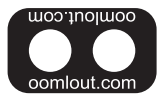
ARDX
experimentation kit for arduino

Experimenter's Guide for Arduino

(ARDX)



A Few Words

About this Kit

The overall goal of this kit is fun. Beyond this, the aim is to get you comfortable using a wide range of electronic components through small, simple and easy circuits. The focus is to get each circuit working then giving you the tools to figure out why. If you encounter any problems, want to ask a question, or would like to know more about any part, extra help is only an e-mail away **help@oomlout.com**.

About Open Source Hardware

All of the projects at SparkFun and .:oomlout:. are open source. What does this mean? It means everything involved in making this kit, be it this guide, 3D models, or code is available for free download. But it goes further, you're also free to reproduce and modify any of this material, then distribute it for yourself. The catch? Quite simple, it is released under a Creative Commons (By - Share Alike) license. This means you must credit .:oomlout:. in your design and share your developments in a similar manner. Why? We grew up learning and playing with open source software and the experience was good fun, we think it would be lovely if a similar experience was possible with physical things.

More details on the Creative Commons CC (By - Share Alike) License can be found at
http://ardx.org/CCLI

About .: oomlout :.

We're a plucky little design company focusing on producing
"delightfully fun open source products"

To check out what we are up to

http://www.oomlout.com

About Seeed Studio

Seeed Studio is an open hardware facilitator based in China, who help designers turn their ideas into products.

http://www.seeedstudio.com/

About Problems

We strive to deliver the highest level of quality in each and every thing we produce. If you ever find an ambiguous instruction, a missing piece, or would just like to ask a question, we'll try our best to help out.

help@oomlout.com

(we like hearing about problems it helps us improve future versions)

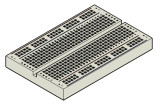
Thanks For Choosing .:oomlout:. and Seeed Studio

.: Where to Find Everything :.

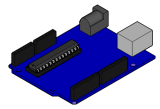
TBCN
table of contents

.: Putting It Together :.

Arduino Holder
x1

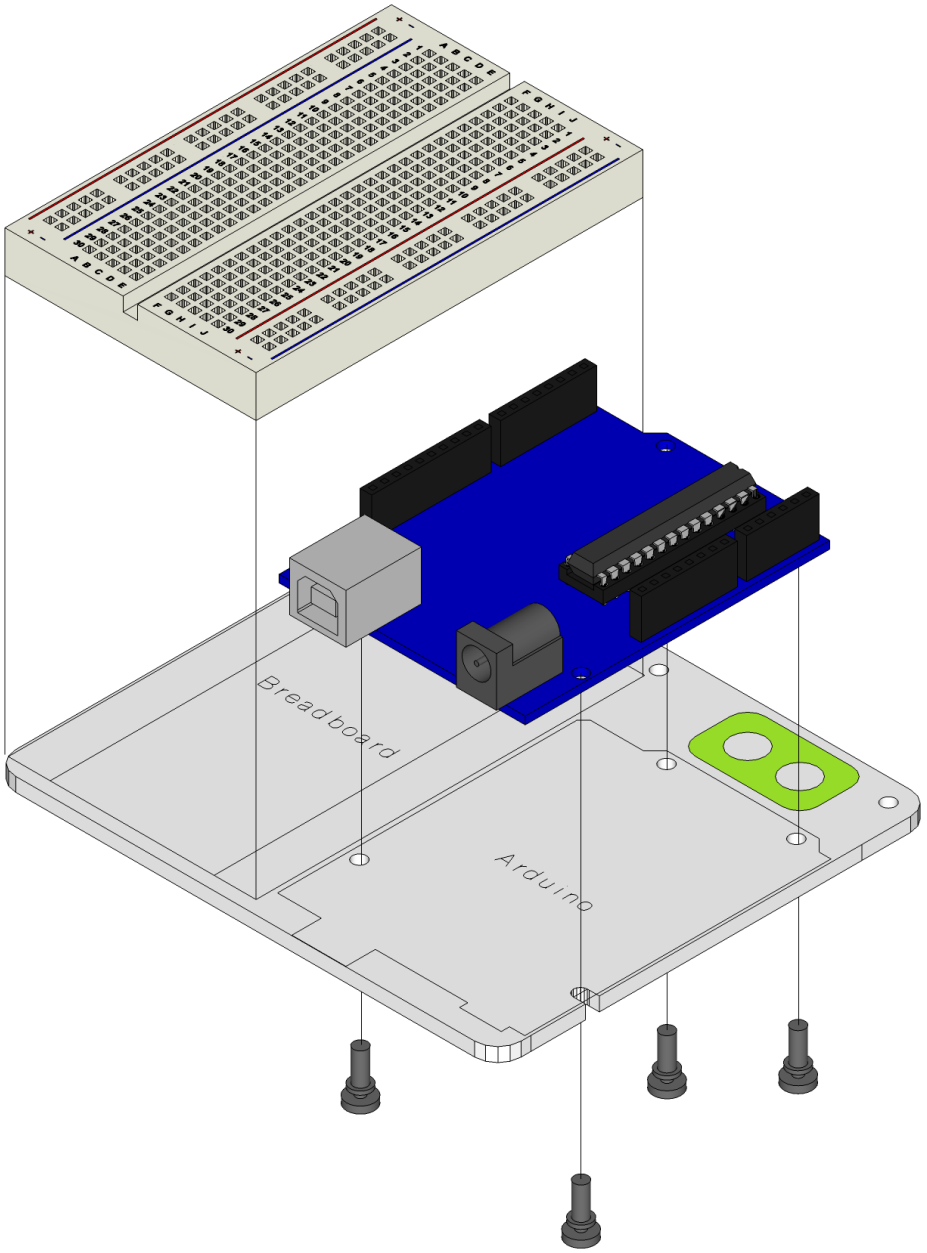


Breadboard
x1



Arduino
x1

3mm Plastic Rivet
x4



.: For an introduction to what an Arduino is, visit :.
.: http://ardx.org/INTR :.

.: Installing the IDE :.

This is the program used to write code for the Arduino. It may seem a little daunting at first but once you have it installed and start playing around, its secrets will reveal themselves.

Step 1: Download the software

Go to
http://arduino.cc/en/Main/Software
download the software for your operating system

Windows® XP

Step 2: Unzip the Software

Unzip
arduino-1.✕-windows.zip (✕- version #)
Recommended Path
c:\Program Files\

Step 3: Shortcut Icon

Open
c:\program files\arduino-1.✕\ (✕- version #)
Right Click
Arduino.exe (send to>Desktop (create shortcut))

Step 4: Plug In Your Arduino

Plug your Arduino in:
Using the included USB cable, plug your Arduino board into a free USB port.
Wait for a box to pop up

Step 5: Add new Hardware

Skip searching the internet
(click the next box when prompted to do so)
Install from a specific location
(click "Install from a list or specific location (Advanced)")
Choose the Location
c:\program files\arduino-00✕✕\drivers\
Finished

Windows Vista®
Windows 7

Step 5: Add new Hardware

Run Device Manager
Start > Run > devmgmt.msc
Choose the Arduino
Other Devices > Arduino Uno (Uno)
Update Driver
click "Update Driver"
Select Driver
click "Browse My Computer for Driver Software"
c:\program files\arduino-1.✕\drivers\
Finished

Mac OS X®

Step 2: Open The .dmg

Open (mount)
arduino-1.✕-macosx.zip (✕- version #)

Step 3: Copy The Application

Go to
"Arduino" (in the devices section of finder)
Move
"Arduino" Application to the "Applications" folder

Step 4: Plug In Your Arduino

Plug your Arduino in:
Using the included USB cable, plug your Arduino board into a free USB port.
Finished

.: NOTE: :.
.: Encountering problems? :.
.: Would like more details? Using Linux? :.
.: http://ardx.org/LINU :.

