A Virtual Embedded Systems Testbed for Instruction and Design

Gerald Baumgartner
Dept. of Computer and Information Science
The Ohio State University
395 Dreese Labs., 2015 Neil Ave.
Columbus, OH 43210-1277
gb@cis.ohio-state.edu, (614) 292-5841

Ali Keyhani
Dept. of Electrical and Computer Engineering
The Ohio State University
205 Dreese Labs., 2015 Neil Ave.
Columbus, OH 43210-1277
keyhani@ee.eng.ohio-state.edu, (614) 292-4430

Abstract

Today virtually all engineering disciplines, such as transportation industry, manufacturing, consumer electronics, and energy systems, to name a few, are using embedded systems. From automobiles, elevators, and dishwashers to power converters, CD-ROM drives, and VCRs to pace makers, wrist watches, and even light switches, a wide variety of industrial and consumer products are being equipped with embedded processors that interact with the electrical and mechanical parts of the product. Development of such embedded systems is a complex process. It involves many aspects of electrical engineering and computer science technologies. With the use of embedded systems increasing that rapidly, so is the need for programmers trained in the skills required by embedded systems programming: basic knowledge of the electrical engineering and computer science aspects with practical hands-on experience on an actual embedded system and with knowledge of the particular application domain.

The conventional way of teaching embedded systems programming is either through the use of an extensive laboratory setup for small numbers of students or using simulation or pure lecturing for larger numbers of students. We will develop a virtual testbed that allows teaching hands-on experience to large numbers of students.

The hardware for the virtual testbed will initially consist of a PC connected to a high-end TI floating-point DSP board. Instead of requiring an extensive laboratory setup of electromechanical devices connected to the DSP, we will simulate the external devices on the PC. User programs on the DSP will access the external devices through device drivers, which will transparently forward the requests through the parallel port to the simulators running on the PC. Using software instruments such as voltmeters and scopes and a Web-based graphical user interface, students can get realistic hands-on experience in programming an embedded processor without the need for external electromechanical hardware.

For the communication between the DSP and the simulators, we will use an XML-based protocol. Any data written to devices or read from devices as well as interrupts and clock synchronization requests will be transmitted as short XML documents. From the simulators' point of view, this will make the communication with the DSP look like the interaction with a web server. The advantage of this architecture is that the virtual testbed can be retargeted to different embedded processors (fixed-point or floating-point DSPs, or even microcontrollers) with minimal effort. Also, this architecture will allow running the simulators on a remote machine or connecting multiple DSPs to one server PC. This will give maximum flexibility for schools to set up labs, ranging from a server with a hand-full of DSPs shared by a large number of students to one PC and one DSP board per seat.

We will develop the instructional material for three project-based courses: an introductory computer science course on Microcomputer Systems, an electrical engineering course on DSP Control of Electromechanical Systems, and a Capstone Design of Embedded Systems course. The course material will be structured as a sequence of modules, each of which centers around a small programming project. The course material, like the interaction with the virtual testbed, will be Web-based to allow wider dissemination and possible future distance learning.

1. Problem Statement

Embedded systems are rapidly becoming ubiquitous. From automobiles, elevators, and dishwashers to power converters, CD-ROM drives, and VCRs to pace makers, wrist watches, and even light switches, a wide variety of industrial and consumer products are being equipped with embedded processors that interact with the electrical and mechanical parts of the product.

Development of such embedded systems is a complex process. It involves many aspects of electrical engineering and computer science technologies. The electrical engineering aspects include circuit theory, digital circuits, control theory, and power electronics system technology. The computer science aspects include systems programming, operating systems technology, programming language and compiler support, and software engineering techniques. Nowadays, development of these products require even broader knowledge than it did in the past. With the advancement of digital signal processing technology, the use of digital signal processors (DSPs) is not uncommon these days. DSP based digital control schemes are replacing the functions that used to be implemented in discrete analog and digital circuitry. This in turn adds signal-processing technology to the list of knowledge required by a product developer.

With the use of embedded systems increasing that rapidly, so is the need for programmers trained in the skills required by embedded systems programming. A trained embedded systems developer must combine basic knowledge of the electrical engineering and computer science aspects with practical hands-on experience on an actual embedded system and with knowledge of the particular application domain.

The conventional way of providing practical experience to electrical engineering students is through the use of extensive laboratory based learning systems. Such systems require an actual hardware setup and a set of laboratory measurement systems that sometimes are costly to build and difficult to maintain. For safety and security reasons, access to the laboratory-based system is usually limited to a certain time and can only be conducted with the presence of a local facilitator. Because of the elaborate setup required, only small numbers of students can be educated in such a laboratory environment. For lack of the required electrical and mechanical hardware, computer science students often get little if any hands-on experience on embedded systems but, instead, would be taught on general-purpose hardware or simulators.

Objectives

The goal of our curriculum development is to build an environment and associated course material for teaching hands-on embedded systems programming to large numbers of students. We will build a virtual embedded systems testbed that provides a realistic embedded systems programming environment by employing an actual embedded processor or DSP board while simulating the external electrical and mechanical devices that are controlled by the processor. With the hardware requirements reduced to one PC and an embedded processor board per work place, this environment will allow teaching large numbers of students in a classroom setting. It will also be useful as an economical development environment for many embedded applications.

