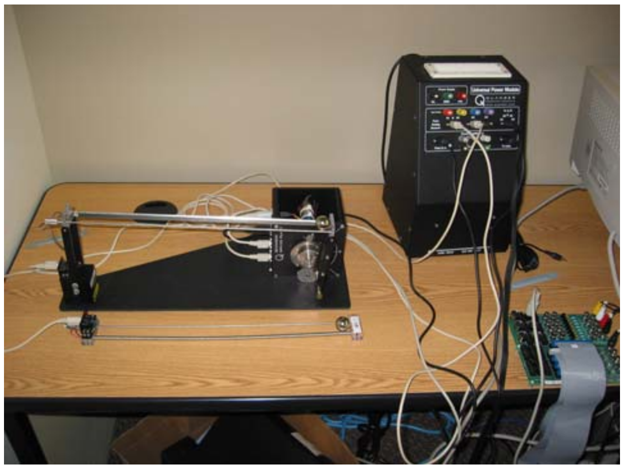

Figure 4 – Quanser System Ball and Balance Beam.

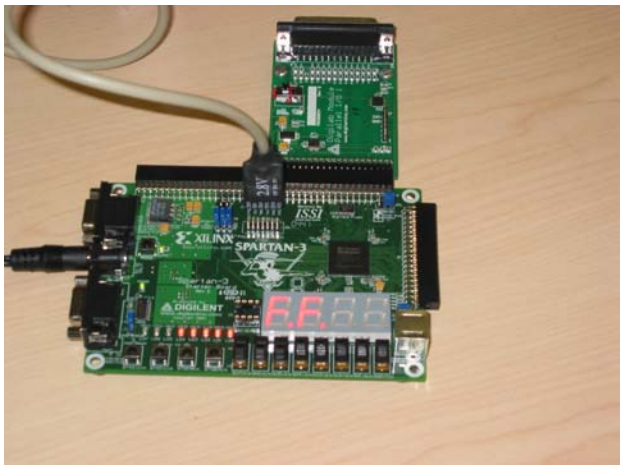

Figure 5 – Digilent Spartan 3 FPGA Board

4. Laboratory software facilities

There is a set of software tools to complement the hardware in the laboratory. The development stations are running the Windows XP Professional operating system. The MGTEK MiniIDE [7] supports assembly language programming on the 68HC12 microcontroller. We received a software grant from Wind River Systems [11] allowing the use of VxWorks and the Tornado integrated development environment. This is the commercial real-time operating system that the students utilize in the laboratory. Matlab and Simulink from The MathWorks [6] are used for simulating and controlling the Quanser experiments. We received software grants from IBM [4] for the Rational Rose development suite and Rational Rose Real-Time as UML modeling tools. Finally, the students work with Rhapsody from I-Logix [5] as a UML modeling tool. Rhapsody's statechart modeling and code generation features are used heavily in the second course in the sequence.

5. Course concepts

We designed a sequence of three courses that provides the student with broad exposure to the real-time and embedded systems domain. The first course, Real-Time and Embedded Systems, provides a general introduction to the area. We expect that this course will have the largest appeal across both disciplines with some aspects particularly attractive to both the computer engineering and software engineering students. The second course, Modeling of Real-Time Systems, has a stronger software engineering flavor. It covers UML modeling of real-time and embedded systems. The third course, titled Performance Engineering of Real-Time and Embedded Systems, deals with measurement of system performance, implementation of time-critical software and the fluid hardware/software boundary. The next sections describe these three courses in detail.

6. Real-time and Embedded Systems course

The first course in this elective sequence is titled Real-Time and Embedded Systems. It presents a general road map of real-time and embedded systems. It introduces a representative family of microcontrollers that exemplify unique positive features as well as limitations of microcontrollers in embedded and real-time systems. These microcontrollers are used as external, independent performance monitors of more complex real-time systems targeted on more robust platforms. The majority of this course presents material on a commercial real-time operating system and using it for programming projects on development systems and embedded target systems. Some fundamental material on real-time operating systems is also presented. This course was first offered at RIT in the spring of 2003. It has since been offered three more times. The textbook for the course is *Real-Time Systems and Software* by Shaw[9].

The topics covered by the class provide an introduction to the area. Class discussion focuses primarily on the fundamentals of real-time systems. The project work spans the range from microcontroller assembly programming through to application development under a commercial real-time operating system.