.:A Small Programming Primer:.

Arduino Programming in Brief

The Arduino is programmed in the C language. This is a quick little primer targeted at people who have a little bit of programing experience and just need a briefing on the idiosyncracies of C and the Arduino IDE. If you find the concepts a bit daunting, don't worry, you can start going through the circuits and pick up most of it along the way. For a more in-depth intro, the Arduino.cc website is a great resource.

Structure

Each Arduino program (often called a sketch) has two required functions (also called routines).

`void setup(){ }`
All the code between the two curly brackets will be run once when your Arduino program first runs.

`void loop(){ }`
This function is run after setup has finished. After it has run once it will be run again, and again, until power is removed.

Syntax

One of the slightly frustrating elements of C is its formatting requirements (this also makes it very powerful). If you remember the following you should be alright.

// (single line comment)
It is often useful to write notes to yourself as you go along about what each line of code does. To do this type two forward slashes and everything until the end of the line will be ignored by your program.

/* */ (multi line comment)
If you have a lot to say you can span several lines as a comment. Everything between these two symbols will be ignored in your program.

{ } (curly brackets)
Used to define when a block of code starts and ends (used in functions as well as loops).

; (semicolon)
Each line of code must be ended with a semicolon (a missing semicolon is often the reason for a program refusing to compile).

Variables

A program is nothing more than instructions to move numbers around in an intelligent way. Variables are used to do the moving.

int (integer)
The main workhorse, stores a number in 2 bytes (16 bits). Has no decimal places and will store a value between -32,768 and 32,767.

long (long)
Used when an integer is not large enough. Takes 4 bytes (32 bits) of RAM and has a range between -2,147,483,648 and 2,147,483,647.

boolean (boolean)
A simple True or False variable. Useful because it only uses one bit of RAM.

float (float)
Used for floating point math (decimals). Takes 4 bytes (32 bits) of RAM and has a range between -3.4028235E+38 and 3.4028235E+38.

char (character)
Stores one character using the ASCII code (ie 'A' = 65). Uses one byte (8 bits) of RAM. The Arduino handles strings as an array of chars.

.:For a full programming reference visit:.
http://ardx.org/PROG

MATHS OPERATORS

Operators used for manipulating numbers. (they work like simple maths).

- **=** (assignment) makes something equal to something else (eg. x = 10 * 2 (x now equals 20))
- **%** (modulo) gives the remainder when one number is divided by another (ex. 12 % 10 (gives 2))
- **+** (addition)
- **-** (subtraction)
- ***** (multiplication)
- **/** (division)

COMPARISON OPERATORS

Operators used for logical comparison.

- **==** (equal to) (eg. 12 == 10 is FALSE or 12 == 12 is TRUE)
- **!=** (not equal to) (eg. 12 != 10 is TRUE or 12 != 12 is FALSE)
- **<** (less than) (eg. 12 < 10 is FALSE or 12 < 12 is FALSE or 12 < 14 is TRUE)
- **>** (greater than) (eg. 12 > 10 is TRUE or 12 > 12 is FALSE or 12 > 14 is FALSE)

CONTROL STRUCTURE

Programs are reliant on controlling what runs next, here are the basic control elements (there are many more online).

```
if(condition){ }
else if( condition ){ }
else { }
```

This will execute the code between the curly brackets if the condition is true, and if not it will test the `else if` condition if that is also false the `else` code will execute.

```
for(int i = 0; i <
#repeats; i++){   }
```

Used when you would like to repeat a chunk of code a number of times (can count up i++ or down i-- or use any variable)

DIGITAL

```
pinMode(pin, mode);
```

Used to set a pin's mode, `pin` is the pin number you would like to address 0-19 (analog 0-5 are 14-19). The mode can either be `INPUT` or `OUTPUT`.

```
digitalWrite(pin, value);
```

Once a pin is set as an `OUTPUT`, it can be set either `HIGH` (pulled to +5 volts) or `LOW` (pulled to ground).

```
int digitalRead(pin);
```

Once a pin is set as an INPUT you can use this to return whether it is HIGH (pulled to +5 volts) or LOW (pulled to ground).

ANALOG

The Arduino is a digital machine but it has the ability to operate in the analog realm (through tricks). Here's how to deal with things that aren't digital.

```
int analogWrite(pin,
                value);
```

Some of the Arduino's pins support pulse width modulation (3, 5, 6, 9, 10, 11). This turns the pin on and off very quickly making it act like an analog output. The value is any number between 0 (0% duty cycle ~0v) and 255 (100% duty cycle ~5 volts).

```
int analogRead(pin);
```

When the analog input pins are set to input you can read their voltage. A value between 0 (for 0 volts) and 1024 (for 5 volts) will be returned.

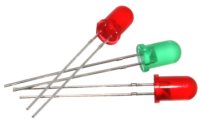
.:A Small Electronics Primer:.

Electronics in Brief

No previous electronic experience is required to have fun with this kit. Here are a few details about each component to make identifying, and perhaps understanding them, a bit easier. If at any point you are worried about how a component is used or why it's not working the internet offers a treasure trove of advice, or we can be contacted at **help@oomlout.com**

Component Details

LED
(Light Emitting Diode)



What it Does:
Emits light when a small current is passed through it. (only in one direction)
Identifying:
Looks like a mini light bulb.

No. of Leads:
2 (one longer, this one connects to positive)
Things to watch out for:
- Will only work in one direction
- Requires a current limiting resistor

More Details:
http://ardx.org/LED

Diode

What it Does:
The electronic equivalent of a one way valve. Allowing current to flow in one direction but not the other.
Identifying:
Usually a cylinder with wires extending from either end. (and an off center line indicating polarity)

No. of Leads:
2
Things to watch out for:
- Will only work in one direction (current will flow if end with the line is connected to ground)

More Details:
http://ardx.org/DIOD

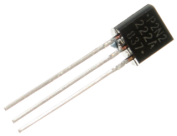
Resistors

What it Does:
Restricts the amount of current that can flow through a circuit.
Identifying:
Cylinder with wires extending from either end. The value is displayed using a color coding system (for details see next page)

No. of Leads:
2
Things to watch out for:
- Easy to grab the wrong value (double check the colors before using)

More Details:
http://ardx.org/RESI

Transistor



What it Does:
Uses a small current to switch or amplify a much larger current.
Identifying:
Comes in many different packages but you can read the part number off the package. (P2N2222AG in this kit and find a datasheet online)

No. of Leads:
3 (Base, Collector, Emitter)
Things to watch out for:
- Plugging in the right way round (also a current limiting resistor is often needed on the base pin)

More Details:
http://ardx.org/TRAN

Hobby Servo

What it Does:
Takes a timed pulse and converts it into an angular position of the output shaft.
Identifying:
A plastic box with 3 wires coming out one side and a shaft with a plastic horn out the top.

No. of Leads:
3
Things to watch out for:
- The plug is not polarized so make sure it is plugged in the right way.

More Details:
http://ardx.org/SERV

DC Motor

What it Does:
Spins when a current is passed through it.
Identifying:
This one is easy, it looks like a motor. Usually a cylinder with a shaft coming out of one end.

