Nexos: A Next Generation Embedded Systems Laboratory

Dennis Brylow
MSCS Department
Marquette University
1313 W. Wisconsin Ave.,
Milwaukee, WI 53226
brylow@mscs.mu.edu

Bina Ramamurthy
CS&E Department
SUNY at Buffalo
201 Bell Hall,
Buffalo, NY 14260-2000
bina@cse.buffalo.edu

ABSTRACT

The Nexos Project is a joint effort at Marquette University (MU) and University of Buffalo (UB) to build curriculum materials and a supporting experimental laboratory for hands-on projects in embedded systems courses. Our approach focuses on inexpensive, flexible, commodity embedded hardware, (the Linksys WRT54GL wireless router,) freely available development and debugging tools, and a fresh implementation of a classic operating system that is now ideal for embedded system exploration. The prototype laboratory environment is being used in multiple courses at our respective Universities, with excellent results. We report on the infrastructure we have developed, the goals and content of our initial course offerings at both schools, and an evaluation of our success thus far.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; K.3.2 [**Computer and Info Science Education**]: Curriculum

Keywords

Embedded systems education, Nexos, Embedded Xinu

1. INTRODUCTION

Embedded systems comprise a large and growing segment of the computing sphere, but efforts to prepare students for work on design and implementation in an embedded context face a number of important obstacles.

First, embedded systems are by their nature quite diverse, both in scope and in function. Typical embedded systems can range from 4- and 8-bit microcontrollers, with hundreds or a few thousand bytes of storage, to full-fledged high-end processors with multiple cores and gigabytes of storage. System requirements can range from low-power, event-driven operation, to gigabit-speed hard real-time deadlines, and can include everything in between. In short, the diversity of embedded systems dwarfs the variation found in desktop and server computing, and this enormous divergence can impact every aspect of system development.

Second, embedded systems traditionally cannot make use of the most powerful compilation, debugging and automated testing tools available for production desktop systems. Their input/output channels are comparatively narrow, they often lack sufficient spare resources to support costly profiling or instrumentation, and their design budgets cannot support large virtual machines or elaborate software runtimes.

Third, computer science and engineering degree programs are already well-established at many institutions, and often do not have room for entirely new courses in the core. Furthermore, concentration on traditional system development models does not necessarily transition smoothly to time-oriented, interrupt-driven, and reactive models characteristic of embedded systems.

Finally, many schools lack the financial resources, laboratory space, or faculty expertise to commit to dedicated embedded instructional equipment. With a few notable exceptions, many of the most popular commercial embedded platforms do not provide inexpensive evaluation boards, are supported only by proprietary development tools, and lack the kinds of useful peripherals that would naturally lead to a body of general purpose laboratory assignments.

This paper describes initial results from joint efforts by Marquette University (MU) and University of Buffalo (UB) to develop a curriculum and support infrastructure to address several of these pressing challenges. The Nexos project aims to provide effective, duplicable, modern curriculum assistance to schools looking to incorporate embedded systems components into core computer science and engineering courses throughout the curriculum. Our approach focuses on inexpensive, commodity hardware—the Linksys WRT54GL wireless router family—readily available to both students and faculty. We are developing hardware and software to support both small- and medium-scale laboratory installations centered around the WRT54GL. Our curriculum development efforts have concentrated on developing laboratory assignments, teaching materials, and a supporting web portal to assist other departments interested in adoption. A textbook / laboratory manual is in progress.

The remainder of this paper is organized as follows: a brief outline of prior and related work; a description of the Embedded Xinu operating system at the core of our laboratory environment and its related tools; a description of the content in prototype courses taught at both UB and MU;

WESE '08 Atlanta, GA USA

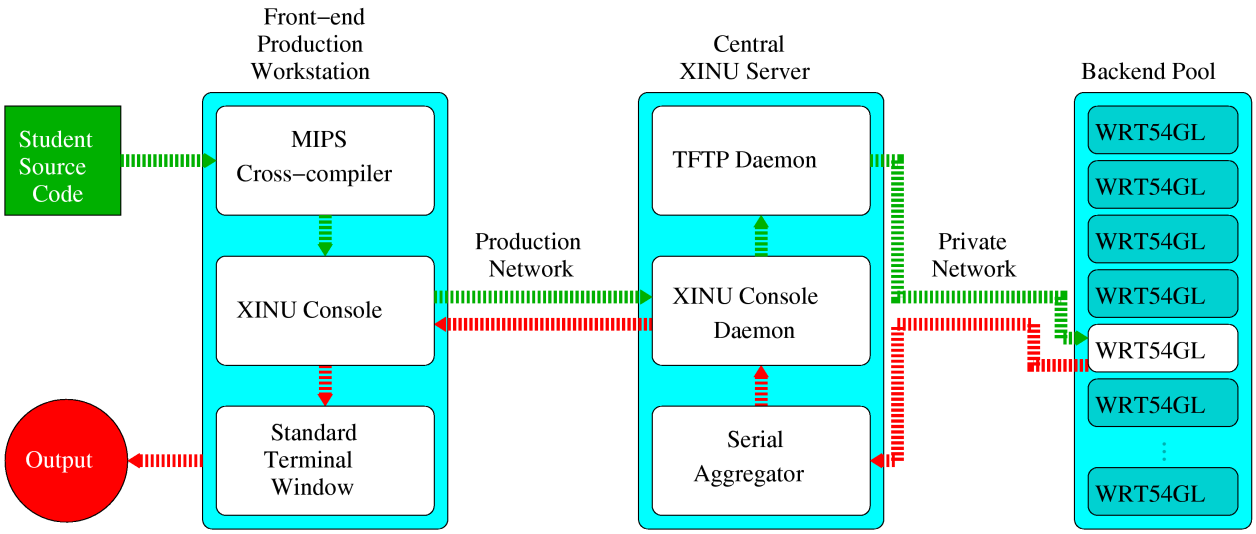

Figure 1: From Student Source Code to Operating System Output

and evaluations of our efforts thus far.