Technological Component

The virtual testbed [Figure 1] will consist of a PC, an off-the-shelve embedded processor board, and the following software components:

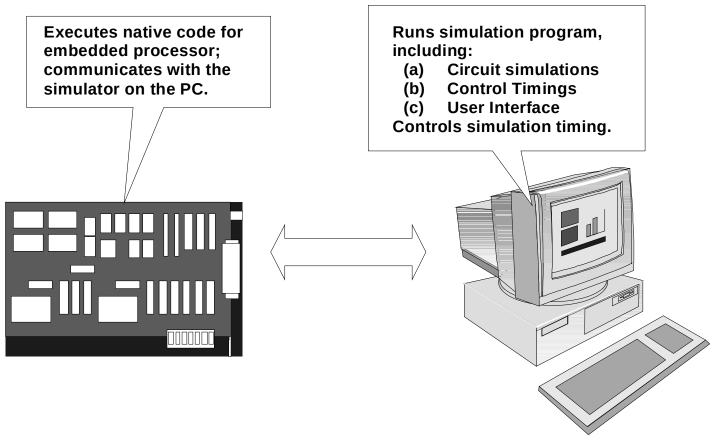

Executes native code for embedded processor; communicates with the simulator on the PC.
Runs simulation program, including:
(a) Circuit simulations
(b) Control Timings
(c) User Interface
Controls simulation timing.

Figure 1: The virtual embedded systems testbed.

1. Off-the-shelve programming tools for the embedded processor, including a compiler, a linker/loader, and a debugger.
2. A custom device driver library that forwards access to external devices through the parallel port to the simulator on the PC.
3. Simulators for a variety of electrical and mechanical devices.
4. Communications software employing an XML-based protocol for connecting the embedded processor with the simulators.
5. Software instruments, such as a voltmeter, for measuring electrical or mechanical quantities of the simulated devices.
6. Software to facilitate connecting the simulators for individual devices and instruments into a larger simulation environment.
7. A web-based graphical user interface for interacting with the virtual testbed.

Since only the device drivers and the communications software depend on the actual embedded processor, we are planning to develop this testbed so it can be easily ported to different embedded processor boards.

Educational Component

In addition to the virtual testbed, we will develop the course material for three courses for teaching embedded systems programming to electrical engineering and computer science students. The course material will be a combination of classroom lecturing material and a web-based learning tool that allows students to interact with the virtual testbed and to study (parts of) the material on their on. It will be designed based on up-to-date instructional technology, see, e.g., [1-12]. The learning system will be developed based on a modular instructional technology curriculum. The material will be structured in a sequence of modules that, successively, introduce the student to more and more advanced knowledge. The modules will have a similar structure, consisting of:

1. An explanation layer that introduces the student to the basic fundamental notions and concepts.
2. An exploration layer that guides the student through the solution process for an example problem.
3. A diagnostic layer that guides the student through the material and tests the student’s knowledge.

This learning system will be useful for administering programming assignments for traditional courses at a university as well as for self-study when combined with a video of the class lecture.

2. Technology: A Virtual Testbed for Hardware-in-the-loop Simulation

The traditional environment for teaching software development for an embedded processor uses either an elaborate laboratory setup, in which the embedded processor controls actual electrical or mechanical hardware devices, or it uses a simulator for the embedded processor as well as for the external devices. The advantage of a laboratory environment is that it is realistic; the disadvantage is that it is expensive, that incorrect interaction with the external devices can damage them, and that it allows teaching only small numbers of students due to the requirements for an appropriate lab. The disadvantage of an environment in which the processor is simulated is that it is slow and not realistic enough.

The goal for the virtual testbed is to provide as realistic a programming environment as possible with a minimal investment in hardware. We will achieve this by using an actual embedded processor while simulating the external electromechanical devices. Special purpose virtual testbeds have been built for a variety of devices [13-16], however programmers had to be aware of whether they were interacting with the real hardware or a simulator. We will develop a virtual testbed that simulates electromechanical devices in a transparent manner using a device driver library that provides the same programming interface to the device simulators as to real devices.

Structure of the Virtual Testbed

In order to make programming the embedded processor as realistic as possible, we will design the virtual testbed under the following assumptions:

1. User code is run on an actual embedded processor or DSP. Typically, an embedded processor board would be connected to a PC through the parallel port or plugged into an expansion slot.
2. Access to devices connected to the embedded processor is intercepted and directed to simulators for these devices. These device simulators would typically run on the host computer.
3. It should be possible to switch any device from simulation to accessing actual hardware without modifying the control program. It should at most require recompiling the program.
4. There should be as few restrictions as possible on what code can be run in the virtual testbed.
5. It should be possible to port the virtual testbed to a different embedded processor with minimal effort.

Initially, we will develop the virtual testbed with a Texas Instruments TMS320C67x floating-point digital signal processor (DSP) as the embedded processor. Using such a high-powered DSP does not allow teaching programming under the constraints of small memory and a primitive processor. However, the most important aspect of embedded systems programming – interacting with external devices – can be taught with a DSP as well as with other embedded processors. In addition, a DSP allows teaching signal processing, and the higher processor speed will aid in developing the virtual testbed. The virtual testbed could then later be ported to an 8-bit processor if teaching programming under severe processor and memory constraints is desired.

The software architecture [17] on the PC side of the virtual testbed, as shown in Figure 2, consists of four main parts: the DSP control engine, the device simulation engine, the instructional modules, and a visualization unit. The DSP control engine provides the interface for the embedded processor board. It consists of off-the-shelve programming tools, such as a compiler, a linker, a loader, a debugger, and tools for controlling the execution of the code on the DSP, relaying signals between the DSP and the device simulators, and retrieving control code variables on the DSP for display.

