An Introductory Capstone Design Course on Embedded Systems

Jeff C. Jensen
National Instruments
Austin, TX 78759-3504
E-mail: jjensen@ni.com

Edward A. Lee
EECS Department
University of California, Berkeley
Berkeley, CA 94720
E-mail: eal@eecs.berkeley.edu

Sanjit A. Seshia
EECS Department
University of California, Berkeley
Berkeley, CA 94720
E-mail: sseshia@eecs.berkeley.edu

***Abstract*—We review an introductory course in embedded systems that characterizes embedded systems not by resource constraints, but rather by interactions with the physical world. This course teaches students the basics of models, analysis tools, and design for embedded systems. Traditional undergraduate courses in embedded systems focus on ad-hoc engineering practices and the use of existing modeling techniques, often omitting critical analysis and meta-modeling; we emphasize model-based design of embedded and cyber-physical systems. Students learn how to model the physical world with continuous time differential equations, and how to model computation using logic and discrete models such as state machines. Students evaluate these modeling techniques through the use of meta-modeling, illuminating the interplay of practical design with formal models of systems that incorporate both physical dynamics and computation. Students learn formal techniques to specify and verify desired behavior. A combination of structured labs and design projects solidifies these concepts when applied to the design of embedded and cyber-physical systems with real-time and concurrent behaviors.**

I. Introduction

We share an interdisciplinary approach to teaching embedded systems that deviates from the more traditional approach of characterizing embedded systems by resource constraints [1]. While resource constraints are an important aspect of embedded systems design, such constraints are part of every engineering discipline and give little insight into the interplay between computation and physical dynamics. Our approach challenges students to draw from topics in physics, circuits, transducers, digital signal processing, digital communications, networking, operating systems, robotics, control theory, algorithms, probability, and logic. Students are exposed to the lowest levels of abstraction for programming embedded systems, including traditional imperative programming models, to the highest levels of abstraction, including graphical system design tools and concurrent models of computation. Here, we focus on the hands-on aspects of the course, partitioned into structured laboratories and a capstone senior design project. The theoretical foundation for the course follows Lee and Seshia, *Introduction to Embedded Systems: A Cyber-Physical Approach* [6].

The course EECS 149 [2], "Introduction to Embedded Systems," at the University of California at Berkeley is targeted at advanced undergraduate juniors and seniors in Electrical Engineering and Computer Science. Prerequisites include an introductory course in signals and systems, an introductory course in computer architecture (which covers both C and assembly programming), and an introductory course in discrete mathematics. While these prerequisites establish a common language for the technical aspects of embedded systems, we believe that the ubiquitous and interdisciplinary nature of embedded systems requires students to investigate topics beyond computer science.

Cyber-physical systems [3] are dynamic systems that integrate physical processes with computation, often in feedback loops, where physical processes affect computations and vice-versa. *Model-based design* [4], [5] emphasizes mathematical modeling to design, analyze, verify, and validate dynamic systems. Mathematical models are used to design, simulate, synthesize, and test cyber-physical systems, and are based on system specifications and analysis of the physical context in which the system resides. A complete model of a cyber-physical system represents the coupling of its physical processes and embedded computations. Design of these systems requires understanding of these joint dynamics, which is the focus of our course.

II. Structured Laboratories

The course begins with a series of structured laboratories, spread across weekly, three hour laboratory sessions. Each laboratory consists of a pre-laboratory assignment, in-laboratory exercises, and a formal writeup, and starts with a brief lecture that covers the instruments used, their theory of operation, and the overall goal of the laboratory. In teams of two or three, students follow prompts from a laboratory guide that lead them towards developing a solution; these solutions are not unique, and teams are encouraged to experiment and innovate.

In addition to traditional C programming, we use LabVIEW [7] (National Instruments) for data acquisition, simulation, and embedded programming. As a graphical system design tool, students see embedded programming at a higher level of abstraction than traditional imperative programming models. LabVIEW supports heterogeneous compositions of several models of computation: continuous systems are expressed as ordinary differential equations or differential algebraic equations, and discrete systems are expressed as difference equations, in the LabVIEW Control, Design, and Simulation Module; concurrent state machines are expressed in models created in the LabVIEW Statechart Module; imperative expressions are expressed as Formula Nodes (a subset of ANSI C) or MathScript Nodes (compatible with scripts created by developers using The Mathworks, Inc. MATLAB software and others); data acquisition and program flow are expressed in structured dataflow, which is general enough to allow the composition of each of these models of computation [9]. While many graphical design tools exist, LabVIEW is attractive because of its adoption of *platform-based design* [10], enabling portability of LabVIEW applications from desktop computers to embedded controllers, wireless sensor networks, or arbitrary targets through synthesis of ANSI C code.