1.1 Prior and Related Work

There has been much recent work on embedded systems education. Koopman et al. of Carnegie Mellon University (CMU) presents an extensive study of the status of embedded system education in [11]. The CMU group discusses the diverse approaches in twelve different embedded application areas, including cross-cutting embedded skills areas such as real-time systems and energy-aware computing, and concludes with a list of lesson learned. The guiding principles for Project Nexos closely follow this list of lessons learned.

Wolf and Madsen [20] discuss embedded systems education for the future and provide a detailed description of their experiences at Princeton University. This paper clearly brings out the importance of hardware-software co-design concepts, recommending C-like languages over an assembly language approach. We concur with Wolf and Madsen that, "next-generation courses in embedded computing should move away from the discussion of components and toward the discussion of analysis and design of systems."

Hamrita et al from University of Georgia [8] reports an interdisciplinary approach that comprises four different courses in robotics, embedded systems, and two courses on microcontrollers. This is an intensive curriculum in embedded systems and would be very hard to adopt for a liberal arts degree in computer science. A summary of the 2005 Workshop on Embedded System Education (WESE 2005) [10] covers a list of presentations on existing embedded systems courses from all over the world. Most of these presentations refer to quite traditional courses.

A variety of platforms have been proposed for embedded systems curricula. Ricks et. al [17] use an aerospace processor to pursue substantially similar teaching goals to our own. Others have focused on robotics platforms [8], dedicated digital signal processors [1, 7], and more familiar microcontroller systems [5]. We believe that a consumer-grade networking applicance like the WRT54GL presents a platform of comparable or better flexibility at a lower cost than much of this prior work, and with a clear path toward reuse for its intended purpose.

Hansson et al. [9] present a graduate-level embedded systems curriculum focusing on a multi-processor system implemented in FPGA. Work such as this highlights the wide diversity of complex topics encompassed by the term "embedded systems". Our work concentrates on single-processor, system-on-a-chip hardware with already defined peripherals in order to make the laboratory topics accessible to undergraduates.

Vahid and Givargis emphasize the need for early introduction of structured, time-oriented programming in the first or second year [19]. We concur that waiting until upper-division electives to begin introducing core embedded concepts is too long – this paper describes UB's sophmore-level course, and an upper-division MU course that builds directly on sophomore-level courses. Vahid and Givargis also present a novel, virtual microcontroller environment that can be used as a starting point in the very earliest courses. A closely related project [18] extends this virtual teaching environment to actual hardware through automatic transformation supported by several real underlying architectures.

Other previous work has emphasized the use of test-driven development (TDD) in embedded systems courses [15], or have presented tools for facilitating embedded system testing [13]. Our project includes automated testing tools both to assist student development and faculty assessment.

The Embedded Xinu kernel is based upon groundbreaking experimental operating system work by Comer in the 1980's [6]. The Nexos Project is based upon MU's own prior work on integrating embedded systems components into existing core courses [4, 3], and UB's prior development of an embedded systems course.

2. LABORATORY ENVIRONMENT

This section describes the Embedded Xinu laboratory environment we have developed to support the experimental emphasis on embedded systems in the new curricula at MU and UB.

2.1 Hardware

As our target platform, we have chosen the ubiquitous Linksys WRT54GL family of wireless routers. These devices are readily available at inexpensive prices, and can already be found in homes, small businesses, and college dorm rooms. They contain a little-endian embedded MIPS 32 processor, operating in the 200-300 MHz range, with 16 MB of RAM, and 4MB FlashROM. Several versions of the router are available, but all have easily accessible serial port connections on the main board that allow direct access to the device firmware.

With only minor modifications to the casing and the addition of an RS-232/TTL serial transceiver circuit, the serial console of the WRT54GL becomes accessible to the student. From the serial console, a user can interrupt the normal boot sequence from FlashROM, and substitute an uploaded kernel image from over the local area network (LAN) port. Thus, students can quickly find themselves running their own embedded O/S kernel on the raw hardware, with no emulation or simulation involved. An overview of this process is shown in Figure 1.

WRT54GL routers are on the upper end of the enormous spectrum of embedded devices available to us; their specifications are roughly equivalent to desktop PCs from the early 1990's. Yet, while they are not as resource-constrained as 4- and 8-bit embedded microcontrollers, they are true embedded systems in the modern context. Processor speed and RAM size are orders of magnitude less than typical desktop PCs, and input/output channels to the processor are narrow. They are missing major components that would be found in non-embedded contexts – no hard disk or optical storage, no video adapter, mouse, or keyboard. They provide a variety of interesting peripherals (wireless and wired network interfaces, Flash ROM storage, LED and general purpose I/O pins,) but present realistic obstacles to traditional debugging techniques. The highly resource-constrained operating systems they contain are event-driven systems expected to run indefinitely with no direct interaction from users or administrators. In short, we feel that they are an excellent choice for prolonged experimentation by students with an interest in embedded computing. As an added benefit, they are a RISC architecture commonly taught in lower division computer organization courses, so many students will already have some familiarity with the instruction set.

The modifications made to the platform are non-destructive, and the WRT54GL once again becomes a fully-functional wireless router running its stock system upon reboot. The Embedded Xinu site [2] provides parts lists, diagrams, directions and pictures for making the simple serial modifications required.

This embedded platform can be used in a stand-alone configuration in which a single student computer with a network card and a serial port manages a dedicated WRT54GL router. The advantage of this configuration is simplicity, and it can be realized in both a laboratory setting, or in many cases at home and in dorm rooms. Students in the initial offering of UB's CSE 321 course (described below) followed this route, with many students choosing to purchase their own routers for use in the course. The cost of the hardware, with modifications, is less than a typical textbook, and the platform can be used later for either continued embedded system exploration, or for its original purpose as a home networking applicance. (Or both, with sufficient effort.)