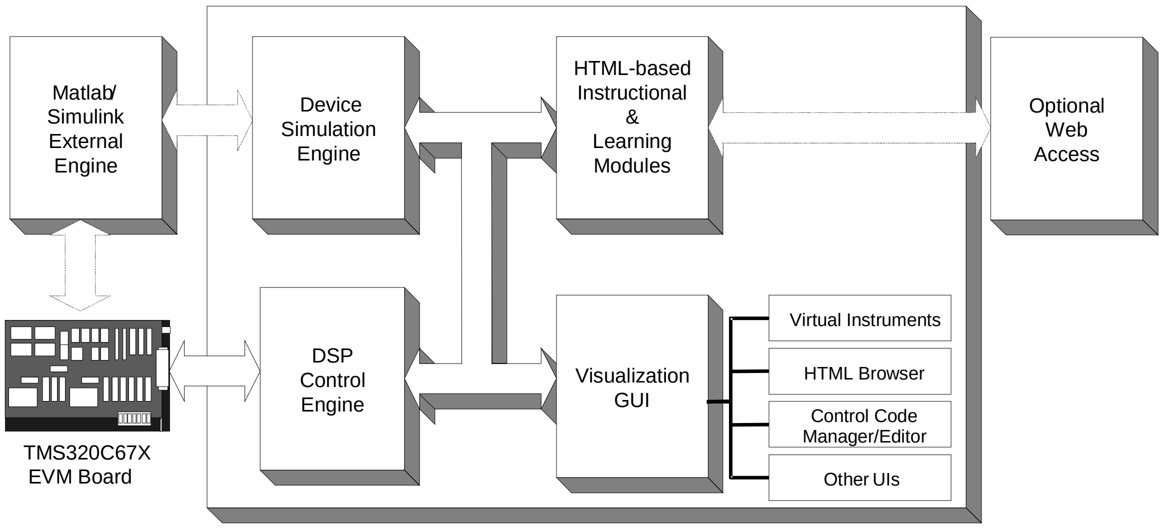


Figure 2: The software architecture of the virtual testbed.

The instructional and leaning modules will be represented in HTML (Hyper-Text Markup Language) format allowing them to be accessed from the Web if the need arises. The visualization unit consists of various visual graphical user interfaces to support the virtual learning environment including an HTML browser and various virtual instruments, such as virtual scopes, virtual meters, or virtual spectrum analyzers, as are commonly found in a laboratory hardware testbed.

The device simulators as well as the display units will be structured as communicating, concurrent object-oriented components. Such a program structure allows a maximum of flexibility in developing individual simulators and in configuring the simulation for a specific application. The simulation engine will be structured in such a way so that the simulation can also be run on an external Matlab/Simulink process. This structure provides greater flexibility and allows the use of various Matlab and Simulink functions in the virtual learning system

Communication between the Embedded Processor and the Host Computer

To make programming the embedded processor as realistic as possible, it is necessary to intercept access to external devices and to forward the requests to the simulators running on the PC. A straightforward solution for intercepting access to the devices is to use a library of device drivers instead of accessing device registers directly. The device drivers can then be configured either to access the actual device or to send a request to the DSP control engine on the host computer.

Instead of using device drivers, it would also be possible to run simple device simulators directly on the embedded processor or to use the debug mode of the DSP for intercepting direct accesses to device registers. However, these solutions are not easily portable to other embedded processors and complicate configuring the simulation for a specific application.

The easiest solution for communicating with the host computer is through the parallel port [Figure 3]. Each access to a device is encapsulated in a packet containing a timestamp, the address of the device register, and the operation to be performed on the device. This packet is sent through the parallel port to the DSP control engine on the host computer, which forwards the request to the appropriate device simulator. Similarly, interrupts generated by any device simulator will cause the DSP control engine to interrupt the DSP. The interrupt handler on the DSP will then determine the source of the interrupt and forward the interrupt to a user-defined interrupt handler.

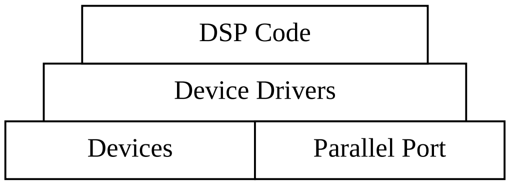


Figure 3: Hardware and software layers on the embedded processor.

In order to make the communication mechanism as general as possible, we will develop an XML-based protocol for transmitting any device requests, device responses, or interrupts between the DSP and the simulators. Any data packet going from the DSP to the simulators or vice versa will be formatted as an XML document. This will make the communication with the DSP look to the simulators like the interaction with a web server. This has the advantage that it abstracts over details of the particular processor and of the underlying connection mechanism between the DSP and the host computer.

This will simplify the process of porting the testbed to different processors, such as fixed-point DSPs or even microcontrollers. When porting the testbed to a new processor, it should only be necessary to write a processor-specific device driver library and to configure the XML parser on the simulator side to handle any processor-specific XML tags. Also, the XML tags need to be associated correctly with the appropriate simulators — the equivalent of connecting external devices to the right pins of the embedded processor.

Since the simulators are only interacting with an XML document, it does not matter where the document comes from. This architecture will, therefore, allow running the simulators on a remote machine or using a server PC as the host for multiple embedded processors. This will give educators a maximum in flexibiltiy for setting up labs. For low-budget environments, a hand-full of DSPs could be connected to one server machine with the simulators running on students' PCs. Students could then reserve a processor for a short time for performing an experiment. A similar setup could be used for distance learning. For a higher-budget environment, an individual PC could be attached to one or more embedded processors.

For many applications, this communication style will be sufficient. However, if simulating a certain device is computationally expensive, the delay in simulation or in the communication could affect the timing of the overall system, and therefore lead to incorrect control of the devices. To prevent this, it is necessary to synchronize the simulation clock with the hardware clock on the DSP by stopping the hardware clock while a computationally expensive simulation is in progress.

For use of the virtual testbed in a laboratory environment, we will also provide a mechanism for controlling the device drivers while the user code is running on the DSP. Using a special interrupt, we will allow switching device drivers between accessing actual hardware devices and forwarding requests to the simulators. This will allow users to develop an application using the simulators and then switching (groups of) devices on the fly from simulation to accessing the real hardware. This mechanism will also aid in debugging the device simulators by running the simulation in parallel with external hardware devices and comparing the results of the simulation with the results obtained from the hardware devices.

