Computers Interacting with the Physical World: A First-Year Course

Roger D. Chamberlain
Ron K. Cytron
Doug Shook
Bill Siever

Roger D. Chamberlain, Ron K. Cytron, Doug Shook, and Bill Siever, "Computers Interacting with the Physical World: A First-Year Course," in *Proc. of Workshop on Embedded and Cyber-Physical Systems Education* (WESE), October 2018.

Dept. of Computer Science and Engineering
Washington University in St. Louis

Computers Interacting with the Physical World: A First-Year Course*

Roger D. Chamberlain, Ron K. Cytron, Doug Shook, and Bill Siever

Dept. of Computer Science and Engineering
Washington University in St. Louis, St. Louis, Missouri, USA
{roger,cytron,dshook,bsiever}@wustl.edu

Abstract. Most introductory embedded systems offerings are upper-division courses. At Washington Univ. in St. Louis, embedded systems principles are introduced in a first-year course that is required for both computer scientists and computer engineers. This paper describes the motivation for the course, its content, the pedagogical techniques used, and lessons learned while developing and administering the course.

Keywords: First-year course · Cyber-physical systems.

1 Introduction

In the spring of 2015, the Dept. of Computer Science and Engineering at Washington Univ. in St. Louis decided to alter its first-year course sequence for computer science and computer engineering majors. While the first course follows the CS1 curriculum from the ACM/IEEE [6], our second semester course had a focus on thread-based parallelism. The faculty decided that material should be moved to a later year and that a new course should focus on how computers interact with the physical world. The new course was offered concurrently with the previous course in the fall of 2015 and has been offered every semester since. It is required for all computer science and computer engineering majors, and is a technical elective to computer science minors and electrical engineering majors.

This paper describes that course: what we teach, how we teach it, and what we have learned in the process. The course content was motivated by the core concepts that are foundational to both computer science and computer engineering while simultaneously being critical to interactions with the physical world. This includes information representation (e.g., digital inputs and outputs represented at the bit level, analog input and output values as non-negative binary integers), timing (*when* something happens as a functional property), automata models (finite-state machines), etc.

The course is lecture-free, but meets twice a week in the instructional laboratories. One meeting is devoted to studio, where students perform guided exercises through the material we expect them to master. The other session provides opportunities for assistance on assignments and is used for assessment purposes.

* The authors would like to acknowledge the generous support of the Larsen family, which has been instrumental in the development of this course.

The students work on an Arduino Uno, which has an 8-bit microcontroller programmed in C/C++, and a traditional computer programmed in Java. There are two significant advantages to using the Arduino platform. First, it has a supportive maker/hobbyist community, which can motivate students. Second, its simplicity is conducive to learning computer architecture and machine/assembly language. While there are a plethora of texts available within the Arduino ecosystem, they primarily target the hobbyist community. Since our course is intended to focus on core concepts, we have authored a text [2] for use with the course, which has the obvious advantage of being well matched to the course goals.

The resulting course is suited to the common needs of both computer science and computer engineering students. In addition, we believe it is well positioned as an important course for any student who wishes to be well educated in cyber-physical systems, independent of major. The subject matter is central to the National Academies' report on cyber-physical systems (CPS) education [9]. Unlike many introductory embedded systems courses, which are upper-division offerings, this is intentionally designed to be a first-year course, accessible after only one semester of introductory computer science.

2 Intellectual Topics

The course provides both exposure to and mastery of a variety of concepts in embedded systems and CPS. Below we list the topics covered. An asterisk (*) indicates topics in which students are expected to gain significant mastery while non-marked items indicate concepts with lesser depth.

2.1 Information Representation

- Binary, hexadecimal, unsigned, and 2's complement representations*
- Non-integer numbers: fixed point and IEEE-754 floating point
- Character representations: ASCII* and Unicode
- Character strings, both as objects and as null-terminated C-style strings
- Bitwise operations*
- Real-world values as voltages and conversion (e.g., to a temperature)*

2.2 Automata

- Moore style finite-state machines*

2.3 Timing and Events

- Using delays in execution (or sleep) to control time-based behavior*
- Delta timing techniques for either a fixed period or fixed rate*
- Using time-division multiplexing of a shared signal line
- The basic structure of an event loop for event-driven programming*

2.4 Circuits Principles, Physical I/O, and User I/O

- Physical buttons and the concept of "bounce" on real digital inputs*
- Analog voltages and digitization of voltages (ADC)
- Pulse Width Modulation
- Ohm's Law and current constraints/current limiting*
- Challenges of noisy data (e.g., step detection in accelerometer data)*
- Pixel displays*

2.5 Communications
- Serial data exchanges*
- Non-blocking communication*
- Byte-ordering concerns when exchanging multi-byte entities*
- Basic protocols and the exchange of tagged records*

2.6 Architecture / Assembly Language
- Assembly language for integer arithmetic*
- Registers and register conventions*
- Function call/use protocols*
- Stack use conventions*
- C-style memory segmentation (stack, heap, and text segments)

3 Instructional Delivery

3.1 Active Learning

For the last decade, the department has had a strong commitment to active learning [11]. Evidence for the benefits of active learning is extensive [10], and while active learning is applied broadly, it is particularly compelling in science, math, and engineering design [3], including computer science [1], computer engineering [12], and cyber-physical systems [8].