No. of Leads:
2
Things to watch out for:
- Using a transistor or relay that is rated for the size of motor you're using.

More Details:
http://ardx.org/MOTO

Component Details (cont.)

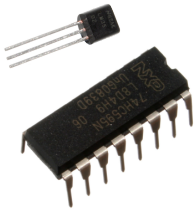
Piezo Element

What it Does:
A pulse of current will cause it to click. A stream of pulses will cause it to emit a tone.
Identifying:
In this kit it comes in a little black barrel, but sometimes they are just a gold disc.

No. of Leads:
2
Things to watch out for:
- Difficult to misuse.

More Details:
http://ardx.org/PIEZ

IC (Integrated Circuit)



What it Does:
Packages any range of complicated electronics inside an easy to use package.
Identifying:
The part ID is written on the outside of the package. (this sometimes requires a lot of light or a magnifying glass to read).

No. of Leads:
2 - 100s (in this kit there is one with 3 (TMP36) and one with 16 (74HC595)
Things to watch out for:
- Proper orientation. (look for marks showing pin 1)

More Details:
http://ardx.org/ICIC

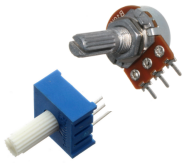
Pushbutton

What it Does:
Completes a circuit when it is pressed.
Identifying:
A little square with leads out the bottom and a button on the top.

No. of Leads:
4
Things to watch out for:
- these are almost square so can be inserted 90 degrees off angle.

More Details:
http://ardx.org/BUTT

Potentiometer

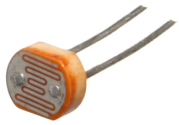


What it Does:
Produces a variable resistance dependent on the angular position of the shaft.
Identifying:
They can be packaged in many different form factors, look for a dial to identify.

No. of Leads:
3
Things to watch out for:
- Accidentally buying logarithmic scale.

More Details:
http://ardx.org/POTE

Photo Resistor



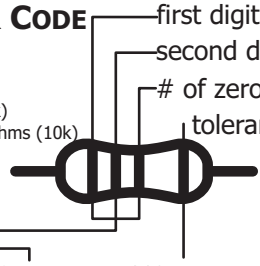
What it Does:
Produces a variable resistance dependent on the amount of incident light.
Identifying:
Usually a little disk with a clear top and a curvy line underneath.

No. of Leads:
2
Things to watch out for:
- Remember it needs to be in a voltage divider before it provides a useful input.

More Details:
http://ardx.org/PHOT

Resistor Color Code

Examples:
green-blue-brown - 560 ohms
red-red-red - 2 200 ohms (2.2k)
brown-black-orange - 10 000 ohms (10k)



0 - Black	5 - Green	20% - none
1 - Brown	6 - Blue	10% - silver
2 - Red	7 - Purple	5% - gold
3 - Orange	8 - Grey	
4 - Yellow	9 - White	

Lead Clipping

Some components in this kit come with very long wire leads. To make them more compatible with a breadboard a couple of changes are required.

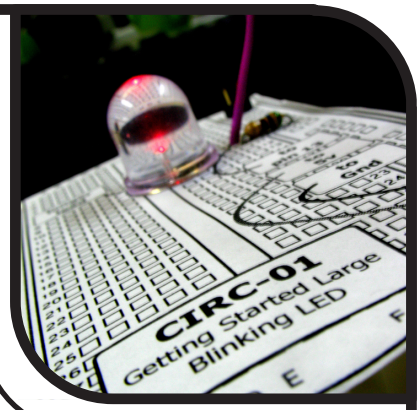
LEDs:
Clip the leads so the long lead is ~10mm (3/8") long and the short one is ~7mm (9/32").

Resistors:
Bend the leads down so they are 90 degrees to the cylinder. Then snip them so they are ~6mm (1/4") long.

Other Components:
Other components may need clipping. Use your discretion when doing so.

CIRC-01

.:Getting Started:.
.:(Blinking LED):.



What We're Doing:

LEDs (light emitting diodes) are used in all sorts of clever things which is why we have included them in this kit. We will start off with something very simple, turning one on and off, repeatedly, producing a pleasant blinking effect. To get started, grab the parts listed below, pin the layout sheet to your breadboard and then plug everything in. Once the circuit is assembled you'll need to upload the program. To do this plug the Arduino board into your USB port. Then select the proper port in **Tools > Serial Port > (the comm port of your Arduino)**. Next upload the program by going to **File > Upload to I/O Board (ctrl+U)**. Finally, bask in the glory and possibility that controlling lights offers.

If you are having trouble uploading, a full troubleshooting guide can be found here: **http://ardx.org/TRBL**

The Circuit:

Parts:


CIRC-01 Breadboard Sheet x1

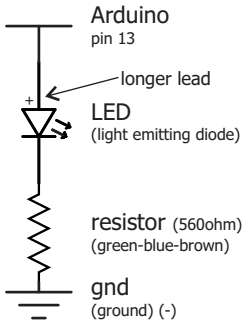

2 Pin Header x4


10mm LED x1


Wire

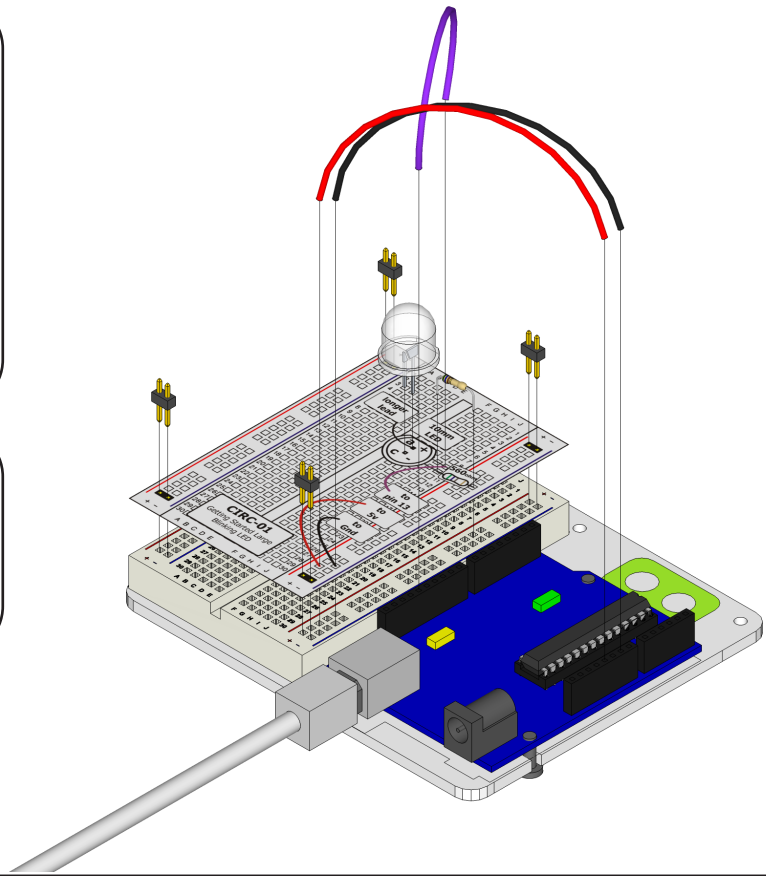

560 Ohm Resistor Green-Blue-Brown x1

Schematic



The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS01
.:view:.
assembly video
http://ardx.org/VIDE01



CODE (no need to type everything in, just click)

File > Examples > 1.Basic > Blink

(example from the great arduino.cc site, check it out for other ideas)

```
/* Blink
 * Turns on an LED on for one second, then off for one second,
 * repeatedly.
 * Created 1 June 2005 By David Cuartielles
 * http://arduino.cc/en/Tutorial/Blink
 * based on an orginal by H. Barragan for the Wiring i/o board
 */

int ledPin =  13;    // LED connected to digital pin 13


// The setup() method runs once, when the sketch starts
void setup()   {                // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);     }

// the loop() method runs over and over again,
// as long as the Arduino has power
void loop()                     {
  digitalWrite(ledPin, HIGH);   // set the LED on
  delay(1000);                  // wait for a second
  digitalWrite(ledPin, LOW);    // set the LED off
  delay(1000);                  // wait for a second
}
```

NOT WORKING? (3 things to try)

LED Not Lighting Up?