Connecting Device Simulators and Software Instruments

The simulation of individual devices ranges from turning an indicator on or off (e.g., for simulating a light switch) to solving systems of equations or differential equations (e.g., for modeling the behavior of electronic circuits). The structure of the device simulating code itself is, therefore, highly irregular and depends on the device being simulated.

However, various intended uses of the teaching system put an emphasis on high flexibility and extensibility of the software. It should be straightforward to add new functionality, such as new device simulators or new software instruments, to the virtual testbed; it should be straightforward to add new teaching modules; and it should be possible to have both individual as well as team projects for the students. It is, therefore, important to develop a common interface to the device simulators as well as to software instruments such as voltmeters and scopes to simplify the task of connecting individual simulators and instruments to each other in different configurations.

Since the task of connecting simulation components is similar to that of connecting graphical user interface (GUI) components for an application, we will use the same technology. I.e., we will write the individual components in an object-oriented language such as C++ [18] or Java [19] using a component architecture such as CORBA, Active-X, or Java Beans and use a GUI builder for visually connecting the components. Implementing the virtual testbed as an object-oriented component library provides optimal flexibility in extending the functionality of the testbed. Using a standard component architecture and a GUI builder simplifies the configuration of the testbed for a specific application. Support for one or more of these component architectures and corresponding GUI builders are available with most commercial compilers. The choice of implementation language and component architecture will depend on the availability and compatibility of a software instrument library, the interface to Simulink, and a GUI builder. Provided all the necessary tools become available, Java would be preferable since it is simpler and easier to use for non-experts.

Remote Access to the Virtual Testbed

We will develop the instructional material as well as the user interface to the virtual testbed in the form of HTML hyper-text documents. The main advantage of this form of presentation is that it allows students to browse the instructional material by following links in the documents instead of reading linearly through a book. With the availability of off-the-shelf web-content authoring tools such as Authorware [20], it is straightforward to produce the instructional material in this form. Given that the instructional material is presented in HTML format, it is a natural choice to provide access to the virtual testbed, e.g., for downloading code to the DSP, for modifying simulation parameters, and for displaying the virtual instruments, through Web pages as well.

Initially, we will develop the virtual testbed to be used on a local machine by a single user. However, given the use of HTML documents for the instructional material and as user interface to the virtual testbed and the use of a component architecture for the implementation of the virtual testbed, it would be natural to allow remote access to the virtual testbed as well [Figure 4].

Once the virtual testbed is working in single-user mode, we will extend it to allow multiple users to access the same virtual testbed, either locally or remotely. This will allow a classroom setting in which one server machine will contain one or more embedded processor boards, which are shared by a larger number of students. The device simulators could then either run on the server or on the students' PCs. The DSP boards would, therefore, be time-shared. Such a structure will also allow several students to work as a team with a single DSP board. With high-speed network access (e.g., with the future Internet 2) and sufficiently fast servers, the virtual testbed could then also be used for long-distance learning.

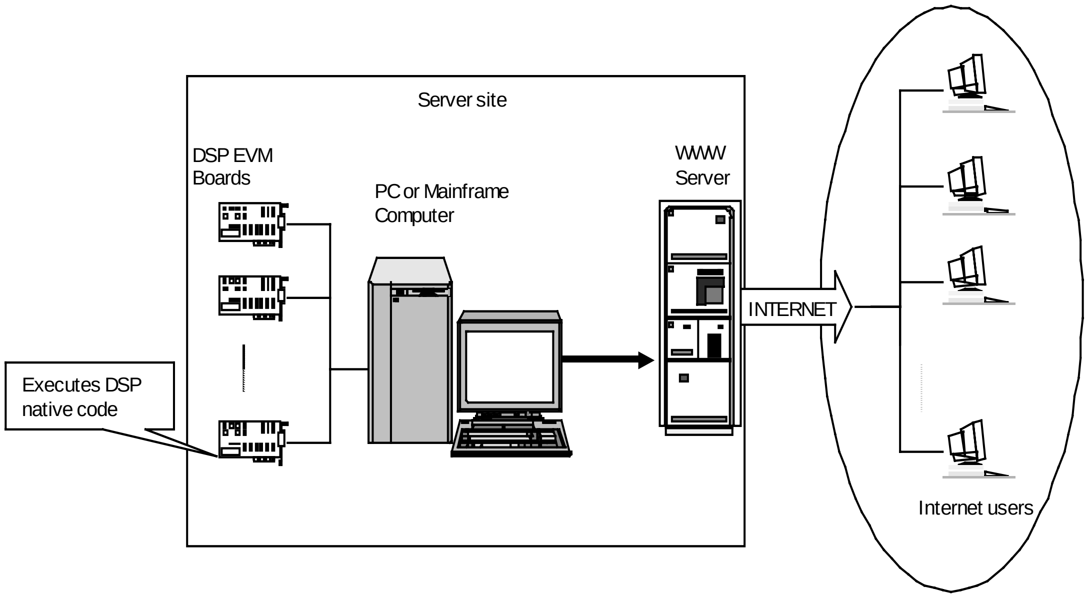


Figure 4: Remote access to the virtual testbed.

3. Curriculum Development

The Existing Curriculum

Currently, the following courses are offered as part of the computer science and electrical engineering undergraduate programs.

1. **Basic Computer Science:** C++ Programming (CIS 321 or CIS 230), Introduction to Computer Systems (CIS 360), Introduction to Computer Architecture (CIS 675).
2. **Basic Electrical Engineering:** Digital Logic Circuits (EE 261), Introduction to Microcomputers (EE 265), Signals and Systems (EE 351 and EE 352).