While active learning is often associated with a *flipped* classroom, the active learning approach we use across the entire first year is *studio*-based [5]. Our studio sessions evolved from observation of our colleagues in Art and Architecture as they work with their students.

Collaboration is important for the following reasons. Studies [7] have shown that women are attracted in greater numbers to the study of computer science when they appreciate the extent to which it is practiced collaboratively. Because the field is largely collaborative, featuring collaboration in our early courses provides a more accurate impression of professional practice to all of our students.

Continual feedback during studio sessions is important to keep groups on track and to reinforce that the material is important to their studies. For our courses, the feedback and guidance is provided by teaching assistants (TA) and by faculty who roam between the groups. The TAs are close in age to the students in the course. Studies have shown that novices are more likely to predict the difficulty of a task for other novices than are experts [4]. Moreover, the TA is more likely to empathize with the difficulty of a task than is an expert.

Finally, the types of problems we pose in studio session are purposefully amenable to multiple solutions and approaches. Examples from the first semester course include designing a flag and anthem for a fictitious country and determining how to draw a Sierpinski triangle. Examples from this course include peak detection (counting steps in accelerometer data) and finite-state machine design for parsing incoming messages from another processor. Students know there is more than one way to solve any of these problems, which liberates them from finding the *right* way and empowers them to find their *own* way.

3.2 Student Assessment

The assessment of students includes studio participation, on-line quizzes, assignments, and three exams. For an individual module on a typical week, students are expected to watch the videos and do any required reading prior to the studio session. A pre-studio quiz (completed on-line) is used to incentivize this requirement. These quiz questions are designed to be straightforward to answer if they have done what was asked of them (e.g., repetition of facts, very simple analysis). Studio is not graded based on correct answers, but rather participation.

The assignment for each module is started on the following laboratory session (on Wednesday following a Monday studio) and is due during the laboratory session one week later. Students are provided with a rubric with the assignment instructions, so they know how their work will be graded. These rubrics often evaluate not just the functionality of a submission (does it work?) but also the approach that students use (did they choose an appropriate implementation?).

The second quiz is due Wednesday evening, the same day that the assignment is due. It is also on-line, and for this quiz the questions are similar in content, scope, and style as those that they are likely to experience on the exams.

4 Modules

The class starts a new module each non-exam week, completing it the following week. There are minor schedule adjustments around the exam, but students generally have one calendar week to complete each module's assignment. Also, the first laboratory meeting (labeled studio 0) is used to familiarize the students with various logistics, such as the development environment (both Eclipse and Arduino IDEs), the code repository, etc. The modules are described below.

Information Representation (Module A) – How information is represented in digital systems, including binary and hexadecimal conversions, integer number representations (including 2's complement), as well as ASCII and UTF text. Introduction to the finite-state machine abstraction, and how to implement a finite-state machine in software.

Microcontroller Platform (Module B) – How to physically implement electrical circuits, a brief introduction to electricity (a physics course is not a prerequisite) and voltage, digital inputs and outputs, and finite-state machine design.

Real-Time Computing (Module C) – Techniques for including time as a functional element in software. Starting with delay-based timing, students progress to delta timing (checking each loop to determine if the desired time has elapsed). This is combined with analog inputs (including the conversion of raw A/D input values into engineering units) and simple filtering (averaging).

Communications (Modules D,E) (2 weeks) – This pair of modules investigates the issues inherent in using a byte stream to communicate between two dissimilar computer systems (the Arduino using C and the desktop PC using Java). Students explore how to serialize multi-byte data types, design protocols that enable synchronization in the face of dropped bytes, and design finite-state machine recognizers to parse incoming messages.

Table 1: Relationship between intellectual topics and modules.