LEDs will only work in one direction. Try taking it out and twisting it 180 degrees. (no need to worry, installing it backwards does no permanent harm).

Program Not Uploading

This happens sometimes, the most likely cause is a confused serial port, you can change this in **tools>serial port>**

Still No Success?

A broken circuit is no fun, send us an e-mail and we will get back to you as soon as we can.

help@oomlout.com

MAKING IT BETTER

Changing the pin:

The LED is connected to pin 13 but we can use any of the Arduino's pins. To change it take the wire plugged into pin 13 and move it to a pin of your choice (from 0-13) (you can also use analog 0-5, analog 0 is 14...)

Then in the code change the line:

```
int ledPin = 13; -> int ledPin = newpin;
```

Then upload the sketch: (ctrl-u)

Change the blink time:

Unhappy with one second on one second off?

In the code change the lines:

```
digitalWrite(ledPin, HIGH);
delay(time on);  //(seconds * 1000)
digitalWrite(ledPin, LOW);
delay(time off); //(seconds * 1000)
```

Control the brightness:

Along with digital (on/off) control the Arduino can control some pins in an analog (brightness) fashion. (more details on this in later circuits). To play around with it.

Change the LED to pin 9: (also change the wire)

```
ledPin = 13; -> int ledPin = 9;
```

Replace the code inside the `{ }`'s of `loop()` with this:

```
analogWrite(ledPin, new number);
```

(new number) = any number between 0 and 255.
0 = off, 255 = on, in between = different brightness

Fading:

We will use another included example program. To open go to

File > Examples > 3.Analog > Fading

Then upload to your board and watch as the LED fades in and then out.

MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC01

CIRC-02

.:8 LED Fun:.
.:Multiple LEDs:.

What We're Doing:

We have caused one LED to blink, now it's time to up the stakes. Lets connect eight. We'll also have an opportunity to stretch the Arduino a bit by creating various lighting sequences. This circuit is also a nice setup to experiment with writing your own programs and getting a feel for how the Arduino works.

Along with controlling the LEDs we start looking into a few simple programming methods to keep your programs small.

`for()` loops - used when you want to run a piece of code several times.
`arrays[]` - used to make managing variables easier (it's a group of variables).

The Circuit:

Parts:

CIRC-02 Breadboard Sheet x1

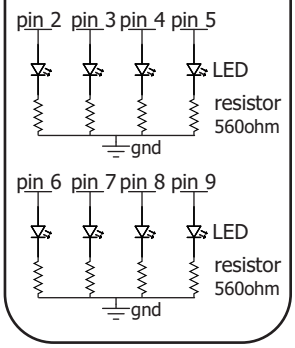
2 Pin Header x4

5mm Green LED x8

Wire

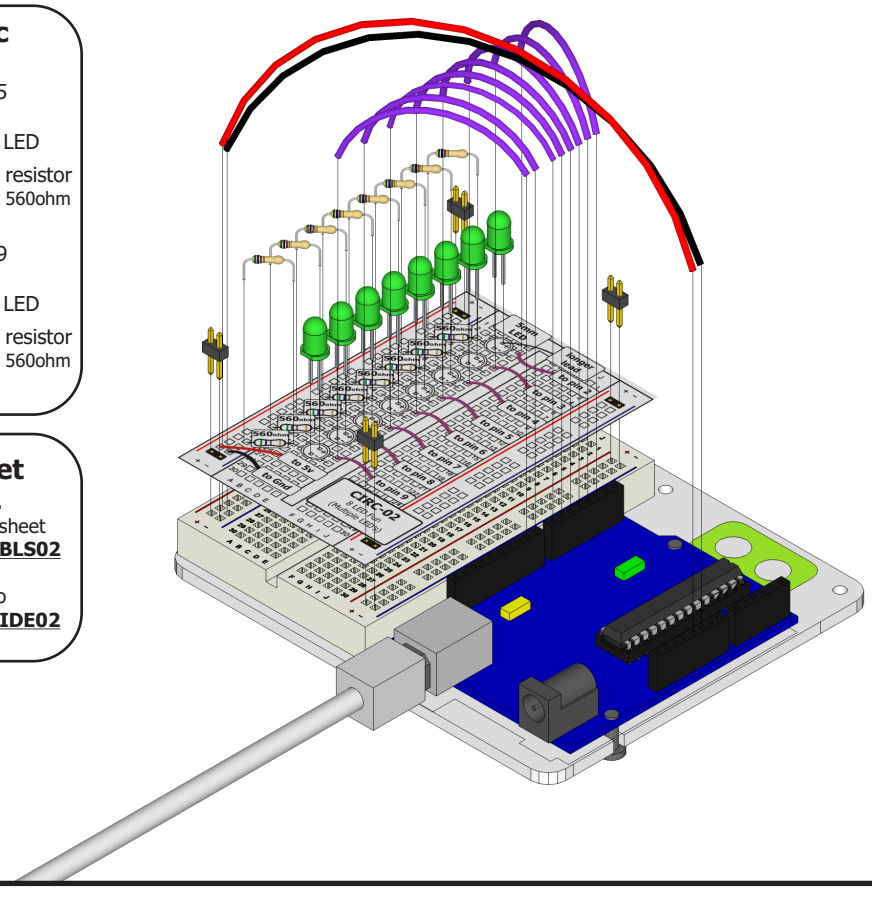
560 Ohm Resistor Green-Blue-Brown x8

Schematic



The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS02
.:view:.
assembly video
http://ardx.org/VIDE02



CIRC-02

.:8 LED Fun:.
.:Multiple LEDs:.

What We're Doing:

We have caused one LED to blink, now it's time to up the stakes. Lets connect eight. We'll also have an opportunity to stretch the Arduino a bit by creating various lighting sequences. This circuit is also a nice setup to experiment with writing your own programs and getting a feel for how the Arduino works.

Along with controlling the LEDs we start looking into a few simple programming methods to keep your programs small.

`for()` loops - used when you want to run a piece of code several times.
`arrays[]` - used to make managing variables easier (it's a group of variables).

