

Arduino in a Nutshell



Jan Borchers

Version 1.8 (Aug 5, 2013)

for Arduino Uno R3 & Arduino IDE 1.0.5

Latest version at: hci.rwth-aachen.de/arduino

ACKNOWLEDGEMENTS



Thanks to Jeff and Drake for playing with the Arduino last night, and almost completing our plans for world domination through an army of robots doing our bidding (insert finger wiggling and evil laugh here). I wrote this booklet in about four hours after getting home last night, and illustrated it this morning. It closely follows our adventures into Arduinoland.

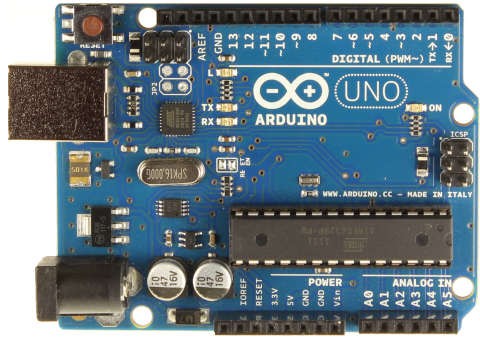
The Arduino team continues to do an awesome job providing this easy-to-use electronics platform, and all schematics were created using the excellent software from *Fritzing.org*. Jim Hollan at UCSD is a great host, and my students back in Germany are bravely suffering through (enjoying?) the time with me away on sabbatical. This booklet is dedicated to Ina who is always wonderfully supportive of her geek.

San Diego, Aug 9, 2012

For great feedback, thanks to the *arduino-teachers*, *Sketching In Hardware* and *i10* mailing lists, especially CTP, David Mellis (Arduino), Gerald Ardito, Jonathan Oser (shieldlist), Linz Craig (SparkFun), Michael Shiloh, Nick Ward, Patricia Shanahan, and Thorsten Karrer!

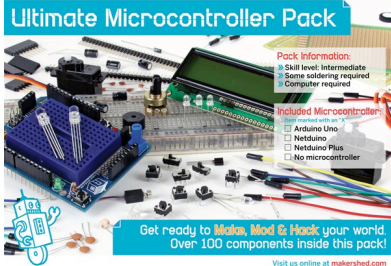
I. INTRODUCTION

The Arduino is a family of **microcontroller boards** to simplify electronic design, prototyping and experimenting for artists, hackers, hobbyists, but also many professionals. Use it as brains for your robot, to build a new digital music instrument, or to make your house plant tweet you when it's dry. Know a little programming, but no electronics? This book will get you started quickly.



Arduinos (we use the standard **Arduino Uno R3**) contain an ATmega **microcontroller** — that's a **complete computer** with CPU, RAM, Flash memory, and **input/output pins**, all on a single chip. Unlike, say, a Raspberry Pi, it's designed to attach all kinds of **sensors**, LEDs, small motors and speakers, servos, etc. directly to these pins, which can read in or output digital or analog voltages between 0 and 5 volts. The Arduino connects to your computer via **USB**, where you program it in a simple language (C/C++, similar to Java) inside the free **Arduino IDE** by uploading your compiled code to the board.


Once programmed, the Arduino can run with the USB link back to your computer, or stand-alone without it — no keyboard or screen needed, just power.



II. GETTING STARTED: BLINK AN LED!

1. Get the **MAKE Ultimate Microcontroller Pack** with an **Arduino Uno**

R3 from makershed.com or your local RadioShack (\$149). Also get a standard **USB A-B** cable and a 9V **battery**. Or, for just the parts we'll use here, get the **Wish List** at sparkfun.com/wish_lists/46366 (\$63). **SparkFun's Inventor's Kit** or **Adafruit's Experimentation Kit** also have most parts we need, and more.

- Download and install the **Arduino IDE** for Mac, Linux or Windows from arduino.cc. Windows users also need to install a driver .INF file, see the website.
- Connect your board via USB. Launch the Arduino app. From the **Tools:Board** menu, select **Arduino Uno**. From the **Tools: Serial Port** menu, select the new serial port (/dev/tty.usbmodem... on Macs). Open the **sketch (program) File:Examples:01.Basics:Blink**. Click the  toolbar button to upload it to your board.