The topics covered by the Embedded and Real-Time Systems course include:

- Introduction to Real-Time and Embedded Systems
- Microcontrollers
- Software Architectures for Real-Time Operating Systems
- Requirements and Design Specifications
- Decision Tables and Finite State Machines
- Scheduling in Real-Time Systems
- Programming for a commercial real-time operating system

- Development for Embedded Target Systems
- Design Patterns for Real-Time Systems
- Language Support for Real-Time
- Real-Time and Embedded Systems Taxonomy
- Safety Critical Systems

There are several programming project assignments given to the students. A pair of students works on each assignment. As was mentioned previously, to the extent that the registration numbers permit a software engineering and computer engineering student are paired together. This course has a mix of projects that allows the computer engineering student to provide the lead on some and the software engineering student to lead the others. The project assignments for this course are:

Microcontroller programming: students program the 68HC12 microcontroller to act as an interval timer and as an independent system performance measurement device. The microcontrollers used assembly language programs to measure and tabulate the inter-arrival times, the "jitter", of a series of 1000 pulses for several experiments described later. The microcontroller's timers have no difficulty measuring the arrival times or interarrival times of the pulses to 1.0 microsecond resolution.

Real-Time Operating System multi-tasking primitives: the main goal for this project is to have the students become familiar with programming under a commercial real-time operating system. Using VxWorks as an example of a commercial real-time operating system, students learn how to program using its concurrency and synchronization primitives. The team must implement a concurrent system such as a transit simulation or an automated factory. The programming is done within a simulated target system running on the development station.

Real-Time Operating System performance measurements: there are two smaller projects that fall into this category. These programs run in the target systems. Both projects make use of the microcontroller project as a timing device. In the first project the students learn how to schedule a periodic task under VxWorks. This task is toggling a bit on the printer port. The microcontroller timer measures the inter-arrival time and jitter of these software-generated periodic pulses. The second project measures the interrupt response time of the target system by having the microcontroller measure the time between generating an interrupt signal to the target and receiving its response. These two projects are run on the target systems and the microcontroller collects 1000 samples with 1.0 microsecond resolution and displays the results.

Final project: there is a final programming project. This project is usually of student motivated with each team thinking of a project. We have seen implementations of user-level drivers for the devices on the target system, an ultrasound distance measurement, simple video games, and a digital oscilloscope.

Students are presented with two different embedded processors and development environments and are confronted with the strengths and weaknesses of each platform/architecture and environment.

Using Bloom's Taxonomy as a guide, the learning outcomes for this course are given in Table 1.

Table 1
Learning Outcomes for Real-Time and Embedded Systems Course

Knowledge	
	• List the scheduling algorithms commonly used in real-time systems. • Describe the steps required to build, install and run a software system on an embedded processor.
Comprehension	
	• Discuss the event sequence for responding to an interrupt.
Application	
	• Apply software engineering practices to the development of several small real-time systems. • Demonstrate the use of a micro-controller as an event timer. • Design and implement measurement tools to collect system performance data. • Design and implement a concurrent system on a real-time operating system.
Analysis	
	• Measure the performance of a real-time operating system.
Synthesis	
	• Design and implement a small-scale real-time application on a real-time operating system.

7. Modeling of Real-Time Systems course

The second course is titled Modeling of Real-Time Systems. The course takes an engineering approach to the design of these systems by analyzing a model of the system before beginning implementation. The course discusses primarily UML based methodologies. Implementation of real-time systems will be developed manually from the models and using automated tools to generate the code. At this point, this course has run twice. *Doing Hard Time* by Douglass [3] is the textbook for the course.

Topics covered by the Modeling of Real-Time Systems course include:

- Introduction to Modeling of Real-Time Systems
- Basic Concepts of Real-Time Systems

- Basic Concepts of Safety-Critical Systems
- Use case analysis for real-time systems
- Structural object analysis for real-time systems
- Behavioral Analysis using statecharts
- Design patterns for real-time and safety-critical systems
- Threading and Schedulability
- Real-Time Frameworks

This course has the strongest software engineering emphasis. The projects progress through phases in the standard waterfall process model with emphasis on analysis and design of the software system. For the software engineering students this is continued practice in the UML modeling that they do in all the courses in their program. The application areas chosen for the projects, i.e. embedded systems, are significantly different from the typical desktop and GUI-over-database projects that they see in their other courses. In this course the software engineering students take the lead on most projects. Many computer engineering students have not done any UML modeling since their second-year software engineering course. The project assignments for this course are:

Requirements and Architectural Design: this assignment starts with the user manual for a consumer electronic device. It requires the students to identify the actors in the system and do a use case analysis. This is then followed by an architectural design and high-level class structural design. A home blood pressure monitor and a digital video recorder are two devices that students have modeled for this project.