The Circuit:

Parts:

CIRC-02 Breadboard Sheet x1

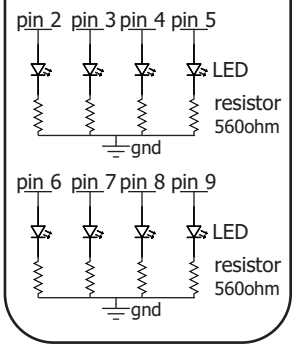
2 Pin Header x4

5mm Green LED x8

Wire

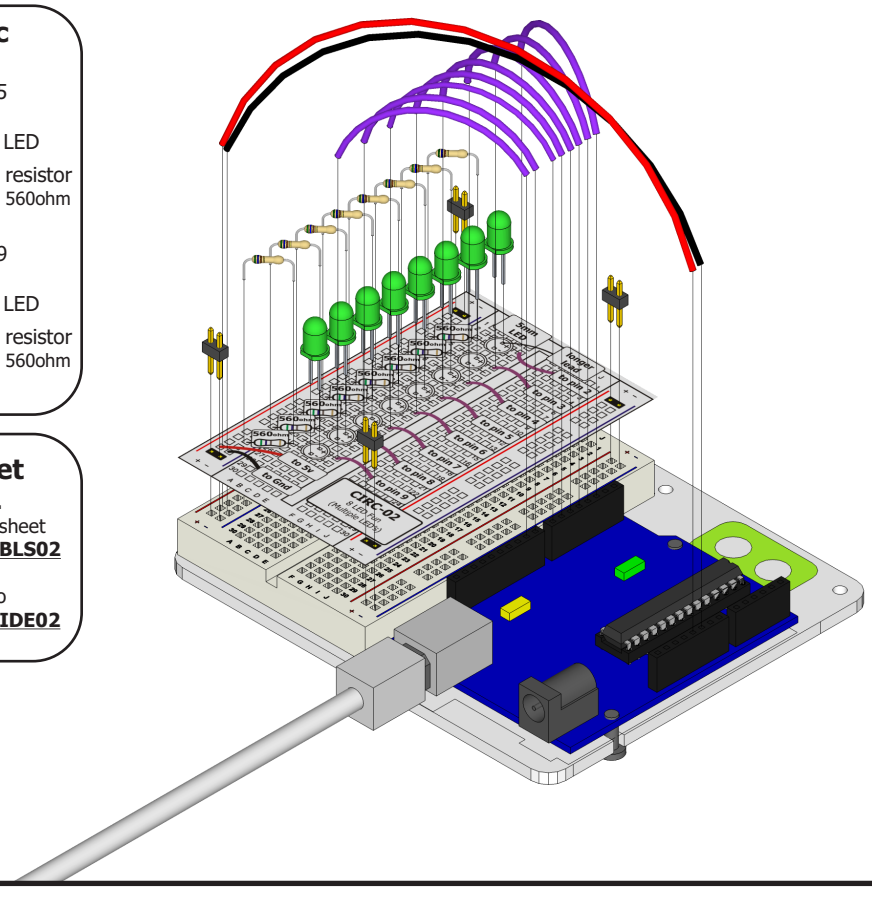
560 Ohm Resistor Green-Blue-Brown x8

Schematic



The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS02
.:view:.
assembly video
http://ardx.org/VIDE02



CIRC-03

.:Spin Motor Spin:.
.:Transistor & Motor:.

What We're Doing:

The Genuino's pins are great for directly controlling small electric items like LEDs. However, when dealing with larger items (like a toy motor or washing machine), an external transistor is required. A transistor is incredibly useful. It switches a lot of current using a much smaller current. A transistor has 3 pins. For a negative type (NPN) transistor, you connect your load to collector and the emitter to ground. Then when a small current flows from base to the emitter, a current will flow through the transistor and your motor will spin (this happens when we set our Genuino pin HIGH). There are literally thousands of different types of transistors, allowing every situation to be perfectly matched. We have chosen a P2N2222AG a rather common general purpose transistor. The important factors in our case are that its maximum voltage (40v) and its maximum current (600 milliamp) are both high enough for our toy motor (full details can be found on its datasheet http://ardx.org/2222).

(The 1N4001 diode is acting as a flyback diode for details on why its there visit: http://ardx.org/4001)

The Circuit:

Parts:


CIRC-03 Breadboard Sheet x1


2 Pin Header x4


Transistor P2N2222AG (TO92) x1

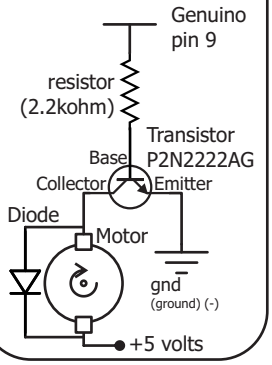

Wire


Toy Motor x1


Diode (1N4001) x1

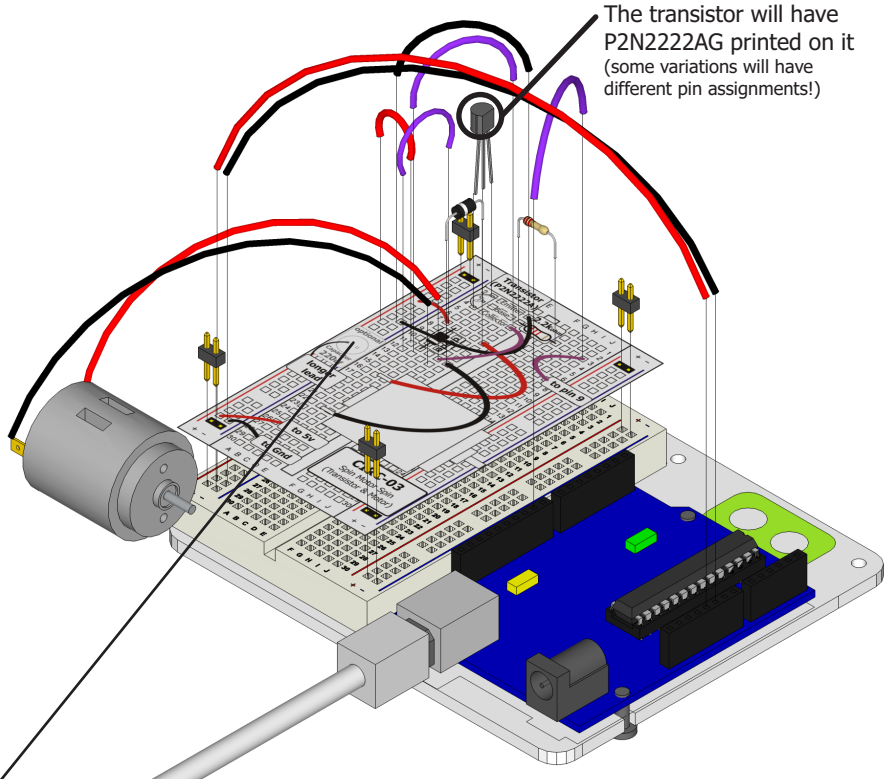

2.2k Ohm Resistor Red-Red-Red x1

Schematic



The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS03
.:view:.
assembly video
http://ardx.org/VIDE03



.:NOTE: if your Genuino is resetting you need to install the optional capacitor:.