As shown in Figure 1, the platform can also be used in a dedicated pool configuration, where a collection of routers are made available for "checkout" on the network, and students can remotely power, upload, and interact with their embedded operating system kernels from any front-end machine on the network with appropriate connection tools. In the pool configuration, students can make use of the platform without requiring dedicated lab space, and the pool can be used by several different courses simultaneously. The next section describes the software infrastructure we have built to support the pool configuration at MU.

2.2 Software

We have ported the venerable Xinu operating system [6] to the embedded MIPS32 processor, and have built appropriate device drivers for several of the key peripherals on the WRT54GL platform. The source code is freely available under a BSD-style license from the Embedded Xinu web portal [2].

Our embedded O/S kernel is small but elegant, and designed to be completely fathomable in a short time by undergraduates working for the first time on embedded systems. The layers of the Embedded Xinu kernel are shown in Figure 2, and include all of the components normally found in an embedded operating system.

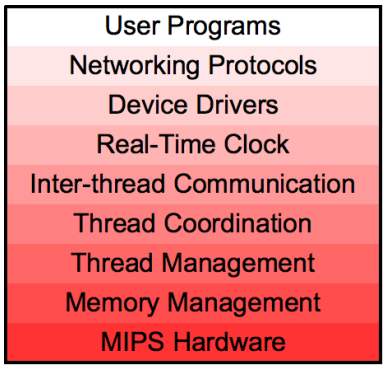

Figure 2: Layers of the Embedded Xinu kernel

Our software can be compiled using freely-available cross-compiler tools that are readily available from the web and mainstream Linux distributions. In the stand-alone configuration, all that is required for software is a cross-compiler and a TFTP server, another component that is freely-available for all of the major desktop operating systems.

For a dedicated pool of backends, additional connection software is required to allocate routers to remote users and to manage special-purpose hardware for remote power control and the large number of serial console connections required. The Embedded Xinu portal supplies parts lists and configuration information for remote power control and network-attached serial annexes; we also supply open source tools for managing the remote connections and controlling the dedicated backend pool.

The MU Systems Laboratory contains a dedicated pool of two dozen backend target routers used for our embedded systems courses and others. The high-level overview of the pool is shown in Figure 1; additional details are available online and in [4].

3. CURRICULUM

This section describes two Nexos Project courses that have been developed in tandem at UB and MU, leveraging the Embedded Xinu infrastructure. The first course, UB's CSE 321, is an embedded systems course normally taken by sophomores as part of the core curriculum. The second course, MU's COSC 198, is an upper-division elective taken by students who have already taken core operating system and hardware system courses with embedded components. While CSE 321 and COSC 198 are two very different courses at two very different institutions, they both leverage a common laboratory infrastructure, and common curriculum elements. The two taken together show the diversity of assignments and course focus that are possible within the Nexos framework. We describe both courses in detail below.

3.1 Embedded Systems Course at UB

The Computer Science and Engineering department (CSE) at University of Buffalo has decided to address the need for a new course to bridge the gap between lower level courses in data structures and computer organization and higher level courses such as operating systems. A new course in embedded systems has been created to bridge this gap, and to strengthen the operating systems curriculum. While the traditional operating systems course is offered at the senior level, the new embedded system course will be taught at the sophomore level. This model enables students to draw ideas from modernized lower level courses, and also gives them ample time to research, explore and apply the embedded operating systems concepts in their research and internship efforts in ways that were not possible with a single senior level course. Prerequisite for the course is Data Structures and Algorithms (CSE 250) or an equivalent. The topics covered in this course (CSE 321 [16]) begin at the boundary of application level software and extend to applications in the real world.

CSE 321 was offered for the first time in fall of 2007 as a pilot. The curriculum for the course is strongly founded on fundamentals that are often overlooked in the modernization of courses. At the same time, the curriculum also exposes current trends and concepts enabling modern innovations and devices such as sensors, heart pace makers, digital music players, and communication devices. The course pedagogy includes four components:

1. A core component that provides coverage for fundamental concepts;
2. An extension that allows room for applied concepts;
3. Field visit(s) to a local industry; and
4. Class visit(s) featuring Q&A sessions with a distinguished scientist in a relevant area.

The core component deals with coverage of fundamental concepts that are expected to be common among the course offerings at various institutions. However the extension component can be customized to meet curricular needs and to reflect the areas of specialization of the host department.

Course topics include resource management, concurrency, secure coding practices, memory management, timeline design and analysis using metrics and schedulability tests, hardware interfaces, device driver programming, memory maps and boot kernels, firmware and ROM-resident system code, communications and networking, and debugging live systems. Intellectually, this course advances considerations of computer system architecture, multi-threaded control, fault tolerance, the translation of requirements into a well-partitioned software architecture and practical design, the subsequent translation into code, the documentation of technical ideas (promoting writing skills), and strategies for system configurability, hardware state tracking, and safety. Class visits by an expert are recommended by Koopman et al in their survey paper [11] to add real impact to concepts like requirements and tradeoffs discussed in class. Field visits are modeled after the field-trips that K-12 curricula have implemented for many years with great success.

The initial offering of CSE 321 used Laplante's *Real-Time Systems Design and Analysis* [12], because of its even-handed presentation of fundamental real-time system topics. Material was also drawn from Liu's *Real-Time Systems* [14], because of the ample examples presented, and the strong, hard real-time scheduling theory component.

A high level description of the curriculum is given in Figure 3. Each of the four components are described in greater detail below. The second column of the table shows a list of hands-on laboratory exercises that are carried out by students on the Embedded Xinu / Linksys WRT54GL environment. These laboratory exercises correspond closely with MU's operating systems course [4]. Thus the implementation in Figure 3 illustrates the feasibility of retrofitting an existing course curriculum with an Embedded Xinu-based lab environment and exercises. Educators may choose to customize this framework to suit their curricular needs.