Design and Implementation: this assignment starts with a clear statement of requirements and requires the team to do a class-level design and implementation. We have used both end-user applications, four-function calculator, and a simulation, controller for a chilled water air conditioning system. The implementation language is Java with the team implementing a graphical user interface to control the program.

Code Generation: through this course we place an emphasis on statecharts as a mechanism for behavior modeling of real-time and embedded systems. In this project the students explore the code generation features of the modeling tool they use. The teams create a statechart-based definition of the behavior and automatically generate C++ code for the application. Typically, the team will be able to create a fully-functioning application entirely from within the statechart model. This is not to say that the team writes no C++ code. Some adornments to states are code snippets that get built into the code that the tool auto-generates. For this project we have used a four-function calculator and garage door opener controller.

Final Project: this project is a modeling exercise done as a take-home final exam. Each student does a thorough identification of actors, a use case analysis, class structural design and system dynamic modeling using sequence diagrams and statecharts. There is no implementation of the systems which to date have been a power window controller for a car and a reverse vending machine that accepts containers for recycling at a local supermarket.

Using Bloom's Taxonomy the learning outcomes for this course are given in Table 2.

Table 2
Learning Outcomes for Modeling Real-Time Systems Course

Knowledge	
	• Specify the characteristics of real-time and safety critical systems.
Comprehension	
	• Discuss the software process for the development of real-time systems and contrast it with development for a standard application. • Identify architectural and design patterns for real-time and safety critical systems.
Application	
	• Apply architectural and design patterns in the analysis and design of real-time systems.
Analysis	
	• Model the dynamic behavior of a real-time system using statecharts. • Describe the requirements for simple real-time systems using use cases. • Model the structure of a real-time system using UML class diagrams.
Synthesis	
	• Implement a simple system on a real-time operating system.

8. Performance Engineering of Real-Time and Embedded Systems course

The third course is Performance Engineering of Real-Time and Embedded Systems. This course is first being offered during the spring quarter of 2005. As of this writing, aspects of the course are still under development. The course is roughly divided in half with the first and second parts emphasizing performance of real-time systems and embedded systems, respectively. This course has an unusual combination of topics and we have not identified a single textbook that is suitable. We are covering the course topics with handouts and other on-line resources for the students.

Topics covered by the Performance Engineering of Real-Time and Embedded Systems course include:

- Performance measurements for real-time and embedded systems
- Profiling of program execution in embedded systems
- Exploration of linear control systems

- Interpretation of linear control parameters
- Hardware system description languages
- Hardware/software co-design

The real-time part of the course presents the control of physical systems on an intuitive level. The intent is to give exposure to control system structure and performance rather than have student design control systems. The software engineers have no background in controls. The computer engineering students are able to contribute to the analytical and control algorithms from their required control systems courses and will take the lead on these projects. Students perform experiments with the inverted pendulum system and a ball and balance beam. These experiments highlight the effect of parameter tuning and system load on control of the physical apparatus. In future offerings, this set of experiments will culminate with student implementations of software controllers.

The embedded systems part of the course uses our target system as the computing element running the VxWorks commercial real-time operating system. We deliberately chose a rather slow (100MHz clock) 486 processor for our target systems so that we could more easily monitor loading effects. This is close to power management policies in low-power embedded devices that prolong battery life by slowing the clock speed. In subsequent course offerings, input and output devices will be connected through an FPGA I/O controller. Students will measure initial system performance when the I/O controller is a pass-through interface between the processor and the devices. The current offering has the students performing a set of JPEG image compressions, first using an all-software approach on the target system, and then off-loading some of the computations to an attached FPGA board. The students will then be able to make a hardware-software co-design tradeoff by placing more device control functionality in the FPGA. At each step the students will measure the change in system performance as the boundary between hardware and software is moved.