Like most graphical system design tools, LabVIEW implements a limited number of models of computation for the purpose of serving industry in the design, prototyping, and implementation of embedded systems; while this enables students to quickly deploy embedded software, it is difficult (if not impossible) to compare and critically analyze variations of each model of computation, or

to incorporate entirely new models of computation. To better serve the exploration and critical analysis of formal models of computation, we assign homework on the Ptolemy II [11] modeling and simulation environment. Ptolemy II is a versatile tool for researching heterogeneity, allowing developers to easily create and simulate new models of computation. While we would prefer the use of a single, all-encompassing modeling environment, we feel that many concepts of model-based design an demonstrable through one or both of these softwares.

A. Week 1: Introduction to LabVIEW

The first laboratory session is a hands-on introduction to the LabVIEW graphical system design environment [8]. Students learn the basics of structured dataflow, the concept of a virtual instrument, and implementations of multiple models of computation, specifically continuous systems, Statecharts, MathScript, and Formula Nodes. Students readily grasp the text-based models, but often struggle with representations of continuous systems if they have not taken a course in modeling or control theory.

B. Week 2: Sensor Modeling

The goal of this laboratory is to perform computer-based measurements. Students first interface to sensors in a controlled environment using a desktop computer, an NI USB-6009 [12] USB data acquisition device, and LabVIEW. Floating-point voltage measurements are continuously acquired from an Analog Devices ADXL-322 [13] two-axis analog accelerometer. Referencing the sensor datasheet, students translate these voltage signals into meaningful units such as g-forces. The first objective is complete when students calibrate the sensor, convert measurements into meaningful units, produce a real-time graph of each axis scaled to $\pm$1g, and calculate pitch and roll in degrees.

Now familiar with the accelerometer and how it is measured, students interface to an embedded device, the Nintendo Wii Remote [14]. The device, part of the Nintendo Wii entertainment system, is a handheld wireless game controller that measures acceleration and light intensity to determine motion and position, as well as momentary depressions of user buttons. The three types of actuators on the Wii Remote are a speaker, LEDs, and a rumble unit that vibrates the device. While the Wii Remote is simple, popular, and widely available, its most compelling feature is that can serve as a conceptual bridge from computer-based to embedded-based measurement. The remote contains a three-axis version of the accelerometer that students previously interfaced, and since its measurements may be transmitted wirelessly to LabVIEW, the exercise isolates the challenge of requesting a measurement from an external embedded device and interpreting its fixed-point (integral) response. We feel that students should be comfortable in this process before advancing to programming of an embedded device.

An onboard embedded processor on the remote contains an analog-to-digital converter (ADC) to sample the accelerometer and a radio to broadcast Bluetooth messages to a host. An online community of embedded systems hackers reverse-engineered and documented these messages [15], and students navigate this community to learn these messages and how to interface the remote to a desktop computer. Students write a LabVIEW application to toggle LEDs and enable or disable rumble. Finally, the previous accelerometer calibration and measurement program is modified to poll accelerometer data from the remote and display it in real-time on a desktop computer, taking note that measurements are no longer read as floating-point voltages, but instead as binary values output by the ADC on the remote.

Fig. 1. Cal Climber cyber-physical system.

C. Weeks 3-4: Microcontroller Programming in C

In this exercise, students advance to programming an embedded system. The Cal Climber [16] (Fig. 1) is a cyber-physical system based on a commercially available robotics platform similar to the the iRobot Roomba autonomous vacuum cleaner. The off-the-shelf platform is capable of driving, sensing bumps and cliffs, executing simple scripts, and communicating with an external controller. The Cal Climber is comprised of the iRobot Create [17], a Texas Instruments LM3S8962 [18] ARM microcontroller, and an Analog Devices ADXL-322 two-axis analog accelerometer. The Cal Climber demonstrates the composition of cyber-physical systems, where a robotics platform is modeled as a subsystem and treated as a collection of sensors and actuators located beyond a network boundary.