Traditionally, embedded systems programming is either taught without any practical exercises, with a simulator, or using an elaborate laboratory setup for small numbers of students. E.g., the Computer and Information Sciences Department of The Ohio State University offers a course on Microcomputer Systems (CIS 676), more than half of which is devoted to interrupt structures and interaction with external devices. Depending on the instructor, students either work on a simple simulator or do not get any programming assignments. The Electrical Engineering Department offers a laboratory course Signal Processing course (EE609), for small number of students in the communication area once a year in Spring quarter in which small numbers of students are exposed to Digital Signal Processor (DSP) programming in a laboratory environment with actual hardware devices connected to the DSP. The curriculum at other universities is similar.

These courses will be the prerequisites for two new restructured courses that will be offered under the proposed curriculum development. The first course is Microcomputer Systems (CIS 676) and the second course is DSP Control of Electromechanical System (EE 647). We are planning to develop the course material such that students with background from either the basic computer science or the basic electrical engineering courses could take CIS 676, which would then become a prerequisite for EE 647.

These courses, like most 600-level courses at Ohio State, could also be taken as electives by graduate students.

In the final stage of their curriculum, undergraduate students at Ohio State have to take a Capstone design course, in which they develop a large application from the ground up. Using the virtual testbed, we will also develop such a design course, EE 682, in which students will design DSP based systems. For this design course, students would start the development using the virtual testbed for rapid prototyping of their design projects and, later, they will switch to actual hardware systems of their design projects.

We are planning to develop multimedia course material for teaching embedded systems programming as part of both a computer science and an electrical engineering curriculum that allows teaching large numbers of students while still giving hands-on programming experience in a realistic environment. The course material will be structured as a serious of teaching modules.

Component Modules

Web-based modularization offers numerous advantages: First, modularization is efficient because it reduces a considerable amount of overlapping of course material. This is true even when the courses are not necessarily related. An excellent example of this is the teaching of interrupt handling in a Computer Architecture class, in a hands-on Microcontroller Systems class, and then again in a Capstone design course. Second, modularization gives the instructor the ability to develop a delivery mechanism most suitable for both the project and the students involved. Since the modules are Web-based, the developer has multiple media (e.g., screen text, animation, simulation, audio, and video) at his/her disposal. Finally, modularization helps to tie multiple disciplines together. Oftentimes a student does not need a complete course as a prerequisite to go on to another course. He or she may only need a particular subject from that course. If the course had been modularized then the student could complete the module and take another course that they need. This concept, called *permeability*, enables students to flow through the curriculum with less effort [9]. The aim of such modularization is to permit access to this basic material to students who might not normally select such courses in their curriculum. The size of modules can be kept small because most modules do not contain all of the overlap of material that a normal course would contain. Since students will primarily take these courses in their late junior and senior year as departmental electives, sufficient depth is not sacrificed either.

Along with the advantages, the modular approach is built upon five key premises. The key premises of the approach are as follows:

1. **Modularization of the subject matter** according to embedded system architecture. This approach differs from conventional instruction in that, rather than emphasizing the scientific background, as is already done in conventional courses, the modules will emphasize the description of specific subsystems and components that find application in embedded systems (e.g., timers, interrupts, D/A converters, etc). This approach is relevant in two important ways: first, it satisfies the need for product-oriented continuing education in industry; second, it lends itself to just-in-time learning in a conventional academic environment for students involved in design projects.
2. **Creation of instructional material** for each module based on the following organization: i) explanation layer; ii) exploration layer; iii) (instructional) diagnostic layer. The same layers, when appropriate, can then be organized in recursive fashion, to reach deeper knowledge. The design instruction principles described earlier in this section will be called upon to assist in the instruction. The motivation for organizing the knowledge recursively is provided by the nature of embedded system design. Each design team member will have the need to explore all of the

relevant technologies whether these fall in their primary field of knowledge (electrical engineering or computer science), or in a secondary field. The layering of the modules will allow one to get a basic understanding or an in-depth understanding.

3. **Use of simulations** (including animation and other graphics where appropriate) to assist the student in "visualizing" the concepts. One of the recurring themes of the virtual testbed is the use of simple as well as advanced simulations as a tool for improved design. While analysis can serve a single-domain design task well, the integration of subsystems from different domains requires the use of appropriate simulators to predict the performance of a design. Our approach is based on the extensive use of simulators providing the student with immediate graphical feedback during the learning process. Such simulators will range from very simple ones, used to demonstrate the basic functional operation of a given subsystem, to increasingly complex and realistic ones, up to the detailed device-level design simulators. Commercial simulation tools, such as Simulink™ for general-purpose simulations, will be used where appropriate. A special purpose virtual testbed for embedded systems programming and design will be developed. This is described in greater detail in the previous section.
4. **Integration of the modules into existing courses** (local or distance learning) while preserving the stand-alone nature of each subject. The creation of stand-alone modules serves a dual purpose. One, to reinforce the concepts presented in traditional or future remote video offerings of a course. Two, to permit access to technical information on a just-in-time basis for the benefit of senior design projects, thesis projects, or for industrial projects. The expertise available at UTS will be exploited to obtain assistance in authoring and integrating the modules.
5. **Network implementation** of the above modules to permit local as well as future remote access. Web implementation will be particularly important for disseminating the teaching material, permitting other universities or industry to fully benefit from updates in instructional modules, device libraries, and simulations.

For an example of these concepts, see the description in the next section of the simulation and teaching material for a senior-level design course on power converters.

4. Sample Course Structures

CIS 676: Microcomputer Systems (Computer Systems and Interfaces)

As part of the computer science curriculum, we plan to develop the following modules for introducing students to embedded systems programming. Since these modules are the most basic, particular emphasis will be given on developing the material to be understandable to both computer science and electrical engineering students and to be usable in both a class room setting and for self-study.