CODE (no need to type everything in, just click)

Download the Code from (http://ardx.org/CODE03)

(then simply copy the text and paste it into an empty Genuino Sketch)

```
int motorPin = 9; //pin the motor is connected to

void setup() //runs once
{
 pinMode(motorPin, OUTPUT);
}

void loop()        // run over and over again
{
 motorOnThenOff();
 //motorOnThenOffWithSpeed();
 //motorAcceleration();
}

/*
 * motorOnThenOff() - turns motor on then off
 * (notice this code is identical to the code we
used for
 * the blinking LED)
 */
void motorOnThenOff(){
  int onTime = 2500;  //on time
  int offTime = 1000; //off time
  digitalWrite(motorPin, HIGH);
                  // turns the motor On
  delay(onTime); // waits for onTime milliseconds
  digitalWrite(motorPin, LOW);
                  // turns the motor Off
  delay(offTime);// waits for offTime milliseconds
}

void motorOnThenOffWithSpeed(){
  int onSpeed = 200;// a number between
                    //0 (stopped) and 255 (full speed)
  int onTime = 2500;
  int offSpeed = 50;// a number between
                    //0 (stopped) and 255 (full speed)
  int offTime = 1000;
  analogWrite(motorPin, onSpeed);
      // turns the motor On
  delay(onTime);      // waits for onTime milliseconds
  analogWrite(motorPin, offSpeed);
      // turns the motor Off
  delay(offTime);    // waits for offTime milliseconds
}

void motorAcceleration(){
  int delayTime = 50; //time between each speed step
  for(int i = 0; i < 256; i++){
       //goes through each speed from 0 to 255
   analogWrite(motorPin, i);   //sets the new speed
   delay(delayTime);// waits for delayTime milliseconds
  }
  for(int i = 255; i >= 0; i--){
       //goes through each speed from 255 to 0
    analogWrite(motorPin, i);   //sets the new speed
    delay(delayTime);//waits for delayTime milliseconds
  }
}
```

NOT WORKING? (3 things to try)

Motor Not Spinning?

If you sourced your own transistor, double check with the data sheet that the pinout is compatible with a P2N2222A (many are reversed).

Still No Luck?

If you sourced your own motor, double check that it will work with 5 volts and that it does not draw too much power.

Still Not Working?

Sometimes the Genuino board will disconnect from the computer. Try un-plugging and then re-plugging it into your USB port.

MAKING IT BETTER

Controlling speed:

We played with the Genuino's ability to control the brightness of an LED earlier now we will use the same feature to control the speed of our motor. The Genuino does this using something called Pulse Width Modulation (PWM). This relies on the Genuino's ability to operate really, really fast. Rather than directly controlling the voltage coming from the pin the Genuino will switch the pin on and off very quickly. In the computer world this is going from 0 to 5 volts many times a second, but in the human world we see it as a voltage. For example if the Genuino is PWM'ing at 50% we see the light dimmed 50% because our eyes are not quick enough to see it flashing on and off. The same feature works with transistors. Don't believe me? Try it out.

In the `loop()` section change it to this

```
// motorOnThenOff();
   motorOnThenOffWithSpeed();
// motorAcceleration();
```

Then upload the programme. You can change the speeds by changing the variables onSpeed and offSpeed.

Accelerating and decelerating:

Why stop at two speeds, why not accelerate and decelerate the motor. To do this simply change the loop() code to read

```
// motorOnThenOff();
// motorOnThenOffWithSpeed();
   motorAcceleration();
```

Then upload the program and watch as your motor slowly accelerates up to full speed then slows down again. If you would like to change the speed of acceleration change the variable delayTime (larger means a longer acceleration time).

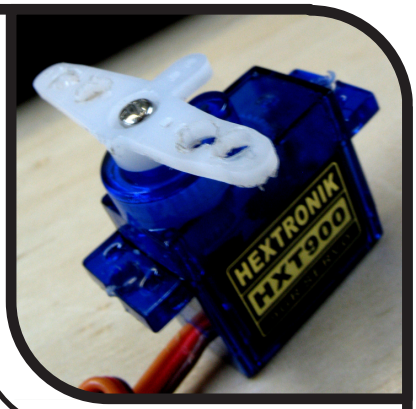
MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC03

CIRC-04

.:A Single Servo:.
.:Servos:.



What We're Doing:

Spinning a motor is good fun but when it comes to projects where motion control is required they tend to leave us wanting more. The answer? Hobby servos. They are mass produced, widely available and cost anything from a couple of dollars to hundreds. Inside is a small gearbox (to make the movement more powerful) and some electronics (to make it easier to control). A standard servo is positionable from 0 to 180 degrees. Positioning is controlled through a timed pulse, between 1.25 milliseconds (0 degrees) and 1.75 milliseconds (180 degrees) (1.5 milliseconds for 90 degrees). Timing varies between manufacturer. If the pulse is sent every 25-50 milliseconds the servo will run smoothly. One of the great features of the Genuino is it has a software library that allows you to control two servos (connected to pin 9 or 10) using a single line of code.

The Circuit:

Parts:

CIRC-04 Breadboard Sheet x1

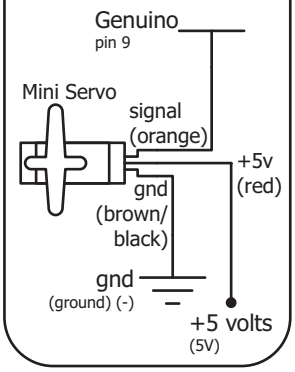
2 Pin Header x4

3 Pin Header x1

Wire

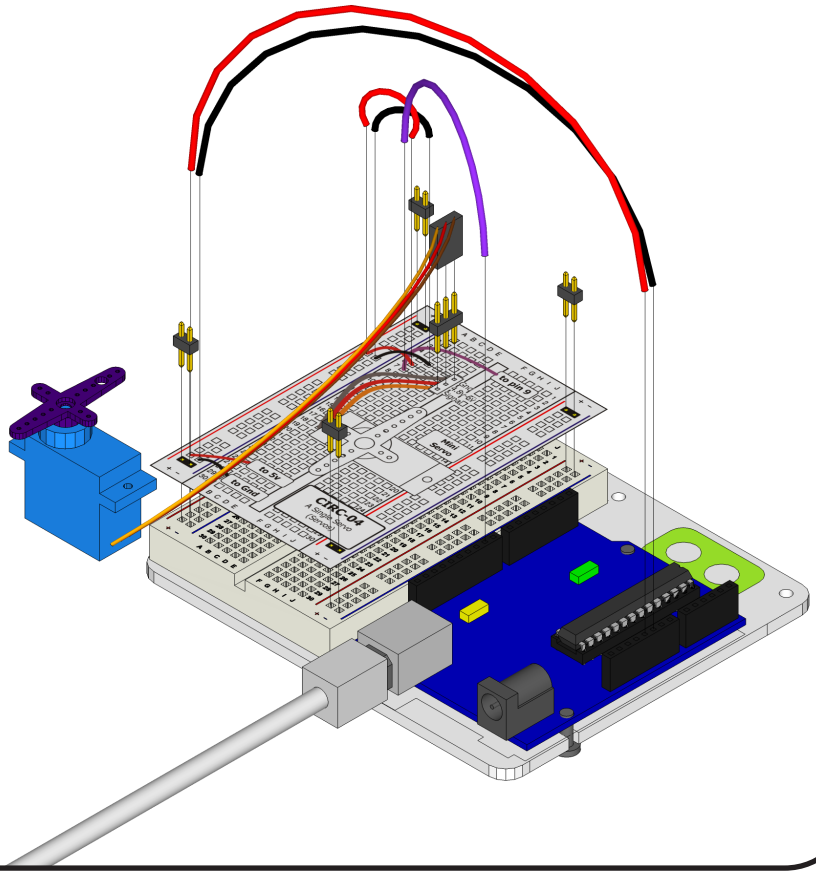
Mini Servo x1

Schematic



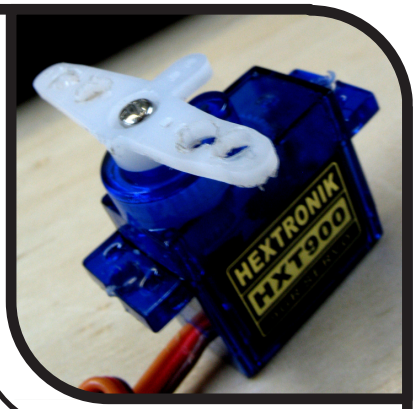
The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS04
.:view:.
assembly video
http://ardx.org/VIDE04



CIRC-04

.:A Single Servo:.
.:Servos:.