The problem statement is as follows [19]:

> *Program the Cal Climber to autonomously climb a hill. Your robot must determine the correct orientation to drive towards the top of a hill, avoid driving off a cliff, and navigate around obstacles that may be present along the way.*

Students begin with a template C program that implements a simple state machine with two states, DRIVE and TURN. The template program simply drives for a fixed distance, then turns through a fixed angle, and repeats, where distance and angle are reported by the iRobot Create. As a pre-laboratory exercise, students review the template program, including the state machine architecture, a communication queue, interrupts, and interface to the ADC. They review datasheets for the embedded controller, the accelerometer, and the iRobot Create.

This exercise is, for many students, their first experience programming an embedded target, performing calculations without the use of floating-point arithmetic, writing software to interact with the real world (rather than process information), or executing non-terminating software in the absence of an operating system. We seek to reinforce the concept of model-based design by providing a template that implements a state machine, and students generally solve the problem through natural extensions of this state machine. Students decide whether to periodically poll sensors on the robot, or if the robot should be configured to stream sensor data to be processed via interrupts; similarly, the accelerometer can be periodically polled, or the ADC can be configured to run at a fixed rate and trigger an interrupt when conversions are complete. Students consider the design

implications of each of these approaches.

The first objective is to program the robot to avoid obstacles while driving in a fixed orientation. Software must respond to collisions detected by momentary bump sensors on the front of the robot, and to cliffs detected by infrared distance sensors on the bottom of the robot. In the second week of this laboratory, the robot is programmed to navigate to the top of a hill using feedback from the accelerometer. Many students in this course have not formally studied control theory, but the physical dynamics of this system are straightforward and students are challenged to derive a simple control algorithm to navigate a hill. Students complete the first objective when their robots navigate a course of rearrangeable obstacles.

Students often do not consider that affecting the wheels of the robot produces acceleration that is registered by the accelerometer, potentially causing chattering or unexpected behavior. Students find the cause of the undesired behavior and develop a strategy to counteract it, such as implementing a lowpass filter, or limiting acceleration by slowly increasing wheel speeds. This struggle emphasizes the joint-dynamics of cyber-physical systems, and informs later discussions of more advanced modeling and simulation techniques that are capable of predicting this behavior.

Students are encouraged to compete for the fastest run through a course of an incline with obstacles. Many groups stay beyond the official laboratory session to experiment with new approaches, tune controllers, and compare solutions.

D. Weeks 5-6: Model-Based Design for Hill Climbing

Moving to higher levels of abstraction, students use LabVIEW to program the Cal Climber. The LabVIEW Embedded Module for ARM Microcontrollers synthesizes C code according to structured dataflow, Statechart, or continuous semantics, and seamlessly compiles and downloads an ARM binary to the embedded microcontroller. Debugging sessions may be established via JTAG, serial, or ethernet, allowing students to display program data on a host computer.

The problem statement is as follows [20]:

> *Use LabVIEW to simulate the control of the Cal Climber as it climbs a hill, generate C code that targets the LM3s8962, and program the Cal Climber to navigate and climb a hill.*

Students begin with a template LabVIEW project that supports both desktop simulation and code synthesis for the embedded target. Sensor events from the iRobot Create are simulated, and students verify correct state changes in software; for example, when a bump sensor is triggered, the application should respond with a collision avoidance algorithm. Simulations of the accelerometer and ADC enable students to verify controller behavior. The first week of this exercise is complete when students implement a state machine, control algorithm, application logic, and a basic simulation test is passed.

The solution to the laboratory is fundamentally different when developed in LabVIEW because of the inherent concurrency of structured dataflow. Students define parallel loops (processes) that perform tasks at varied rates; for example, sampling and filtering the accelerometer at 200 Hz, reading the iRobot Create sensors at 67 Hz, and executing a control algorithm and wheel commands at 40 Hz (Fig. 2). The higher level of abstraction focuses students on software timing and concurrent behavior instead of low-level constructs like system clocks and interrupts. The obvious tradeoff is that students loose a certain visibility into the low-level aspects of software, but we expect that the previous, C-based lab gives reasonable exposure to traditional low-level programming.