3.1.1 Objectives

Given the wide diversity of embedded systems courses already in existence, it is important to define the objectives of CSE 321 – the specific skills and knowledge that the authors intend for students to acquire from the course.

First, students completing CSE 321 will be able to identify the unique characteristics of real-time and embedded systems. They will be able to explain the general structure of real-time systems, and define the unique design problems and challenges of real-time and embedded systems. They will apply real-time system design techniques to various software problems presented throughout the term.

Second, students will gain hands-on experience designing, implementing, and testing embedded software. They will work with both familiar, data-driven problems, and with asyncronous, interrupt-driven, and time-oriented problems.

3.1.2 Component I: Core

The core covers the fundamental topics we expect to be common among most Embedded Operating Systems courses. The objective of this component is to lay down a strong foundation for the students, and to sharpen their C programming and basic hardware-level competencies. It covers nuances of memory allocation, deallocation, memory leaks and implications, buffer management and overflow implications, basic device driver design, concurrency and event driven programming. This component also offers an opportunity to discuss open source software development and the GNU GPL (General Public License). The history of the Linksys WRT54GL wireless router itself yields a very interesting case study for this topic.

Topics	Laboratory Exercises/Demonstrations
I. Core	
Operating system fundamentals; Secure coding : pointers, memory management	1: String operations and memory allocation
Buffer overruns: string vulnerabilities	2: Experimenting with buffer overflow
Device drivers	3: Writing and testing a basic device driver
Context switch and scheduling	Introduce Linksys WRT54GL
Interrupt handlers	4: Context switch and scheduling
Open source OS and GPL	Demo: Embedded Linux
Concurrency and process control	5: Process control
Event driven and asynchronous programming	6: Preemption and synchronization
Analysis and optimization for performance, space and power	Demo: analysis and optimizations
II. Extension	
Embedded systems: Memory maps and boot kernels	7: Heap management
Firmware and ROM resident execution strategies	Demo: Show and tell of sample firmware
Wireless devices; Deeply embedded systems: sensor motes and introduction to sensor network	8: Implement and test multi-level device driver
Digital signal processors	
Handheld devices : end-use applications such as Internet appliances, smart devices, GPS driven instruments	9: Design and implementation of basic file system
Debugging live systems	Demo: debugging
Realtime systems; predictability, deadlines and schedulability	Demo: Cardiac pace-maker
III. Field Trip	
To local industry: Atto Tech Inc. and/or Wilson Greatbatch Inc.; Moog Controls Inc.	Students write a report about their visit
IV. Class Visit	
Prominent researcher in the area; industrial practitioner	Students ask questions

Figure 3: CSE 321 Course Topics and Supporting Laboratory Exercises

3.1.3 Component II: Extension

The second component of the course is intended to be a variable and applied section that can be customized to suit the needs of a particular instructor or curriculum. At a minimum, it covers kernel level aspects such as bootstrapping, ROM resident firmware strategies and debugging aspects. For CSE 321, real-time systems concepts were introduced; another institution could, for example, replace this material with an emphasis on sensor networks, or with material on ubiquitous computing.

3.1.4 Component III: Field Trip

Student participation in a college-level field trip begins well before the actual date; students start by writing a report on their personal objectives for the trip and their expectations, and complete the report after the visit. While Embedded Xinu-based laboratory exercises provide hands-on experience, a field trip to an industry that applies the concepts students learned in the course can be a valuable tool to inspire and motivate them toward further study of embedded systems.

3.1.5 Component IV: Class Visit

This final component gives an opportunity for the students to meet an expert in the field of embedded systems, a pedagogical pattern that seems to be quite popular with the students. Question and answer (Q&A) format is used for this component to allow wider student participation and to avoid the monotony of the lecture format.

The combination of modernized curriculum, hands-on laboratory experiments, an industrial field-trip and "ask the expert" Q&A sessions is intended to fill the interest bandwidth of "net generation" students, keeping them motivated to perform well, and ultimately helping them to benefit from the course.

3.2 Laboratory Exercises at UB

The first three lab exercises (Figure 3) aim to review and reinforce the C language skills of the students. Many students do not have adequate background in some important concepts such as memory allocation and memory leaks, particularly when lower level courses have relied on garbage-collected languages like Java as the first language. UB CSE department uses Java for its introductory courses and moves on to other languages (C, C++, and Lisp) in higher level classes. Lectures and labs are supported by in-class demonstrations of concepts and devices. For example, one of the demos involves Linux, the GPL, embedded Linux, and the history of the Linksys WRT54GL wireless router as a popular device for embedded Linux hobbyists. This is followed by a comparison of embedded Linux and Embedded Xinu that brings out the need for analysis and optimization for performance and size. This demonstration partially fulfills the need for qualitative requirements that are critical for many embedded applications.

The third laboratory exercise, writing and testing a basic device driver, begins the experiences specifically aimed at building embedded system design competence. Rather than view a device driver through the lens of a heavily abstracted operating system layer, students work directly with the memory-mapped control and status registers of a serial device to build input and output primitives that they will

reuse for the remainder of the term. This assignment serves not only to practice reading and synthesizing technical documentation for a peripheral I/O device, but also to use C language struct pointers to organize interaction with external devices, a common motif in embedded systems.

The fourth laboratory exercise, on context switching and scheduling, concentrates not on the heavy-weight process boundary abstractions common in desktop operating systems, but rather on the light-weight, resource-conscious register and stack swapping seen in embedded systems without a mature real-time operating system. While the Embedded MIPS processor on the WRT54GL platform has full virtual memory support, the absence of an obvious backing store makes it sensible to concentrate on a shared, physical memory view of thread contexts that has a distinctive embedded flavor not found in typical operating system courses.