	Time ⟶						
Topic ⇓ Module ⇒	A	B	C	D,E	F	G	H,I,J
Information Representation (§2.1)	■	■	■	■		■	■
Automata (§2.2)	■	■		■		■	
Timing (§2.3)		■	■		■	■	
Circuits & I/O (§2.4)		■	■	■	■	■	
Communications (§2.5)				■		■	
Architecture / Assembly Lang. (§2.6)							■

Multiplexing (Module F) – Introduction to the concept of time-division multiplexing, asking the students to interface to and author the software to drive a 5 by 7 pixel LED display. Real-time computing concepts are reinforced, and students are introduced to the fundamental mechanisms involved in larger, more sophisticated, pixed-based displays (e.g., font design, time multiplexing).

Integrative Project (Module G) – Design and implementation of an integrative project that seeks to reinforce concepts that have been introduced earlier in the semester. A second analog sensor is made available (an accelerometer) for the project. Different projects have been used in different semesters, including the development of a pedometer and a predator-prey game (using the accelerometer to sense the tilt of the microcontroller).

Assembly Language (Modules H,I,J) (3 weeks) – This three week set of modules has an introduction to basic computer architecture and machine language, and then focuses on assembly language, using the simple, 8-bit instruction set. Week 1 covers data representation (especially multi-byte data) and data manipulation (e.g., understanding the need for a carry bit). Week 2 introduces techniques for implementing if-then-else logic and looping structures. Week 3 covers pointers and array access.

The relationship between intellectual topics and individual modules is shown in Table 1. The columns in the table represent modules, and the rows represent topics. There is some overlap between topic names and module labels; however, that typically implies that the specific topic is central to the module of the same name, not that it is the only topic present in the module.

In addition to the modules described above, there are a few modules that have been developed and used in previous semesters, but are not in current use.

Interrupts – Students explore how interrupts literally interrupt currently executing code. In addition, students see how interrupts improve responsiveness to events, like button presses (which can further exacerbate bouncing).

IP Networking – This module introduces students to topics in networking. IP protocols, including both TCP/IP and UDP/IP are covered, as well as domain name service and socket-based communications.

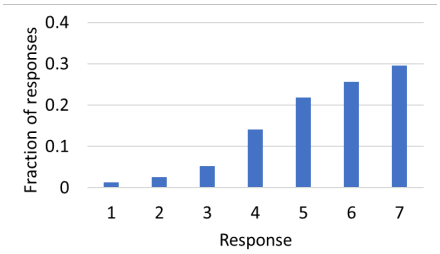

Fig. 1: Student ratings of statement, "The material was covered at a reasonable pace" (scoring: 1 - Strongly Disagree, 7 - Strongly Agree).

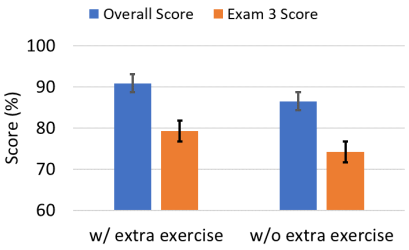

Fig. 2: Relationship between overall scores and exam 3 scores with and without the extra exercise. Bars indicate mean and whiskers std. err.

5 Lessons Learned[1]

5.1 Pacing of Material

As recently as the spring semester of 2017, the pacing of material was uneven and somewhat too fast. The following are quotes from end-of-semester student evaluations.

- "The pace was inconsistent."
- "It moved a little too fast in parts and wasn't always as in-depth as I would've liked."
- "We moved pretty quickly through material and I never felt like I had time to fully grasp concepts because the assignments took so much time."

In reaction to these comments, we've evolved the course into the organization that is described in Section 4. For example, the networking module was (at least temporarily) retired and the assembly language material was converted from a 2 week series into a 3 week series, without expanding the content.

The results of the pace ratings for spring 2018 are shown in Figure 1 (using a 7-item Likert scale). The mean score of 5.47 (std. dev. of 1.41) is very close to the department's average of 5.66, and a substantial improvement from the mean score of 4.51 for the course in spring of 2017.

5.2 Analysis Problems

Based on commentary in the literature that posits the benefits of analysis problems in the context of design courses [13], we were concerned that the present course doesn't have sufficient analysis content (i.e., the bulk of studio questions and assignment questions are design questions rather than analysis questions).

To test this theory, we generated an additional set of analysis problems, designed to be helpful in preparing students for the third exam, and made them

[1] The work is under the oversight of the IRB at Washington Univ. in St. Louis.

available to students one week prior to the exam. To provide an incentive for the students to attempt them, students were told that there would be a small amount of extra credit for those that did well.