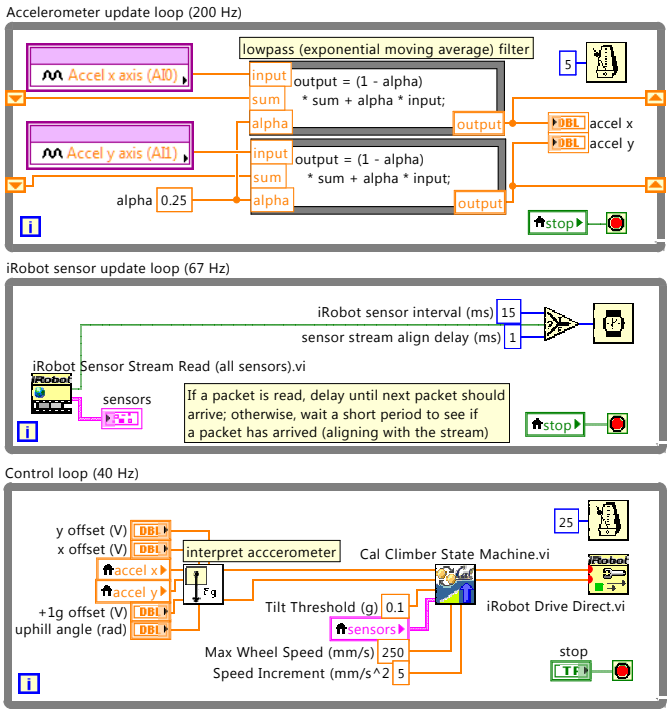

Fig. 2. Cal Climber LabVIEW software.

The following week, students program the Cal Climber. No modification is needed to migrate the previous LabVIEW desktop simulation to the embedded controller, so any necessary code modifications relate to the fidelity of the simulation and modeling of the physical system. Having first verified software through simulation, most students complete this exercise with relative ease.

III. Capstone Design

The remaining eight weeks of the course are dedicated to capstone design projects. Each project culminates in a demonstration session and poster presentation, and satisfies a senior design requirement for graduation from the department of Electrical Engineering and Computer Science. Though we encourage students to choose from a list of suggested projects, many students opt to design their own projects from scratch, using concepts learned throughout the course.

Capstone design projects are kicked-off with a session on project management [21] led by an experienced project manager. The goal is as much to buoy student success as it is to prepare them for interactions with project managers in industry. Students submit a one-page charter that is an overarching project specification; they later submit a project plan of action, a timeline for milestones, and a division of responsibilities. Projects must apply at least two of the following concepts: concurrency, modeling of physical dynamics, reliable real-time behavior, modal behavior governed by finite state machines coupled with formal analysis, real-time networks, simulation strategies, and design methodologies for embedded systems design.

Team are paired with a mentor with relevant experience who is a professor, graduate student, researcher, or industry professional. Progress is checked weekly, alternating between in-class presentations and one-page milestone reports comparing progress to the original project charter. Halfway through the project, students are asked to host live demonstrations during a department open-house,

which encourages students to achieve functional milestones. Students periodically submit peer evaluation forms to identify any issues with a particular team member; to our surprise, students are often more critical of themselves than their teammates. Project management is a crucial component of the capstone design project, and we refer the reader to Koopman [22] for his complementary treatment of software engineering practices used in embedded systems courses.

IV. Results and Further Research

This is a work in progress, and the actual impact of the course is difficult to measure objectively. Students are generally interested and engaged, celebrating their projects as well as those from other teams. Students are proud of what they accomplish, and even post project presentation videos to the internet. Final project submissions include an autonomous helicopter, a distributed autonomous network of Cal Climbers retrofitted with custom optical emitters and sensors for localization, a vehicle convoy, a body sensor network and robotic emulator, interactive games, facial recognition and tracking, several implementations of collision avoidance, and distributed music. Project presentation videos are available on the course website [2]. Such anecdotes give some insight into the impact of the course, but how do we know for sure whether a particular change in the course or laboratory design is actually an improvement? We are pleased, at least, to witness through the course that students surprise both themselves and their instructors, that projects demonstrate an understanding of the theoretical concepts introduced in lecture, and that students have received job offers from industry mentors.

A common issue faced by students is the need to localize autonomous mobile robots. Significant portions of projects have dealt with this issue, with each team attempting a different method. To alleviate this problem, we acquired and are in the process of testing an in-laboratory localization system. The system tracks robots that have been fitted with optical markers whose coordinates are broadcast wirelessly. The system will be available to students for design projects in the next offering of the course, providing an infrastructure to aid the study of distributed systems.

We are investigating physics-based simulators that enable full, closed-loop simulations of the Cal Climber. National Instruments Labs has published a robotics simulator that includes a complete model of the Cal Climber, and we are in the process of evaluating this pilot software. This potentially allows students to verify software in a fully simulated environment, including ground that is not inclined but is rough, prompting students to more carefully model the environment and develop a strategy for noise suppression. This would contribute to a more complete form of model-based design.