The fifth laboratory exercise, on process control, views the system as a finite state machine, with each thread in one of several allowable states. Concentration is on both the design tools for dealing with such a system, and on a realistic implementation of code that produces proper state transitions in an embedded system. This assignment can incorporate any number of scheduling paradigms, ranging from cooperative scheduling or simple time-sliced round-robin up to complex priority queues, offline scheduling, or earliest deadline first real-time scheduling.

The sixth exercise, preemption and synchronization, recasts the finite state machine of thread management with new, time-oriented transitions reflecting fixed thread quanta. Unlike an operating systems course, which would start with high-level timing abstractions already present, the focus in an embedded systems course is on working directly with timer control registers and timer interrupts. In addition, basic synchronization primitives are presented in this newly-concurrent setting. Emphasis is on simple locks and semaphores, suitable for embedded resources, rather than expensive monitor constructs or other highly-abstracted programming language features.

The seventh exercise, heap management, deals directly with one of the most important resource constraint in many embedded systems – allocation of memory. It is highly unusual to see embedded systems running with virtualized allocation and garbage collection facilities, but this is the context most familiar to students who learn to work primarily in Java, Python, or Scheme in prior courses. This laboratory exercise delves directly into the sea of memory pointers, free lists and accounting blocks that comprise memory management in many kinds of embedded system.

The eighth exercise extends the simple, synchronous serial driver to build a fully-buffered, asynchronous serial driver. Working directly with I/O interrupts, hardware FIFOs, and multiple devices, students confront the genuine complexity of interaction between embedded processors and their peripherals. A modern serial UART is itself best described by a finite state machine with some timed transitions, and a simple teletype (TTY) protocol on top of this adds another state machine at a different level of abstraction.

The final laboratory exercise in CSE 321, the basic file system, is again a finite state machine with timed transitions, describing interaction between a client and a backing store. An offline disk file can be used as target with the addition of a block-oriented "serial disk" layer on top of one of the asynchronous serial port drivers from the previous assignment. However, work is also in progress to make use of the FlashROM hardware already present on the platform, a storage medium common in embedded contexts.

Our evaluation UB's first offering of CSE 321 is presented in Section 4. The next section presents the counterpart course offered at MU in the subsequent term.

3.3 Embedded Systems Course at MU

MU's COSC 198: Embedded Systems was offered for the first time in spring of 2008. The course was offered as an elective, with our embedded operating systems course [4] as a prerequisite. As a result, COSC 198 could be considered the advanced level course in a sequence of systems courses that revolve around embedded laboratory experiences; students in COSC 198 had mostly already completed laboratory assignments similar to those described for CSE 321 in the previous section.

The target population of the course was upper division undergraduates and junior graduate students in computer science, computer engineering, and biomedical engineering. In addition to the embedded operating systems prerequisite, most students also had prior coursework in hardware systems, digital logic or computer architecture. Twelve students enrolled in this first offering, a mix of third- and fourth-year undergraduates from the target majors noted above, and a couple of first- and second-year computer science graduate students.

COSC 198 was comprised of three main components: first, a lecture component covering new material on embedded and real-time systems, following roughly the chapters in a recommended textbook; second, a seminar component in which students read, presented, and discussed research publications from venues in embedded systems and related areas; and third, a laboratory component in which student teams designed, implemented, and tested their own new subsystems in the Embedded Xinu framework. Each of these components will be discussed below.

The stated outcomes for this course were that upon completion students would be able to:

1. Read, understand, and present current research papers in the area of embedded systems;

2. Design, implement, and test their own embedded system components for integration into a larger system; and

3. Document complexities of hardware/software interaction in their embedded system components in sufficient detail that the work can be understood and replicated by others.

Design of embedded systems can include a startling range of topics, including both hard and soft real-time system analysis, dynamical system characterization and feedback control, various kinds of signal processing, etc. For this course, our emphasis was on implementation of time-oriented and data-oriented state machines for I/O driven systems.

3.3.1 Lecture Topics

Lectures topics in COSC 198 were chosen primarily to introduce new material that would be directly relevant to subsequent research paper assignments, and to laboratory projects. It corresponds roughly to the "Core" component

of CSE 321 developed at UB. Some intentional overlap with prior courses was necessary because of the diverse backgrounds of the students. Engineering majors had already worked with feedback control systems, but computer science majors had not. Conversely, computer scientists had dealt more extensively with asynchrony than the engineers. Some of these distinctions are particular to the respective curricula at MU, and would not be the rule at other institutions.

Given that the primary Embedded Xinu laboratory platform is a consumer networking appliance, student interest naturally turned toward embedded networking technology as laboratory projects were brainstormed. In the final third of the term, much of the lecture material was devoted to intermediate-level networking topics, to support this direction. This took the spot of the "Extension" segment of the curriculum developed at UB. This emphasis on embedded networking was quite advantageous; certainly the platform is well-suited to networking, but the hardware also supports working with other serial-driven peripherals, as well as GPIO-linked devices. But more importantly, low level network protocols are fertile ground for developing examples of the time-oriented, state-driven behavior that we feel exemplifies the core of embedded system design.

Lecture topics in the term included:

- characteristics of embedded systems;
- characteristics of real-time systems;
- soft vs. hard real-time;
- cyclic, asynchronous, and unpredictable systems;
- control systems and feedback;
- digital sampling, error, and resolution;
- scheduling and schedulability analysis;
- worst-case execution time analysis;
- case studies of real embedded/real-time systems; and
- networking state machines (Ethernet, IP, ICMP, UDP, DHCP, TCP).