We then compared scores for the two groups of students, those who did attempt the extra credit exercise and those who did not. The results of this comparison are shown in Figure 2. There clearly is a correlation between overall course grade and whether or not a student attempted the extra exercise. Those who attempted the extra exercise scored over 4% higher overall (the scores presented exclude the extra credit provided from the exercise). This is statistically significant ($p = 0.02$, non-paired data). When we examine the scores on exam 3, however, the story is different. While the mean score differential is similar (at 5% in this case), the variability in scores is wide enough that this result is not statistically significant ($p = 0.11$, non-paired data), so we cannot rule out the null hypothesis that the difference in the means is due to chance.

Our current opinion is that the correlation seen in the overall scores is likely due to the fact that better students (more likely to achieve a higher score prior to the availability of an optional exercise) are also more likely than their peers to take advantage of an optional exercise, especially when it provides extra credit. Going forward, we are still interested in whether or not additional analysis problems can help students learn the material better, and will likely pursue it by altering the mix of analysis vs. design problems within the studio exercises.

5.3 Logistics

The removal of a live lecture session in favor of on-line videos has been a challenge for some students. Since videos are static content, there is an extra layer of effort that students must take if they do not understand the presented concepts or need additional explanation. Interactions with professors are also fewer and farther between as students are spread out across multiple computer labs instead of meeting as a single, large group.

In the fall of 2017, we introduced a recitation section to address these issues. The recitation sections would occur after the weekly studio, but before the weekly assignment was due, so students could ask questions about topics they had seen in the on-line videos and studio. The effect of these recitation sessions was somewhat mixed, as shown by comments on student evaluations:

- "I only attended a handful of the recitation, specifically when I was confused on a topic, and every time I came my questions were answered and it was very helpful."
- "Kind of a waste of time."

A common complaint is that recitation sessions were short and unstructured. While they were intended to be loosely structured and student-driven by design, a more structured approach with pre-prepared review questions and practice problems may prove to be more beneficial for students.

6 Conclusions

We have described a first-year course that provides foundational material common to both computer science and computer engineering majors. We believe it is helping students discover an interest in computer engineering despite limited prior exposure to the field. Moreover, we believe it provides an introduction to the pervasiveness and breadth of computing early in the curriculum.

References

1. Ahmad, K., Gestwicki, P.: Studio-based learning and App Inventor for Android in an introductory CS course for non-majors. In: Proc. of 44th ACM Technical Symposium on Computer Science Education. pp. 287–292 (2013)
2. Chamberlain, R.D., Cytron, R.K.: Computing in the Physical World. pre-release edn. (2017), `http://www.ccrc.wustl.edu/~roger/cse132/cc_v0_06.pdf`
3. Freeman, S., Eddy, S.L., McDonough, M., Smith, M.K., Okoroafor, N., Jordt, H., Wenderoth, M.P.: Active learning increases student performance in science, engineering, and mathematics. Proceedings of the National Academy of Sciences **111**(23), 8410–8415 (2014)
4. Hinds, P.J.: The curse of expertise: The effects of expertise and debiasing methods on predictions of novice performance. Journal of Experimental Psychology: Applied **5**(2), 205–221 (1999)
5. Hundhausen, C.D., Narayanan, N.H., Crosby, M.E.: Exploring studio-based instructional models for computing education. In: Proc. of 39th ACM Technical Symposium on Computer Science Education. pp. 392–396 (2008)
6. Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society: Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. ACM, New York, NY, USA (2013)
7. Krause, J., Polycarpou, I., Hellman, K.: Exploring formal learning groups and their impact on recruitment of women in undergraduate CS. In: Proc. of 43rd ACM Symposium on Computer Science Education. pp. 179–184 (2012)
8. Marwedel, P., Engel, M.: Flipped classroom teaching for a cyber-physical system course – an adequate presence-based learning approach in the internet age. In: Proc. of 10th European Workshop on Microelectronics Education (EWME). pp. 11–15 (May 2014)
9. National Academies of Sciences, Engineering, and Medicine: A 21st Century Cyber-Physical Systems Education. The National Academies Press, Washington, DC, USA (2016)
10. Prince, M.: Does active learning work? a review of the research. Journal of Engineering Education **93**(3), 223–231 (Jul 2004)
11. Sowell, R., Chen, Y., Buhler, J., Goldman, S.A., Grimm, C., Goldman, K.J.: Experiences with active learning in CS3. Journal of Computing Sciences in Colleges **25**(5), 173–179 (May 2010)
12. Striegel, A., Rover, D.T.: Problem-based learning in an introductory computer engineering course. In: Proc. of 32nd IEEE Frontiers in Education Conference. pp. F1G–7–F1G–12 vol.2 (Nov 2002)
13. Wood, K.L., Jensen, D., Bezdek, J., Otto, K.N.: Reverse engineering and redesign: Courses to incrementally and systematically teach design. Journal of Engineering Education **90**(3), 363–374 (Jul 2001)