Using Bloom's Taxonomy the learning outcomes for this course are given in Table 3.

Table 3
Learning Outcomes for Performance Engineering of Real-Time and Embedded Systems Course

Knowledge	
	• Identify PID control modes • Identify the major characteristics of a Field-Programmable Gate Array (FPGA)
Comprehension	
	• Distinguish differences between PID control modes • Contrast effects of system parameters on control of a physical system.
Application	
	• Profile the execution of an embedded system • Be able to program an FPGA doing minor revisions to VHDL code
Analysis	
	• Describe hardware/software tradeoffs in the design of an embedded system. • Analyze the profiling data to determine which areas of the program would benefit most from performance tuning. • Compare performance of systems based on performance data.
Synthesis	
	• Design a test and measurement plan to collect system performance data. • Demonstrate the effects of moving the hardware/software boundary in a design

9. Evaluation plan

This project has two components in its evaluation plan.

External evaluation: a faculty member from one of our collaborating institutions evaluated our work at the end of the first year in May 2004. At this same time we had an external review by someone working in local industry developing real-time and embedded systems. Near the end of the NSF funding period in June 2005 we will again arrange a review by faculty from our collaborating institutions and local industrial representatives.

Course evaluations and surveys: students enrolled in the courses are given concept surveys at the beginning and end of each course to assess their domain learning through each course. Course evaluations will ask students to assess the course materials, the laboratory environment, the teaching effectiveness and whether the course has increased their interest in real-time and embedded systems or helped them get a co-op or full-time position.

10. Future work

This section describes some areas for improvement that have been identified and other activities for the future.

- One challenge has been to develop courses interesting to the software engineers and computer engineers. The Modeling course is very well liked by the software engineering students but is not as attractive to the computer engineers. We need to balance that course more. Even the SE students suggest that we select projects with more explicit time-dependent requirements. We will also consider designing a project that requires implementation on the Java Micro Edition platform.
- The main exposure to VxWorks is in our first course. We do not have a strict prerequisite structure within these three courses thus we are hesitant to put projects requiring implementation on VxWorks in

the other two courses. We need to create a very succinct tutorial on writing applications for VxWorks that we can use in the two courses that currently do not cover the RTOS in detail. It took us quite a while to settle on a configuration for VxWorks in the lab that could easily support 13 simultaneous target systems and give easy distribution of new VxWorks images. We next need to work on giving students the necessary control to create their own images when their project is developing a kernel-level driver. We will also investigate the use of a real-time variant of Linux in these courses.

- The lack of a suitable textbook for the performance engineering course is an issue for that course. We will assess the best approach to follow after the course has run for its first time in our spring 2005 term.
- There are other devices that we would like to have students use with their project work. At the top of the list would be interfacing to cheap USB webcams. Unfortunately, we have not yet identified any cameras that publish their USB interface.
- A last element of dissemination of our work, which will take place at the end of the project, is to collect all of our course materials, projects, exams, etc. onto a password protected website and publicize its availability to the engineering education community.
- The facilities are mostly in place now and this has attracted the attention of other faculty members. We already have one faculty member scheduled to develop a fourth course to be taught in the lab next year.

11. Acknowledgements

This project is being conducted under the sponsorship of a National Science Foundation grant under the Course, Curriculum and Laboratory Improvement Program (NSF DUE-0311269) and in collaboration with Professor Yann-Hang Lee of Arizona State University and Professor Ronald Schroeder of Southern Polytechnic State University. We would also like to thank Mr. Todd Mosher of Alstom Transport Systems for his review of our project’s first year.

12. References

[1] Diamond Systems, http://www.diamondsystems.com.

[2] Diligent, http://www.digilentinc.com.

[3] Douglass, B. P., *Doing Hard Time – Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns,* Addison Wesley, Reading, 1999.

[4] IBM Rational Software, http://www.rational.com.

[5] I-Logix, http://www.ilogix.com.

[6] The MathWorks, http://www.mathworks.com.

[7] MGTEK, http://www.mgtwk.com/miniide.

[8] Quanser Systems, http://www.quanser.com.

[9] Shaw, A. C., *Real-Time Systems and Software*, John Wiley & Sons, Inc., New York, 2001.

[10] Starnes, T, "Microcomputers Infest the Home", Gartner Research, Inc. 2002.

[11] Wind River Systems, http://www.windriver.com