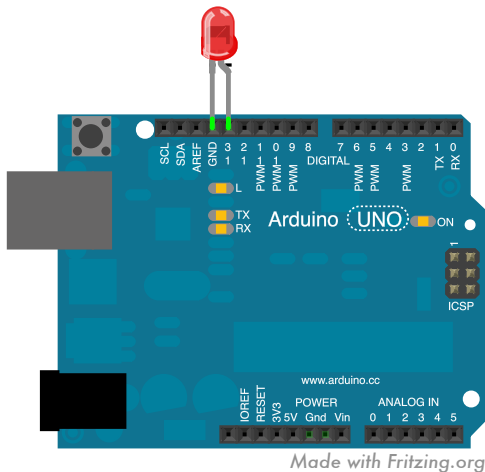
After some flickering, its tiny **yellow LED** should **blink** regularly (1 second on, 1 second off). **You've programmed your first microcontroller!** Change the durations in **delay()** and upload to see the effect.

III. RUN WITHOUT A COMPUTER

1. **Disconnect** the USB cable from your board.
2. Put the **9V battery** into the **battery case** (takes some fiddling).
3. Plug the **barrel plug** from the battery case into the **round socket** on the Arduino, and turn on the **switch** on the battery case if it has one.
4. Your sketch starts running as soon as the board is powered up, and the LED blinks, until you turn off power — **no computer needed!** That's a great way to build small, **autonomous** systems around an Arduino.

The Arduino **converts** the 9V from the battery down to 5V using a regulator on the board. You can also connect anything from **7–12** volts DC to the barrel plug socket (2.1 mm / 5.5 mm diameter, center positive), or stick cables directly into the **Vin** and GND (Ground) pins to power the board from 7–12 volts — great if you don't have a barrel plug on your power source.

Don't attach a 5V power source directly to the **+5V pin** though — it's a voltage **output** pin only, and you may fry your onboard regulator. Use the USB connector instead.



IV. CONNECT A BIG LED

1. **Always disconnect or turn off your power source before you change your circuit** to avoid shorts. They may shut down your USB port, or worse.

2. Bend and stick the **longer** lead (+) of any **red, yellow or green LED** into Digital Pin 13 on the Arduino. Stick its **shorter** lead (–) into the GND pin next to pin 13, as shown.

3. **Connect USB** – now your **big LED blinks** too.

The “Blink” sketch outputs a **high signal (5V)** on pin 13 using `digitalWrite(led,HIGH);` waits for 1000 ms (1 s) using `delay(1000);` then outputs a **low signal (0V)** and waits another second. This makes your LED blink. The yellow onboard LED is also connected to pin 13, so it blinks along.

Every Arduino sketch has one **setup()** method that runs once whenever the Arduino powers up or resets, and a **loop()** function that is **repeated** after that until the board is powered off or reset again. No OS, no multiple apps!

V. ADD A RESISTOR

Connecting an LED directly to 5V and GND will usually **fry** it because of too much current flowing through it. It survived only because the Arduino can't provide more than **40 mA** (milliamps) of current on each pin.

That's still more than the **20 mA** standard LEDs like and need, however. LEDs also **drop** (consume) around **2V** of "forward" voltage (Vf). For precise values, google, e.g., "SparkFun red 5mm LED" ([SparkFun](#) sells great basic components and documents them well). To limit the current, add a **resistor** before or after the LED.

What's the right resistor value? The Arduino pins provide 5V. 2V are dropped by the LED. That leaves 3V to drop over the resistor, at a current of 20 mA. Ohm's law (I use the picture on the right to remember it) says

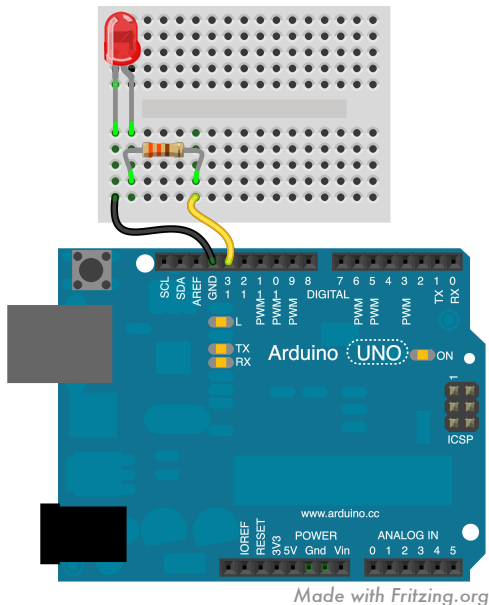
U (voltage) = R (resistance) x I (current), or

$R = U / I = 3 \text{ V} / 20 \text{ mA} = 3 \text{ V} / 0.02 \text{ A} = \mathbf{150 \Omega}$.