1. Introduction to embedded systems, architecture of the DSP, structure of the virtual testbed.
   Project: program a light switch given the code for accessing the devices.
2. Device structure, accessing devices, using counter-controlled loops for timing.
   Project: program a traffic light using counter-controlled timing loops.
3. Accessing the hardware timer, interrupt structure, writing interrupt handlers.
   Project: program a digital watch using a timer and interrupts for button press events.
4. Serial port signals, programming the serial port.
   Project: produce a simple melody by switching the serial port on and off using a timer.
5. Parallel port, DMA controllers and D/A converters.

Project: download a sound file from the host using DMA or the parallel port and play the sound file on the D/A converter.

6. Floating-point arithmetic and fixed-point arithmetic.
   Project: download a sound file and oversample it using both floating-point and fixed-point arithmetic.
7. Simple step motor control, feedback loop, audio CD format.
   Project: play an audio CD by interacting with a (simulated) CD ROM drive controller chip, control the drive speed, oversample, and play the music on the D/A converter.
8. Motor control, advanced feedback loops.
   Group project: program an elevator; take user requests, schedule the elevator stops, measure the weight, and control acceleration and deceleration depending on weight to stop the elevator precisely at each floor.

EE 647: DSP Control of Electromechanical Systems

This course in electrical engineering will build on the material from the previous course.

1. Introduction to laboratory hardware, software, and C programming.
2. Digital interfacing, general purpose digital I/O, relays and switches interfacing.
3. Timer operations, interrupt systems, programming the timer functions: output compare, input capture, and pulse-width modulator, application of timer functions: time delay, waveform generation, frequency measurement.
4. Stepper motor drive and shaft encoder, stepper motor drive circuits, encoders operations and interfacing , incremental encoders, position and speed control of stepper motors.
5. Analog interfacing, signal conditioning circuits, A to D operations and programming.
6. DC motor speed control, base drive and power converter circuits, open loop and closed loop speed control.
7. AC motor drive and control, Volt/Hertz control of induction machines, vector control of induction machines.
8. Group project: DSP design of electric propulsion system.

EE 682: Capstone: Design of Embedded Systems

In prior research, we have developed a special-purpose virtual testbed and some instructional material. Here we present an example of the use of the testbed in designing a power converter. In this virtual testbed, the DSP programmer needed to be aware of whether he or she interacted with actual hardware or with the simulators. Also, the device simulators were written for a special purpose and lacked a general mechanism for configuring the simulation. We present this material here to give a flavor of how we envision the instructional material.

The virtual learning system described here is intended for the design and control of a power converter for UPS applications [Figure 4]. It consists of a three-phase controlled rectifier front end and a three-phase voltage PWM inverter with output filter at the back end supplying a three-phase AC load.

The circuit simulation engine provides the computational engine for the simulation of the circuit responses using the state space based approach with an ideal power switches model. The ideal power switches models are preferred here since they allow the students to build up the understanding of different circuit operational modes and allow for development of state space equations for control purposes. The simulation engine is specifically tailored to and controlled by the instructional/learning module. The instructional/learning module, for example, can ask the simulation engine to simulate an individual stage of the power converter independently.

The simulation engine contains a circuit state space equations generator which can automatically generate state space equations corresponding to each circuit mode of the power converter based on a network list file similar to SPICE. A switching sequence algorithm is then used to select the current state space equation based on the currents and voltages across the switches calculated by the solver and the control signals from the control engine. The state space equations generated by the simulation engine can be optionally transferred to an external Matlab/Simulink process, which can then be used to simulate the power converter circuit.