3.3.2 Seminar Topics

Interwoven with the lecture component of the course, students worked alone and in pairs to read, present, and lead class discussions of current research in embedded systems and related areas. Working from an initial list of possible papers, and then branching out on their own later in the term, students drew papers from such venues as PLDI and SOSP, LCTES, ASPLOS, PODC, USENIX, and SIGCSE. Topics covered included: embedded, real-time operating system kernels; fine-grained locking and lock-free synchronization; designing against interrupt-driven overload; memory protection and isolation in embedded processors; general purpose computing with GPUs; optimizing embedded networking protocol stacks; resource bound and program analysis in embedded systems; and other embedded system laboratories and curricula.

While some of these topics proved to be too ambitious to cover thoroughly in this level of course, most provoked lively class discussion of the challenges facing modern embedded system designers.

3.3.3 Laboratory Projects

For the practical laboratory component of COSC 198, students split into teams of six. In the first portion of the term, the teams worked in parallel on competing implementations of the same prescribed embedded system software components, and then presented their work to the rest of the class in a "bake-off" format. In the latter half of the term, the teams were allowed (in consultation with the professor) to brainstorm their own project goals, and then pursue an assigned subset.

Team work throughout the term emphasized best practice software design principles, including revision control and collaboration tools, testsuite validation, regular design reviews, and group code reviews.

The first laboratory project focused on completing an Ethernet device driver for the Embedded Xinu / Linksys WRT54GL platform. A modern Ethernet device interface is asynchronous, interrupt-driven, and includes direct memory access (DMA) buffer pools and client-side packet pools. Working with such a device is an excellent review of resource constraints and low-level hardware interaction for students that have been away from an embedded context for several terms.

The second laboratory project was to implement a packet sniffer for receiving, classifying, and displaying various types of network traffic. It would be easy to let such an assignment slip into a tediously long list of network datagram details. In order to maintain an embedded focus, only the most common datagram types were examined. Fixed-sized, resource-constrained buffering is emphasized, and it is necessary to coordinate multiple interrupt-driven devices to deliver correct output.

The third laboratory project entailed building support for several low-level networking protols. ARP (Address Resolution Protocol) is one of the key underlying pieces of the modern TCP/IP Internet. Its packet format is simple, but modeling a proper ARP cache requires a small state machine (Figure 4) with a time-oriented transitions between several of the states (dotted arrows).

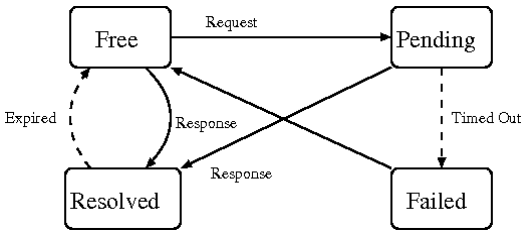

Figure 4: States of an ARP cache entry

Teams implemented enough of ARP and ICMP (Internet Control Message Protocol) to support ping and traceroute primitives from their embedded operating system kernels.

The fourth laboratory project added an implementation of UDP (User Datagram Protocol), the work horse of unreliable network communication. Students implemented enough of UDP to support their own UDP service clients, including DHCP and rdate. DHCP (Dynamic Host Control Protocol) is again interesting particularly because it is best understood as a state machine with timed transitions on several of the edges, as shown in Figure 5.

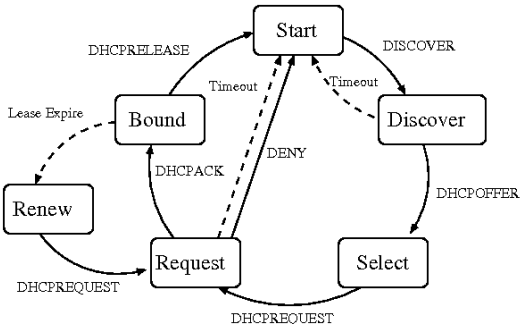

Figure 5: DHCP timed state machine

At this point, the teams went their separate ways based upon their interests. One group proceeded to implement a working subset of TCP (Transmission Control Protocol), the reliability layer of Internet communication, allowing their embedded operating system to establish connections and exchange data with many other external services. The TCP state machine has more than a dozen states, with many more transitions. An embedded implementation requires careful allocation of resources, particularly with regard to reassembly and retransmission. Features like congestion control add complex, feedback-driven timing behavior, and are an excellent exercise in embedded system development.

Another group pursued memory protection, a mechanism using the virtual memory hardware of the underlying architecture to guarantee memory isolation between processes in the embedded operating system. While neither team were entirely successful in achieving their advanced goals, each of the teams met key benchmarks and ended the semester with substantial accomplishment. Both of these final projects are of the level typically found in second-year graduate-level systems courses at some other Universities.

4. EVALUATION

This section discusses the results of various assessment mechanisms used at both UB and MU to weigh the educational value of the new curricula in conjunction with the embedded laboratory enviroment.

4.1 Embedded Systems Course at UB

The prototype offering of CSE 321 was small and comprised only 6 students. UB had ample opportunity to get feedback by closely observing the small number of students, but some assessment instruments lacked both statistical significance and anonymity with such a small enrollment. Next fall's offering is likely to have a normal enrollment of 20 to 30 students, and assessment will continue.

The pilot offering of CSE 321 at UB used the stand-alone configuration for the WRT54GL router platform; each student had a router and associated connectors. Besides the low cost of the hardware, we had some significant observations that are really impressive about the environment: 1) time to set up was small (about 2 hours of student lab time per unit), plus the cross compiler configuration time; 2) portability was high (2 of the 6 students decided to buy their own hardware to work at home since it was less than the cost of many texts) and students could carry their equipment around in a kit bag; and 3) the set-up was self-administered, meaning that we did not need to depend on department system administrators to install and configure the environment for us. This is especially important for the adoptability of Nexos in both large research schools where priorities are elsewhere and in small, remote schools that lack technical expertise to maintain specialized systems.