A mnemonic diagram for Ohm's Law. It consists of a large blue letter 'U' positioned above a horizontal line. Below the line are the letters 'R' and 'I' in blue, with a small space between them. This visualizes the equation U = R * I.

Choose the **next bigger resistor** you have; in our case it's **330 Ω** (Orange–Orange–Brown–Gold). Use the **color code table** in the Make Pack booklet, google "resistor color codes", or get Adafruit's simple [Circuit Playground](#) app or the comprehensive [Electronic Toolbox Pro](#) app for your iPhone/iPad.



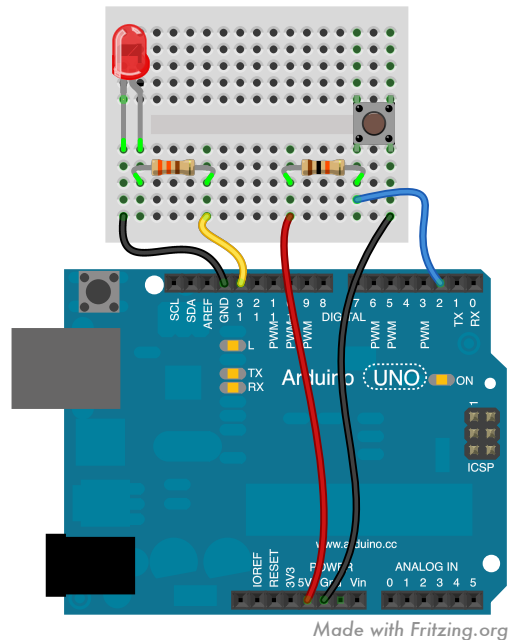
Disconnect USB. In the **mini solderless breadboard**, each vertical **column** of 5 holes is **connected** inside the board. Stick the LED, 330 Ω resistor and jumper wires in as shown. Current will now flow from Arduino **pin 13** through the **resistor** and the **LED** to **GND** when pin 13 is HIGH.

Connect USB. Your LED will glow **slightly less bright** than before, but will last forever. The current is now around $3\text{ V} / 330\ \Omega = \mathbf{9\text{ mA}}$. Current is the same everywhere in a simple closed circuit without branches. So it doesn't matter if you put the resistor before or after the LED.

Tip: Always use red wires for connections to 5V, **black wires** for connections to GND, and other colors using a schema you like. I use **yellow wires** for outputs to LEDs, **green wires** for outputs to motors and servos, and **blue wires** for sensor inputs. It'll help avoid confusion, short-circuits, and fried components. Trust me; I've been there.

VI. DIGITAL INPUT: READ A BUTTON

Disconnect USB. Add a **pushbutton**, **10 kΩ resistor** (Brown—Black—Orange—Gold) and **wires** as shown. Orient the button so the pins that are **closer** are **next** to each other. These connect when you push it; the pins below each other are always connected.



Change the **Blink** code so it only blinks the LED while pin 2 is LOW: Define a global integer (int) variable **pushbutton**. Set it to 2 in your **setup()**. In your **loop()** code, use **if (digitalRead(button)==LOW) {...}**. Don't forget the curly braces and the double equal sign. Now, the LED will only blink while you press the pushbutton!

We are using pin 2 as a **digital input** to detect if its voltage is closer to GND or 5V. Every digital pin 0..13 can be an input or output pin. While input is the default, it's good style to add **pinMode(pushbutton,INPUT);** to your **setup()** function to make it more readable. Remember to end each statement with a **semicolon**.

Tip: For **help** with any function, click on it, then select the **Help:Find In Reference** menu. I also use the language reference at arduino.cc/en/Reference a lot; more tutorials are at arduino.cc/en/Tutorial/Foundations.

The 10 k Ω resistor is a **pullup resistor**. It provides a defined voltage (5V) to pin 2 when the button switch is open (it “pulls it up to 5V”). Otherwise pin 2 would be connected to nothing, or “**floating**”, and pick up random electromagnetic noise like an antenna, leading to unpredictable HIGH/LOW values when you read it.

When you push the button, it pulls pin 2 low (connects it to GND = 0V), and a small current flows through the resistor and switch to GND. All 5V then “drop” across the resistor. Arduino inputs themselves just “measure” the voltage on their pins while consuming hardly any current.

VII. INTERNAL PULLUP RESISTORS

Remove the 10 k Ω pullup resistor from the board. Now your LED may be blinking or not, since pin 2 is floating. Change `setup()` to say **`pinMode(button, INPUT_PULLUP);`** and upload. This connects an **internal pullup resistor** to that pin inside the ATmega chip. It works like your external pullup resistor, but you can simplify your circuit.