MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Spring Semester, 2008

Course notes for Week 1

There will be three handouts issued in 6.01 each week:

1. *Weekly assignment:* This includes work to do in the software lab and the design lab, as well as a two-part homework—one part due before the design lab, one part due before the following week's lecture.

2. *Weekly course notes:* (This handout is the course notes for week 1.) There is no course textbook. Instead, we'll distribute notes that cover the course material as we progress though it week by week. This week there are two distinct sets of note (all in this document: a high-level overview of the goals, approach, and material of the course, and detailed technical notes on Python).

3. *Weekly lecture handout:* This is to help you follow along in lecture and to review afterwards.

All three handouts will be distributed in lecture and also available online from the 6.01 web site `mit.edu/6.01`, linked from the course calendar. You should bring all three handouts with you to the software and design labs.

Course Overview

1 Goals for 6.01

We have a number of goals for this course. Our primary goal is to enhance your ability to solve complex problems by strengthening your skill in the most pervasive strategy for dealing with complexity: using abstraction and modularity. We will examine the use of abstraction and modularity in a number of contexts associated with problems in electrical engineering (EE) and computer science (CS), as we hope to help you develop a more fundamental understanding of abstraction and modularity. We also hope that you will see more than just the similarities in broad perspective, but will also develop skills in using several specific abstraction strategies that have persisted in importance. To accomplish our primary goal, we will study how to analyze and design systems that interact with, and attempt to control, an external environment. Such systems include everything from low-level controllers like heat regulators or cardiac pacemakers, to medium-level systems like automated navigation or virtual surgery, to high-level systems that provide more natural human-computer interfaces.

Our second goal is for you to learn that making mathematical models of real systems can help in the design and analysis of those systems; and to give you practice with building those models. In particular, we hope you will develop insight in to the difficult step of deciding which aspects of the real world are important to the problem being solved, and then how to model in ways that give insight into the problem.

We also hope to engage you more actively in the educational process. Most of the work of this course will not be like typical problems from the end of a chapter. You will work individually and in pairs to solve problems that are deeper and more open-ended. There will not be a unique right answer. Argument, explanation, and justification of approach will be more important than the answer. We hope to expose you to the ubiquity of trade-offs in engineering design. It is rare that a single approach will be best in every dimension; some will excel in one way, others in a different way. Deciding how to make such trade-offs is a crucial part of engineering.

Another way in which we hope to engage you in the material is by having many of you return to the course as a lab assistant. Having a large number of lab assistants in the class means that you can be given more interesting open-ended design problems, because there will be staff readily available to make sure you do not get stuck. Even more important, the lab assistants will question you as you go; to challenge your understanding and help you see and evaluate a variety of approaches. This Socratic method has proven to be of great intellectual value to both classroom students and lab assistants.

Finally, of course, we have the more typical goals of teaching exciting and important basic material from electrical engineering and computer science, including modern software engineering, linear systems analysis, electronic circuits, and estimation and decision-making. This material all has an internal elegance and beauty, as well as crucial role in building modern EE and CS systems.

2 Modularity, abstraction, and modeling

Whether proving a theorem by building up from lemmas to basic theorems to more specialized results, or designing a circuit by building up from components to modules to complex processors,

or designing a software system by building up from generic procedures to classes to class libraries, humans deal with complexity by exploiting the power of abstraction and modularity. And this is because there is only so much complexity a single person can hold in their head at a time.

Modularity is the idea of building components that can be re-used; and abstraction is the idea that after constructing a module (be it software or circuits or gears), most of the details of the module construction can be ignored and a simpler description used for module interaction (the module computes the square root, or doubles the voltage, or changes the direction of motion).

One can move up a level of abstraction and construct a new module by putting together several previously-built modules, thinking only of their abstract descriptions, and not their implementations. This process gives one the ability to construct systems with complexity far beyond what would be possible if it were necessary to understand each component in detail.

Any module can be described in a large number of ways. We might describe the circuitry in a digital watch in terms of how it behaves as a clock and a stopwatch, or in terms of voltages and currents within the circuit, or in terms of the heat produced at different parts of the circuitry. Each of these is a different *model* of the watch. Different models will be appropriate for different tasks: there is no single correct model.

The primary theme of this course will be to learn about different methods for building modules out of primitives, and of building different abstract models of them, so that we can analyze or predict their behavior, and so we can recombine them into even more complex systems. The same fundamental principles will apply to software, to control systems, and to circuits.

2.1 Example problem

Imagine that you needed to make a robot that would roll up close to a light bulb and stop a fixed distance from it. The first question is, how can we get electrical signals to relate to the physical phenomena of light readings and robot wheel rotations? There is a whole part of electrical engineering related to the design of physical devices that connect to the physical world in such a way that some electrical property of the device relates to a physical process in the world. For example, a light-sensitive resistor (photo-resistor) is a sensor whose resistance changes depending on light intensity; a motor is an effector whose rotor speed is related to the voltage across its two terminals. In this course, we will not examine the detailed physics of sensors and effectors, but will concentrate on ways of designing systems that use sensors and effectors to perform both simple and more complicated tasks. To get a robot to stop in front of a light bulb, the problem will be to find a way to connect the photo-resistor resistor to the motor, so that the robot will stop at an appropriate distance from the bulb.

2.2 An abstraction hierarchy of mechanisms

Given the light-sensitive resistor and the motor, we could find all sorts of ways of connecting them, using bits of metal and ceramic of different kinds, or making some kind of magnetic or mechanical linkages. The space of possible designs of machines is enormous.

One of the most important things that engineers do, when faced with a set of design problems, is to standardize on a *basis set* of components to use to build their systems. There are many reasons for standardizing on a basis set of components, mostly having to do with efficiency of understanding and of manufacturing. It's important, as a designer, to develop a repertoire of standard bits and