Evaluation question	Avg
Understand the fundamental components and operation of an embedded system	1.5
Can design and implement an embedded system	2.0
Understand the fundamental components and operation of a realtime system	2.25
Can design and implement a realtime system	2.0
Are able to program using Xinu environment	1.25
Understand the interface between hardware and software	1.0
Understand the interaction of hardware and software	1.25
Are able to get a complete picture of an O/S	2.0
Are able to embed the system you programmed in real hardware	1.75

Figure 6: Student evaluation of course objectives for CSE 321

Figure 6 contains a summary of aggregate results from the special-purpose assessment instrument section relevant to the technical objectives of the course. Responses are reported as averages over a five point scale, ranging from *Strongly Agree* (1) to *Strongly Disagree* (5).

Evaluation question	Avg
Hardware resources appropriate	2.0
Software resources appropriate	1.25
Computer resources adequate for assignments	1.5
Computer resources available and accessible	1.5
Enabled "hands on" embedded system experience	1.25
Enabled "hands on" realtime system experience	2.25
Able to work with Embedded Xinu	1.0
Able to work with WRT54GL platform	1.0
Xinu wiki useful	1.25

Figure 7: Student evaluation of Embedded Xinu in CSE 321

Figure 7 contains a summary from the special-purpose assessment instrument section relevant to the Embedded Xinu laboratory resources used for the term projects. Responses are reported as averages over a five point scale, ranging from *Strongly Agree* (1) to *Strongly Disagree* (5).

Overall, UB successfully duplicated the basic Embedded Xinu environment to teach a course on embedded and realtime systems. Student suggestions for improvement included complaints about the difficulty of soldering wires for the serial transceiver. By the start of the spring term, MU had provided a solution in the form of a printer circuit board design for easy transceiver assembly. UB students frequently consulted the Embedded Xinu Wiki [2] maintained by MU and sent in requests for more information. One of the updates UB requested was a wiki page for shell programming; the prompt MU response is at http://xinu.mscs.mu.edu/Shell.

Students were excited about the application of knowledge from their earlier courses. For example, they designed a UML class diagram for Embedded Xinu, and experienced the effects of single threaded kernel through the errors they encountered. Students explored the hardware on their own by buying parts such as serial-to-USB cables from Ebay and USB drivers from Radioshack.

UB plans an offering of the course for fall 2008, with normal enrollment (20-30 students). This will be supported by an extended, scaled-up version of Embedded Xinu, and a dedicated pool of backend platforms.

4.2 Embedded Systems Course at MU

COSC 198: Embedded Systems was the first offering of its kind at MU, so it cannot be quantitatively compared with predecessor courses. Instead, we rely on three other measures to assess the student learning resulting from the course. First, we have quantitative, direct measure of the practical subsystem deliverables produced from the laboratory component of the course. Second, we have the standard MU course evaluation forms completed by students at the end of the term. Finally, we have the results of a specially produced assessment instrument, jointly developed for the Nexos project by UB and MU, and administered by an independent evaluator. Student participation in the assessment was anonymous and voluntary, and the protocol was approved by both the UB and MU Institutional Review Board (IRB) for research involving human subjects.

The systems produced by the teams in the first group of laboratory projects was either very good or excellent in each case. For each major functionality milestone, teams passed all or all but one of the key testing criteria. In the second, advanced phase of laboratory projects, the teams were not able to pass the majority of testing criteria – however, they had set highly ambitious goals for themselves.

Evaluation question	Median	Decile
The course as a whole was:	5.0	9
The course content was:	4.9	9

Figure 8: Student evaluation of COSC 198

Figure 8 summarizes anonymous student evaluation of the course using the standard MU assessment instrument. Answers were given in a scale from *Excellent* (5) to *Very Poor* (0), and the aggregate score is reported as a median. The decile rank compares the median score of an item in this course with the median scores for all courses at MU in the past two years; a decile of 9 indicates a median score in the top 10% of all courses.

Figure 9 continues the summary of anonymous student evaluations, with all questions phrased to rank *relative to other college courses* the student has taken. The scale runs from *Great* (7) to *None* (0).

Figure 10 contains a summary of the special-purpose assessment instrument section relevant to the technical objectives of the course. Responses are reported as averages over a five point scale, ranging from *Strongly Agree* (1) to *Strongly Disagree* (5). Note that this scale is reversed from the previous assessment tool, and lower numbers are better.

Finally, Figure 11 contains a summary of the special-purpose assessment instrument section relevant to the Embedded Xinu laboratory resources used for the term projects.

Evaluation question	Median	Decile
Learn the conceptual and factual knowledge of course	6.3	8
Develop an appreciation for the field in which course resides	6.8	9
Understand written material in field	6.5	8
Develop ability to express self in writing or orally in field	6.4	8
Understand & solve problems in field	6.5	9
Apply course material to real world issues or other disciplines	6.7	8
General intellectual development	6.8	9

Figure 9: Student evaluation of COSC 198 relative to their other college courses

Evaluation question	Avg
Understand the fundamental components and operation of an embedded system	1.6
Can design and implement an embedded system	1.8
Understand the fundamental components and operation of a realtime system	1.5
Can design and implement a realtime system	1.8
Are able to program using Xinu environment	1.3
Understand the interface between hardware and software	1.4
Understand the interaction of hardware and software	1.3
Are able to get a complete picture of an O/S	1.4
Are able to embed the system you programmed in real hardware	1.5

Figure 10: Student evaluation of COSC 198

Responses are reported as averages over a five point scale, ranging from *Strongly Agree* (1) to *Strongly Disagree* (5).

4.3 Discussion

Both the direct measure and the two indirect measures assessing the Embedded Systems course at MU are uniformly excellent, and indicate that the Embedded Xinu platform enabled an engaging and rewarding learning experience. Students' written comments were similarly glowing, and suggested a greater focus on case studies and application-centered papers for the next iteration of the course.