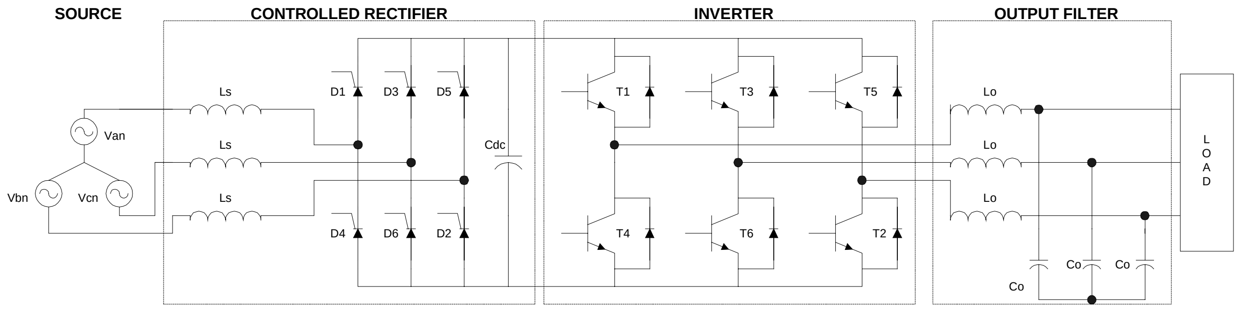


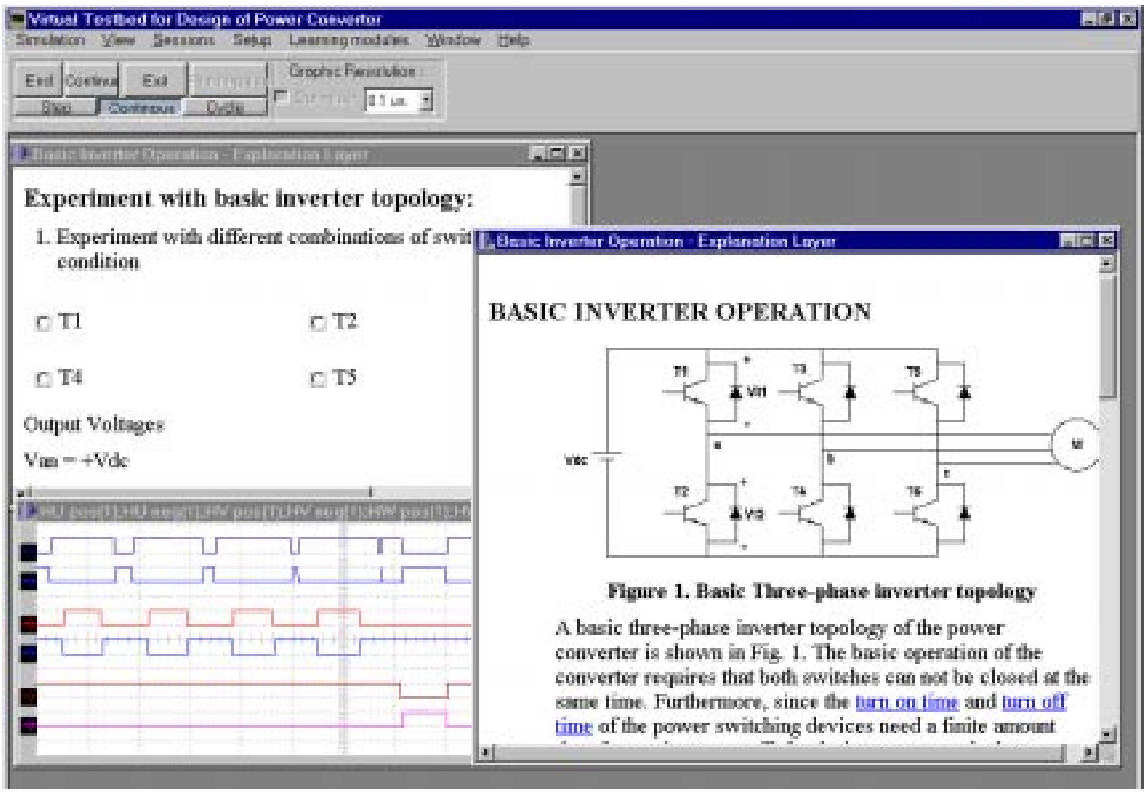
Figure 4: Power converter for UPS applications.

The Instructional and Learning Modules

The instructional and learning modules are designed based on the learning system described earlier. All modules have a similar structure, consisting of an explanation layer, an exploration layer, and an (instructional) diagnostic layer. An example will best illustrate these concepts. It is based on a "power converter" sub-system. The basic operation of the converter does not permit switches on the same leg to be closed at the same time. Furthermore, because of the finite turn-on time and turn-off time of the power switching devices, the designer must provide a delay such that both power switches in one leg of a converter are never closed at the same time. To accomplish this goal, a deadband is introduced in the PWM signals. The following example sub-module is intended to assure that a student has mastered the principles involved in the need for the PWM signals deadband. The example sub-module includes:

1. **Explanation layer:** Basic concept of power devices turn-on and turn-off time; operation of an inverter.
2. **Exploration layer:** Experiment with a converter operation and observe its behavior through simulation and graphics. Figure 5 shows parts of the explanation and exploration being displayed in the application's HTML browsers.
3. **Diagnostic layer:** Pass a diagnostic test to be allowed access to the next (deeper) level. The diagnostic layer is designed to provide a step-by-step understanding of the concept being addressed. For example, PWM signal generation. Every time a wrong answer is encountered, the corresponding explanation layer section is displayed to make sure that the concept is well understood before the next problem is given.

For the power converter in Figure 4, the learning modules cover the materials from basic circuit operations - including circuit modes and switching principles - to the digital control design. The digital control design includes PWM generation strategies, DQ transformations, and basic control loop concepts.


Virtual Testbed for Design of Power Converter
Simulation View Sessions Setup Learning modules Window Help
End Continue Exit
Step Continuous Cycle
Graphic Resolution
0.1 us
Basic Inverter Operation - Explanation Layer
T1 T2 T4 T5
Basic Inverter Operation - Explanation Layer
T1 T3 T5 T2 T4 T6 Vdc M a b c

Experiment with basic inverter topology:

1. Experiment with different combinations of swit
condition

Output Voltages

Van = +Vdc

BASIC INVERTER OPERATION

Figure 1. Basic Three-phase inverter topology

A basic three-phase inverter topology of the power converter is shown in Fig. 1. The basic operation of the converter requires that both switches can not be closed at the same time. Furthermore, since the turn on time and turn off time of the power switching devices need a finite amount

Figure 5: Learning modules displayed in the application’s HTML browsers.