A kit-based approach to the lab that is commercially supported and costs about the equivalent of a traditional textbook could go a long way towards facilitating export of the laboratory curriculum. The iRobot Create meets these requirements, and greatly influenced laboratory development. We interfaced the internal microprocessor on the iRobot Create with a more advanced external controller, for better or worse, only because it was underpowered and difficult to reprogram, though we have no reason to believe that students benefit educationally from a more powerful processor. We seek a kit that offers mobility, sensing, and actuation together with a suitably powerful and adaptable embedded controller, as a platform for model-based design of cyber-physical systems.

Acknowledgment

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at the University of California at Berkeley, which receives support from the National Science Foundation (NSF awards #CCR- 0225610 (ITR),#0720882 (CSR-EHS: PRET), #0647591 (CSR-SGER), #0720841 (CSR-CPS), #064436 (CNS-CAREER), the U. S. Army Research Office (ARO #W911NF-07- 2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312 and AF-TRUST #FA9550-06-1-0244), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, Lockheed Martin, National Instruments, Thales and Toyota.

References

[1] E.A. Lee, "Introducing Embedded Systems: A Cyber-Physical Approach," keynote at Embedded Systems Week (ES Week) 2009, *Workshop on Embedded Systems Education (WESE)*. Grenoble, France, October 2009.

[2] E.A. Lee and S.A. Seshia, "EECS 149: Introduction to Embedded Systems." Available: http://chess.eecs.berkeley.edu/eecs149. [Accessed October 13th, 2010].

[3] E.A. Lee, "Cyber Physical Systems: Design Challenges," in International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), May 2008.

[4] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, "Developing Applications Using Model-Driven Design Environments," *IEEE Computer*, vol. 39, no. 2, pp. 33, February 2006.

[5] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-Integrated Development of Embedded Software," in Proceedings of the IEEE, vol. 91, no. 1, January, 2003.

[6] E.A. Lee and S.A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, digital version 1.01. Berkeley, California, 2010. Available: http://LeeSeshia.org. [Accessed October 13th, 2010].

[7] LabVIEW, *LabVIEW User Guide*, National Instruments, June 2009.

[8] National Instruments, "National Instruments LabVIEW Campus Workshop" January 2009 Edition. Part number 351286B-01, January 2009.

[9] J. Kodosky, J. MacCrisken, and G. Rymar, "Visual Programming using Structured Data Flow," In IEEE Workshop on Visual Languages, IEEE Computer Society Press, Kobe, Japan, pp. 3439.

[10] K. Keutzer, A.R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Transactions*, vol. 19, no. 12. December 2000.

[11] J. Eker, J. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity - The Ptolemy Approach," *Proceeedings of the IEEE*, vol. 91, no. 1. January, 2003.

[12] National Instruments, "NI USB-6009 14-bit, 48 kS/s Low-Cost Multifunction DAQ," datasheet.

[13] Analog Devices, "Small and Thin $\pm$2g Accelerometer" ADXL322 Datasheet rev. 0, January 2007.

[14] Nintendo, *Wii Operations Manual* version RVL-S-GL-USZ, Nintendo Corporation, 2009.

[15] Ian Rickard, "Wiimote" website. Available: http://homepage.mac.com/ianrickard/wiimote/wiili_wimote.html. [Accessed October 16th, 2010].

[16] J.C. Jensen, "Elements of Model-Based Design," University of California, Berkeley, Technical Memorandum. UCB/EECS-2010-19, February, 2010.

[17] iRobot, *iRobot Create Owner's Guide,* iRobot Inc., January 2006.

[18] Texas Instruments, "LM3s8962 Microcontroller" DS-LM3S8692-7787 datasheet. September, 2010.

[19] J.C. Jensen and I. Liu, "EE 149: Microcontroller Programming in C; Interfacing Sensors and Actuators with iRobot Create," revision 2.05, University of California, Berkeley, Department of EECS. January, 2010.

[20] J.C. Jensen, "EE 149: Microcontroller Programming in LabVIEW; Interfacing Sensors and Actuators with iRobot Create," revision 1.01, University of California, Berkeley, Department of EECS. January, 2010.

[21] C. Brooks, "Project Management and Embedded Systems," lecture for EECS 149 at the University of California at Berkeley, Department of EECS. March, 2010.

[22] P. Koopman, *Better Embedded System Software*, 1st ed. Pittsburgh, PA: Drumnadrochit Education, 2010.