The indirect measures at UB were similarly impressive, although it is difficult to draw strong conclusions before we have a larger sample. Assessment our curriculum materials and infrastructure will continue with future offerings of these courses, with evaluation of student performance in subsequent courses, and with experiences reported by other schools currently adopting the fruits of the Nexos Project.

5. CONCLUSIONS

We have presented the results of inaugural offerings of embedded systems courses at two Universities, based upon a common laboratory environment to support hands-on experimentation. Our infrastructure focuses on flexible, inexpensive, ubiquitous consumer hardware, and open-source tools that are widely available. We believe this constitutes

Evaluation question	Avg
Hardware resources appropriate	1.3
Software resources appropriate	1.3
Computer resources adequate for assignments	1.3
Computer resources available and accessible	1.3
Enabled "hands on" embedded system experience	1.2
Enabled "hands on" realtime system experience	1.6
Able to work with Embedded Xinu	1.3
Able to work with WRT54GL platform	1.6
Xinu wiki useful	1.9

Figure 11: Student evaluation of Embedded Xinu in COSC 198

a "next generation" over prior work that relies on expensive, proprietary, special-purpose hardware and closed tool sets.

Our courses concentrate on key aspects of embedded system software development: time-oriented design and analysis, resource-constrainted implementations, testing and debugging on real hardware, and interrupt-driven reactive systems. The current choice of platform is particularly well-suited to network applications, but can support a wide variety of interactions over serial ports and GPIO pins.

Embedded systems education covers a fantastic array of topics, and we do not attempt to address development of digital electronics background, hardware/software co-design, FPGA's, real-time operating systems, or domain-specific programming languages. The combination of software-centric embedded and real-time topics we have chosen suits our two university's population of computing majors, and we believe other schools will also find it suitable.

The Nexos Project comprises our curriculum materials, hardware and software tools, and a community web portal to support widespread adoption of the same. Our initial assessments have been extremely promising, and we believe that our approach can help bring embedded systems education to many schools that would not otherwise have resources, space, or expertise to initiate such an undertaking.

6. ACKNOWLEDGMENTS

The authors are grateful for the students in MU's COSC 198 and UB's CSE 321 courses who persevered through our experimentation in the 2007-2008 school year. This work is supported in part by NSF grant DUE-CCLI-0737476 at MU, and DUE-CCLI-0737243 at UB.

7. REFERENCES

[1] M. Benjamin, D. Kaeli, and R. Platcow. Experiences with the blackfin architecture in an embedded systems lab. In *WCAE 2006: Workshop on Computer Architecture Education*, 2006.

[2] D. Brylow. Embedded XINU project, 2007. `http://www.mscs.mu.edu/~brylow/xinu/`.

[3] D. Brylow. An experimental laboratory environment for teaching embedded hardware systems. In *WCAE 2007: Workshop on Computer Architecture Education*, pages 44–51. ACM Press, June 2007. ISBN: 978-1-59593-797-1.

[4] D. Brylow. An experimental laboratory environment for teaching embedded operating systems. In *SIGCSE 2008: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, volume 40, pages 192–196, New York, NY, USA, 2008. ACM.

[5] B. H. C. Cheng, D. T. Rover, , and M. W. Mutka. A multi-pronged approach to bringing embedded systems into undergraduate education. In *Proceedings of ASEE 98: American Society for Engineering Education Annual Conference*, 1998.

[6] D. E. Comer. *Operating System Design: The XINU Approach*. Prentice Hall, 1984.

[7] D. Franklin and J. Seng. Experiences with the blackfin architecture for embedded systems education. In *WCAE 2005: Workshop on Computer Architecture Education*, 2005.

[8] T. K. Hamrita, W. D. Potter, and B. Bishop. Robotics, microcontroller and embedded systems education initiatives: An interdisciplinary approach. *International Journal of Engineering Education*, 21(4):730–738, 2005.

[9] A. Hansson, B. Akesson, and J. van Meerbergen. Multi-processor programming in the embedded system curriculum. In *Proceedings of WESE 2008: Workshop on Embedded Systems Education*, pages 33–40, Oct 2008.

[10] D. J. Jackson and P. Caspi. Embedded systems education: future directions, initiatives, and cooperation. *SIGBED Review*, 2(4):1–4, 2005.

[11] P. Koopman, H. Choset, R. Gandhi, B. Krogh, et al. Undergraduate embedded system education at carnegie mellon. *Transactions on Embedded Computing Systems*, 4(3):500–528, 2005.

[12] P. A. Laplante. *Real-Time Systems Design and Analysis*. Wiley-IEEE Press, 3rd edition, 2004.

[13] V. Legourski, C. Trödhandl, and B. Weiss. A system for automatic testing of embedded software in undergraduate study exercises. *SIGBED Review*, 2(4):48–55, 2005.

[14] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[15] J. Miller and M. Smith. A TDD approach to introducing students to embedded programming. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 33–37, 2007. ACM Press.

[16] B. Ramamurthy. CSE 321: Realtime and embedded systems, 2007. `http://www.cse.buffalo.edu/~bina/cse321/fall2007/`.

[17] K. G. Ricks, W. A. Stapleton, and D. J. Jackson. An embedded systems course and course sequence. In *WCAE 2005: Workshop on Computer Architecture Education*, 2005.

[18] S. Sirowy, D. Sheldon, T. Givargis, and F. Vahid. Virtual microcontrollers. In *Proceedings of WESE 2008: Workshop on Embedded Systems Education*, pages 57–62, October 2008.

[19] F. Vahid and T. Givargis. Timing is everything – embedded systems demand early teaching of structured time-oriented programming. In *Proceedings of WESE 2008: Workshop on Embedded Systems Education*, pages 1–9, October 2008.

[20] W. Wolf and J. Madsen. Embedded systems education for the future. *Proceedings of the IEEE*, 88(1):23–30, January 2000.