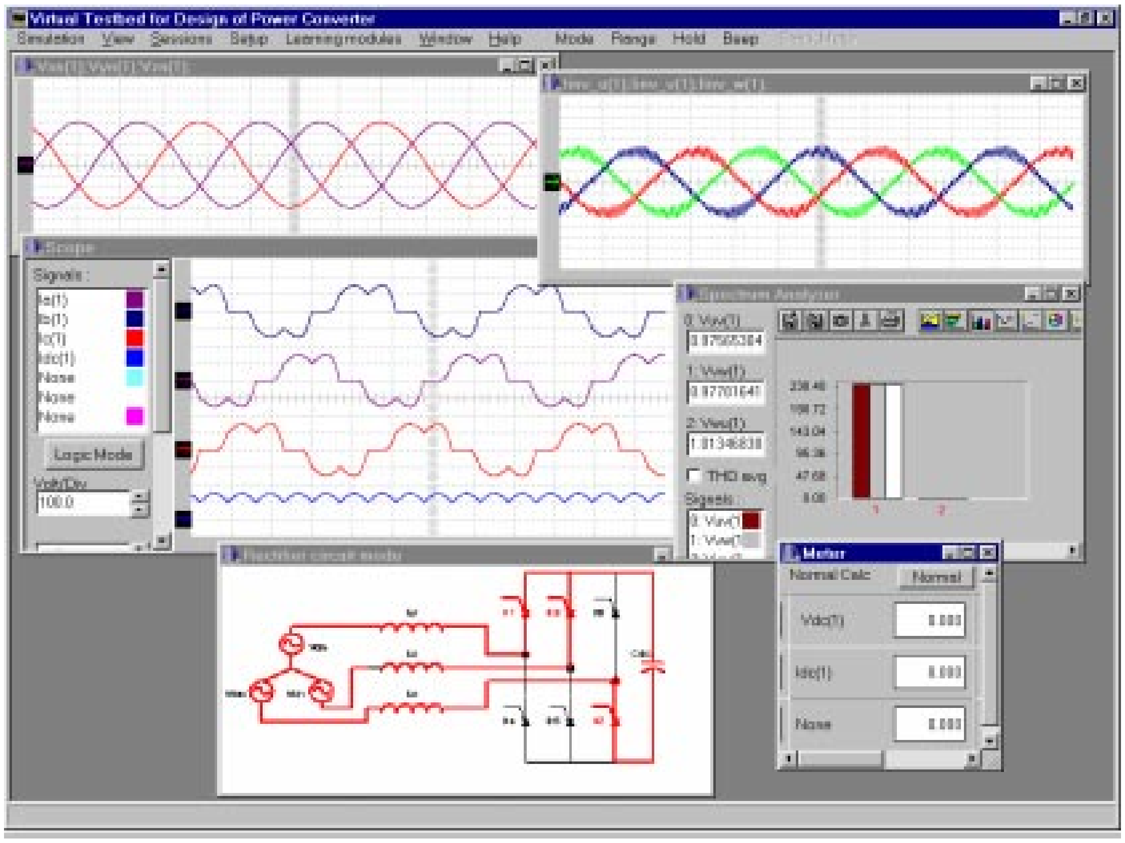


Figure 6: Virtual instruments and circuit mode

All the learning modules will be put in HTML format and can be browsed using a web browser provided by the application environment.

Graphical User Interface

As explained earlier, the integrated virtual learning system provides a realistic environment for students to experiment with the power converter circuit. For this purpose, the virtual learning system is equipped with different virtual instruments commonly found in a laboratory setup. These include virtual scopes, meters, and spectrum analyzers as shown in Figure 6.

In addition to the virtual instruments, the application environment is also capable of displaying the circuit modes of the power converter as shown in this figure for the rectifier circuit.

Other graphical user interfaces included in the system are the control code text editor, DSP code variable trace, and other customization tools.

Bibliography

1. Schmeck, R., Ed., Learning Strategies and Learning Styles, Plenum Press, New York, 1988.
2. R.M. Felder and L. K. Silverman, “Learning and Teaching Styles in Engineering Education”, *Engineering Education*, Vol 78, 1988
3. McKeachie, W.J., Ed., *Learning, Cognition, and College Teaching.* New Directions for Teaching and Learning, No. 2. Jossey-Bass, San Francisco, 1980, p. 32.
4. McCauley, M. H., “Psychological Types of Engineering Students-Implications for Teaching,” *Engineering Education*, Vol. 66, No. 7, 1976.
5. C.C Bonwell and J. A. Eison, “Active Learning: Creating Excitement in the Classroom”, *ASHE-ERIC Higher Education Report No. 1*., George Washington University, 1991.
6. R.M. Felder, "Active, Inductive, Cooperative Learning: An Instructional Model for Chemistry?" *J. Chem. Ed., 73*(9), 832-836 (1996).
7. Conciatore, J., “From flunking to Mastering Calculus,” Black Issues in Higher Education, Feb. 1, 1990
8. R. E. Fullilove and P. U. Triesman “Mathematics Achievement Amoung African American Undergraduates at the Univ. of California at Berkeley” *J. of Negro Education*, Vol. 59, No. 3, 1990.
9. R.M. Felder, G.N. Felder, M. Mauney, C.E. Hamrin, Jr., and E.J. Dietz, "A Longitudinal Study of Engineering Student Performance and Retention. III. Gender Differences in Student Performance and Attitudes," *J. Engr. Education, 84*(2), 151-174 (1995).
10. Greenfield Coalition, “Design and Development Process Handbook for Knowledge Area and Computer Based Instruction”, April 11, 1997.
11. J.M. Keller, “Development and use of the ARCS Model of Motivational Design”, Journal of Instructional Development, Vol. 10 No. 3, 1987
12. J.M. Keller, “A model for motivationally adaptive computer-assisted instruction”, Journal of Research on Computing in Education, Vol. 27, No. 3, 1995
13. A. Keyhani and M. Marwali “A Virtual Testbed for Design and Control of Resonant Converters” IEEE Winter Power Engineering Conference, Panel Session on Use of Information Technology in Education” Tampa, Florida, 1998
14. A. Keyhani and A. Proca “ A Virtual Testbed for Instruction and Design of Permanent Magnet Machines “IEEE Winter Power Engineering Conference, Panel Session on Use of Information Technology in Education” Tampa, Florida, 1998
15. Keyhani, A., Marwali, M.N., Baumgartner, G., “A Virtual Testbed for Instruction Design and Control of Power Converters.” 1998 IEEE Power Engineering Society Summer Meeting, San Diego, CA, July 1998. Submitted to IEEE Transactions on Energy Conversion.
16. Rizzoni, G., Keyhani, A., Washington, G.N., Chandrasekaran, B., Baumgartner, G., “Education in Electronic Systems at the Ohio State University.” 1998 International Mechanical Engineering Congress & Exposition, Anaheim, CA, November 1998.
17. Shaw, M., Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, Englewood Cliffs, NJ, 1996.
18. Ellis, M.A., Stroustrup, B., *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, MA, 1990.
19. Arnold, K., Gosling, J., *The Java Programming Language*, Addison-Wesley, Reading, MA, 1996.
20. Macromedia, Inc., “Authorware 4,” http://www.macromedia.com/software/authorware/.