Introductory Microcontroller Tutorial

This brief tutorial will provide an introduction to working with microcontrollers using the Arduino platform. By the end of this tutorial you will have set up a programming environment on your computer, written several brief functions in C, and built an orientation-controlled servo motor.

1. IDE

Arduino is supported by an all-inclusive Integrated Development Environment. Visit http://arduino.cc/, click on “Getting Started,” and follow the instructions for your operating system. In brief, they are:

(a) Download the software (Windows users should choose the .exe installer)
(b) Run the installer (drag to “Applications,” etc.)
(c) Install the UNO drivers (Windows only; follow the instructions on arduino.cc)

2. Hello, World

The easiest way for a microcontroller to communicate with the outside world is with an LED. Conveniently, the Arduino UNO has a green LED permanently wired to digital pin 13. Use the USB cable to connect the Arduino to your computer, then load the IDE. Before getting to work, we need to make sure the IDE is configured for your microcontroller.

(a) Navigate to Tools → Board and select “Arduino Uno”
(b) Then go to Tools → Serial Port and select the correct port. This can be a little tricky. Some general advice:
   - **Windows:** It’s usually not COM1
   - **Mac/Linux:** The strings are rather cryptic. Process of elimination may help (e.g., the string will not contain the word “bluetooth”). You may need to look at the list of options, close the menu, unplug the Arduino, look again, and note what option disappeared – when you plug the Arduino back in, that’s the port you should choose.

With the setup complete, navigate to File → Examples → 01.Basics → Blink to load the sample code. Press the Upload button (an arrow pointing right, on the toolbar) and the IDE will compile your code and send it to the Arduino. Once the transfer is complete you should see an LED on the board begin to blink.
3. Hello, (Slightly) More Interesting World

Now we’ll modify the code to do something a little more interesting. The blinking is controlled by the last five lines of code, the function that begins `void loop()`. This function repeats for as long as the microcontroller has power. The timing of the blink is controlled by the `delay()` function, which causes the microcontroller to wait the specified number of milliseconds.

(a) Alter the `loop()` function to double the frequency of the blink.
(b) Add more stages to the blink to create a more interesting pattern.

4. Switch-controlled LED

Arduino code is divided into three main sections: declarations, `setup()`, and `loop()`. In the opening declarations section, variables and functions can be declared, and as we will see later, external libraries can be included. The `setup()` function runs once every time the Arduino is powered on. As we have already seen, the `loop()` function then runs in an infinite loop.

The most common configuration to determine the status (closed vs. open) of a switch is a *pull-up resistor*. In this circuit, the voltage across the switch is either 0 Volts or 5 Volts depending on its status.

```
5V
  +

|      |
|      |
|      | 10 kΩ
|      |
|      |

V_{out} -
```

This configuration is so common that Arduino microcontrollers have internal pull-up resistors available to be connected to every digital I/O pin, so the only required external part is the switch. The pull-up resistors are enabled by using the special mode `INPUT_PULLUP`, typically invoked in the `setup()` function:

```c
void setup()
{
    pinMode(pin_number, INPUT_PULLUP);
}
```

Connect one end of a M-M jumper wire to ground, and the other to digital pin 8. This wire is your “switch.” Modify your code to follow this structure:
(a) Declarations: use the variable `terrible_switch` to store the integer 8 and `led` to store the integer 13.

(b) `setup()`: use the `pinmode(pin, DIRECTION)` function to make pin `led` an OUTPUT and pin `terrible_switch` an INPUT_PULLUP.

(c) `loop()`: use the `digitalRead(pin)` function coupled with an `if/else` statement to write a HIGH to pin `led` if the `terrible_switch` pin reads high, otherwise write LOW. The structure of `if/else` is:

```plaintext
if (true_or_false_statement)
{
    // do something
}
else
{
    // do something else
}
```

Compile and upload your code. The LED on the Arduino board (connected internally to pin 13) should now turn on and off when you plug or unplug the jumper wire from pin 8, which connects and disconnects the “switch.”

5. **Sound!**

Connect a small piezo buzzer (shown below) with its red wire plugged into pin 11 and its black wire into GND (for convenience later, use the GND near pin 13). Be gentle; the wires are small. Now modify your code so that, in addition to changing the state of the LED, you also generate a 500 Hz tone for one state, and turn off the sound in the other.

![Piezo buzzer](image)
But how???

The Arduino platform is, at this stage of its development, almost comically well-supported. The tip of the iceberg is available at the Arduino web site. Navigate a web browser to http://arduino.cc, hover over “Learning,” then click on “Reference.” Scroll down the page to find a function (or two) that look promising (hint: look in the right column, under “Advanced IO.”)

Compile and upload the code and enjoy your newfound ability to irritate everyone around you. You may find the buzzer growls instead of turning off. (Ask me why!) You can make it non-audible by adding delay(50); after the no-tone command, to wait 50 milliseconds between updates.

6. This switch is embarassing!

Luckily there is a vast assortment of off-the-shelf components that you can use to accessorize your microcontroller. Time permitting, we have available pre-built capacitive touch switches you can use instead of the wire between GND and pin 8. It is so easy to use it’s almost criminal. The PCB is shown below:

(a) Use M-F jumper wires to connect the GND pin to Arduino GND and VDD to Arduino 5V.

(b) Connect OUT to Arduino pin 8, replacing the wire.

You should be able to leave your code unchanged. The new switch may function differently from the old – in particular, check how pressed/unpressed now corresponds to plugged/unplugged.

7. I need more!

Some further experiments you might do:

(a) Use Serial.begin(9600); in setup() followed by Serial.print("bing!"); in your if-then statement to report a button press out the serial port back to your computer (you can use the Serial Monitor in the Arduino IDE to view incoming serial data; make sure to set the baud rate to 9600 there also).

(b) Add the wire-as-switch back into your design, using another Arduino pin as its input. Write code to send one pitch for the wire, another for the touch sensor, a third for both, and silence for nothing.

(c) Look online for libraries of tone() melodies. Incorporate the code and play some music instead of single tones.