Sound Part 1: Beyond Bending

Bending the circuit of a “surface mount” device can only take you so far. The accessible components are small and difficult to work with, but, worse, the majority of the components are covered in a “blob” of epoxy. Like singing greeting cards, modern toys contain an amazing amount of electronics inexpensively mounted and secured from competitors (as well as us) in black glue. While we are completely blocked from exploring our toy’s brain, you can follow this link to learn more about how realistic sound is inexpensively generated in greeting cards and toys. But our experiments continue. In previous experiments, we changed the monster’s voice, now we will make it sing a different song. And, to do that, we need a new brain.

In shopping for a new brain, let’s begin by considering what we know about the old one. Look at the schematic of your toy---the one you created earlier. Look at the connections to the IC. Can you group all of the connections into one of these 4 categories: (1) the resistor/capacitor circuit used as a clock, (2) inputs from switches, (3) outputs to control devices such as the speaker, and (4) connections to the battery, both ground and power?

Example Schematic. Can you label each connection to the IC according to its function?

This classification of connections---input, output, power, and clock---generalizes to most microcontroller---the term used to describe our new brain. What’s the difference between an IC and a microcontroller, you may ask. It’s like the difference between an
“internal organ” and a brain. A brain is an internal organ but not all internal organs have the same function nor are they as complicated as the brain.

A **microcontroller** is really a small, inexpensive computer that is typically used to receive input signals and control output devices based on those signals, not unlike the IC in our toy. The big difference between a microcontroller and the IC in our toy is that a microcontroller is programmable!

When selecting a microcontroller you need to consider things such as: cost, processing speed, the number and kind of inputs and outputs needed, the amount of memory needed, and the power that is available. But, as with your toy, the choice of microcontroller has already been made for you; we will be using an ATmega microcontroller. And, just as a brain is easier to work with inside of a head, our microcontroller will come mounted on a development board, the package is called the **Arduino**.

**Meet your Microcontroller**

In selecting the Arduino, we have really selected 3 things: a piece of hardware, a programming environment, and a philosophy. Let’s start with the philosophy. The Arduino is an open-source physical-computing platform. The hardware is open source (i.e., the schematics for constructing the board are freely available) as is the software comprising the development environment. And, like most open source projects, the development remains an on-going community effort to which you can contribute. Finally, **physical computing** is the ubiquitous use of microcontrollers to sense the environment and control devices that respond to the environment. As Massimo Banzi, Arduino Co-Founder, said, “It involves the design of interactive objects that can communicate with humans using sensors and actuators controlled by a behavior implemented as software running inside a microcontroller.”

Now, the hardware. Checkout [this fact sheet](#) and then try your hand at answering the questions below.

1. How many input/output pins are there? Can each of those pins be used for either input or output?


3. What is the clock speed?

4. What type of power does the board require and how is it supplied?

5. Does the microcontroller have erasable memory, in other words can it be programmed more than once?
But It Doesn’t Do Anything

The microcontroller only does what you tell it to do. You have to provide the instructions that bring the brain to life. Your instructions determine whether a connection is interpreted as input or output and the relationship between the two. In the world of electronics, inputs and outputs are electrical signals—current and voltage. In a digital device, such as a microcontroller, signals are interpreted as being either on or off—these are the only 2 states. Indeed, even the instructions you provide must be represented as on-off signals in a digital device. The instructions, i.e., the program, are stored in the microcontroller’s memory as on-off signals grouped into bytes. Each byte is composed of 8 on-off signals called bits. Bytes can also be grouped into larger chunks called words. Any grouping of bits can be interpreted as a number represented in the base 2 number system—a binary number. If you are unfamiliar with these concepts please check the following link to find out more.

We don’t want to write instructions in terms of on/of signals or even binary numbers so we need to choose a microcontroller that comes with an Integrate Development Environment (IDE) that allows us to write instructions (i.e., programs) in a “human-like language” on our desktop computer, translate the instructions into “machine language” (i.e., on/off signals), and then transfer them to the memory of the microcontroller. Fortunately, all this and more comes with the Arduino.

Try It!

Enough talk, let’s try it. The Arduino development environment has already been installed on the computers in this room, but you must use your Linux account to access it. To write your first program, do the following:

(1) Connect your Arduino board to your desktop computer via the USB cable provided. Do not use the USB ports on the monitor as I’m told that they don’t work. Make sure there is a quick blink from the test LED on the Arduino when it is first connected.

(2) Login into your Linux account and open a Terminal Window. Start the Arduino IDE by typing “arduino” (do not include the quotes) at the prompt in your terminal window.

(3) When prompted, choose a folder as the default folder for storing your Arduino programs.

(4) Write a program in the Arduino language.

(5) Compile the program to translate it into machine language, and
(6) Upload the program to the Arduino board.

Of course, step (3) is the hard one. We won’t write our first program; instead, you should download it from the IDE by selecting: File > Examples > Digital > Blink. Look at the program and be prepared to participate in a classroom discussion of the instructions that you see.

To compile and upload your sample program, do the following:

(1) Select Tools > Serial Port and then select the option containing USB in its name.

(2) Select Tools > Board and then select the version of the Arduino that matches your hardware.

(3) Press the “Verify” button on the IDE tool bar to compile your program (i.e., translate your program to machine language)

(4) Now simply click the "Upload" button on the IDE tool bar. Wait a few seconds - you should see the RX and TX LEDs on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar of the IDE.

What happened when your program ran on the microcontroller? The LED built into the Arduino board, depicted in the picture below, should have been blinking. Is this what you expect to happen based on having read the program?

Based on your observations and your analysis of the program, answer the following questions:
1. What is the meaning of the following statement: `pinMode(ledPin, OUTPUT)`?

2. What is `ledPin` in the statement above?

3. What is the meaning of the command `digitalWrite(<pin number>, <state>)`, where `<pin number>` and `<state>` are *parameters*---they stand for the values you must specify when using the command `digitalWrite()` such as `ledPin` for `<pin number>`, and `HIGH` for `<state>`?

4. What is the meaning of the statement: `delay(1000)`? What would happen if the statement where changed to `delay(500)`? Try it, if you are not sure.

**Add Hardware**

Now let’s modify the program. *Blink* controls an LED that is “build-in” to the Arduino board; let’s try controlling an external LED. Create the following circuit using your breadboard. Note that common nodes like “ground” are typically given their own symbol---you will need to connect the circuit to the “ground” port on the Arduino as shown on the wiring diagram.

Run Blink again---what happens? Modify the circuit so that the LED is connected to pin 8 (or any digital pin). What changes to do you have to make to the program to control the LED now? Try it.

For more information on what we just did, I highly recommend the following [link](#).