What We're Doing:

Spinning a motor is good fun but when it comes to projects where motion control is required they tend to leave us wanting more. The answer? Hobby servos. They are mass produced, widely available and cost anything from a couple of dollars to hundreds. Inside is a small gearbox (to make the movement more powerful) and some electronics (to make it easier to control). A standard servo is positionable from 0 to 180 degrees. Positioning is controlled through a timed pulse, between 1.25 milliseconds (0 degrees) and 1.75 milliseconds (180 degrees) (1.5 milliseconds for 90 degrees). Timing varies between manufacturer. If the pulse is sent every 25-50 milliseconds the servo will run smoothly. One of the great features of the Genuino is it has a software library that allows you to control two servos (connected to pin 9 or 10) using a single line of code.

The Circuit:

Parts:

CIRC-04 Breadboard Sheet x1

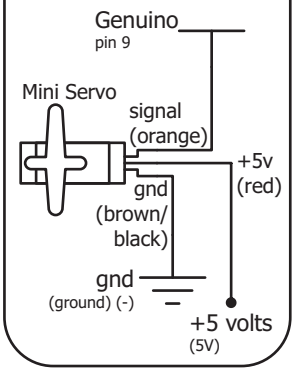
2 Pin Header x4

3 Pin Header x1

Wire

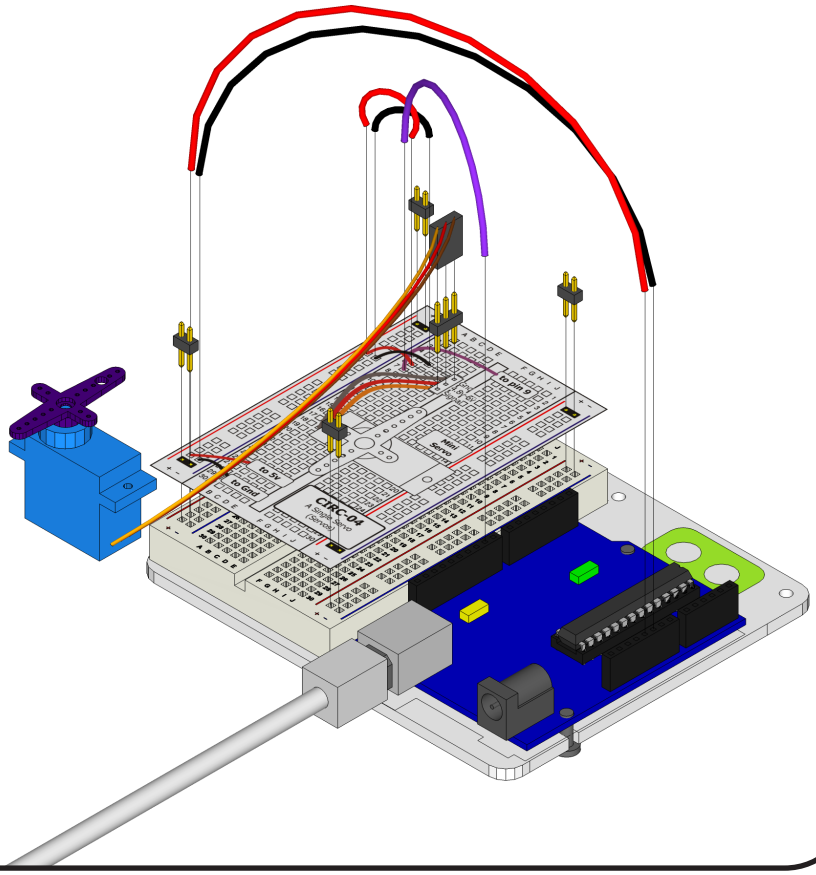
Mini Servo x1

Schematic



The Internet

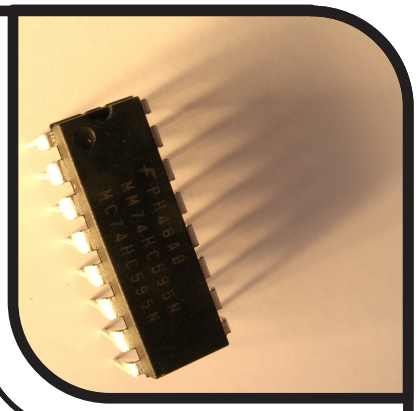
.:download:.
breadboard layout sheet
http://ardx.org/BBLS04
.:view:.
assembly video
http://ardx.org/VIDE04



CIRC-05

.:8 More LEDs:.

.:74HC595 Shift Register:.



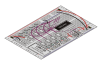
What We're Doing:

Time to start playing with chips, or integrated circuits (ICs) as they like to be called. The external packaging of a chip can be very deceptive. For example, the chip on the Genuino board (a microcontroller) and the one we will use in this circuit (a shift register) look very similar but are in fact rather different. The price of the ATMega chip on the Genuino board is a few dollars while the 74HC595 is a couple dozen cents. It's a good introductory chip, and once you're comfortable playing around with it and its datasheet (available online http://ardx.org/74HC595) the world of chips will be your oyster. The shift register (also called a serial to parallel converter), will give you an additional 8 outputs (to control LEDs and the like) using only three Genuino pins. They can also be linked together to give you a nearly unlimited number of outputs using the same four pins. To use it you "clock in" the data and then lock it in (latch it). To do this you set the data pin to either HIGH or LOW, pulse the clock, then set the data pin again and pulse the clock repeating until you have shifted out 8 bits of data. Then you pulse the latch and the 8 bits are transferred to the shift registers pins. It sounds complicated but is really simple once you get the hang of it.

(for a more in depth look at how a shift register works visit: http://ardx.org/SHIF)

The Circuit:

Parts:



CIRC-05 Breadboard Sheet x1

2 Pin Header x4

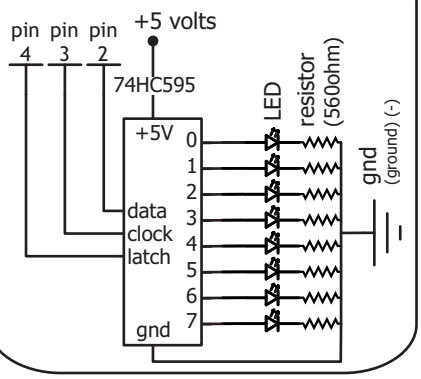
Shift Register 74HC595 x1

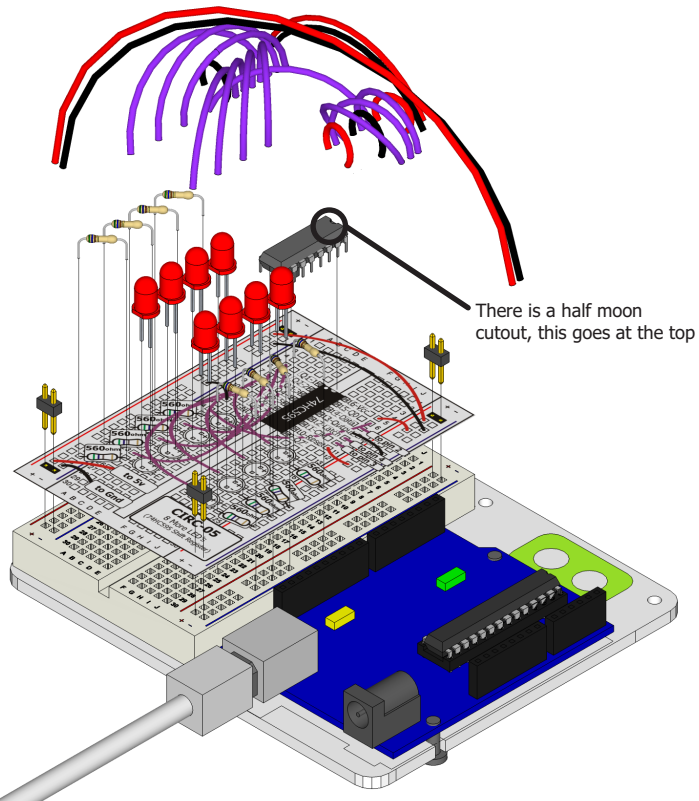
Wire

Red LED x8

560 Ohm Resistor Green-Blue-Brown x8

Schematic





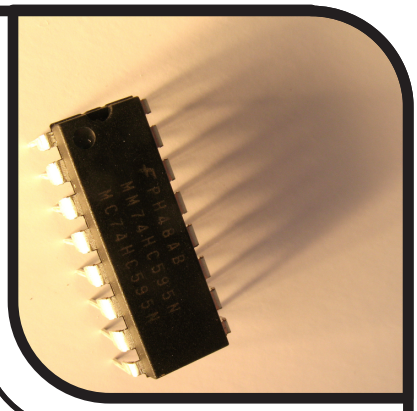
The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS05
.:view:.
assembly video
http://ardx.org/VIDE05

CIRC-05

.:8 More LEDs:.

.:74HC595 Shift Register:.



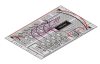
What We're Doing:

Time to start playing with chips, or integrated circuits (ICs) as they like to be called. The external packaging of a chip can be very deceptive. For example, the chip on the Genuino board (a microcontroller) and the one we will use in this circuit (a shift register) look very similar but are in fact rather different. The price of the ATMega chip on the Genuino board is a few dollars while the 74HC595 is a couple dozen cents. It's a good introductory chip, and once you're comfortable playing around with it and its datasheet (available online http://ardx.org/74HC595) the world of chips will be your oyster. The shift register (also called a serial to parallel converter), will give you an additional 8 outputs (to control LEDs and the like) using only three Genuino pins. They can also be linked together to give you a nearly unlimited number of outputs using the same four pins. To use it you "clock in" the data and then lock it in (latch it). To do this you set the data pin to either HIGH or LOW, pulse the clock, then set the data pin again and pulse the clock repeating until you have shifted out 8 bits of data. Then you pulse the latch and the 8 bits are transferred to the shift registers pins. It sounds complicated but is really simple once you get the hang of it.

(for a more in depth look at how a shift register works visit: http://ardx.org/SHIF)

The Circuit:

Parts:



CIRC-05 Breadboard Sheet x1

2 Pin Header x4

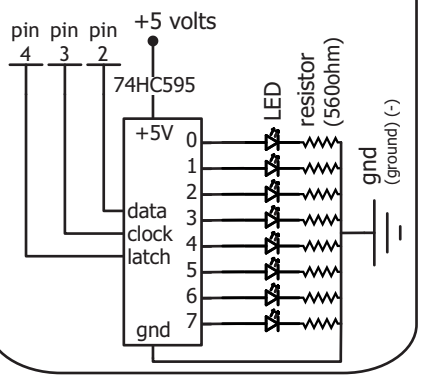
Shift Register 74HC595 x1

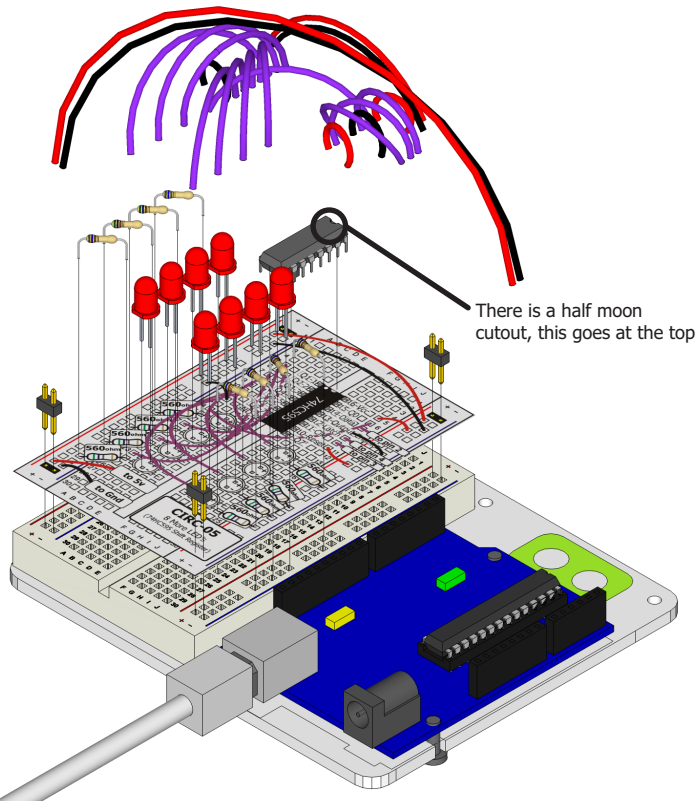
Wire

Red LED x8

560 Ohm Resistor Green-Blue-Brown x8

Schematic



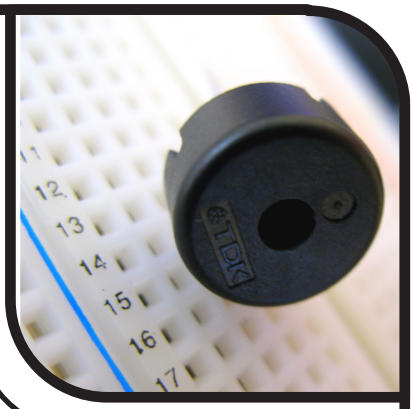


The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS05
.:view:.
assembly video
http://ardx.org/VIDE05

CIRC-06

.:Music:.
.:Piezo Elements:.

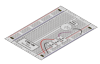


What We're Doing:

To this point we have controlled light, motion, and electrons. Let's tackle sound next. But sound is an analog phenomena, how will our digital Genuino cope? We will once again rely on its incredible speed which will let it mimic analog behavior. To do this, we will attach a piezo element to one of the Genuino's digital pins. A piezo element makes a clicking sound each time it is pulsed with current. If we pulse it at the right frequency (for example 440 times a second to make the note middle A) these clicks will run together to produce notes. Let's get to experimenting with it and get your Genuino playing "Twinkle Twinkle Little Star".

The Circuit:

Parts:



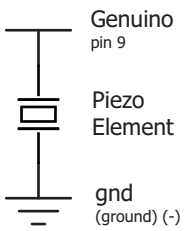
CIRC-06 Breadboard Sheet x1

2 Pin Header x4

Piezo Element x1

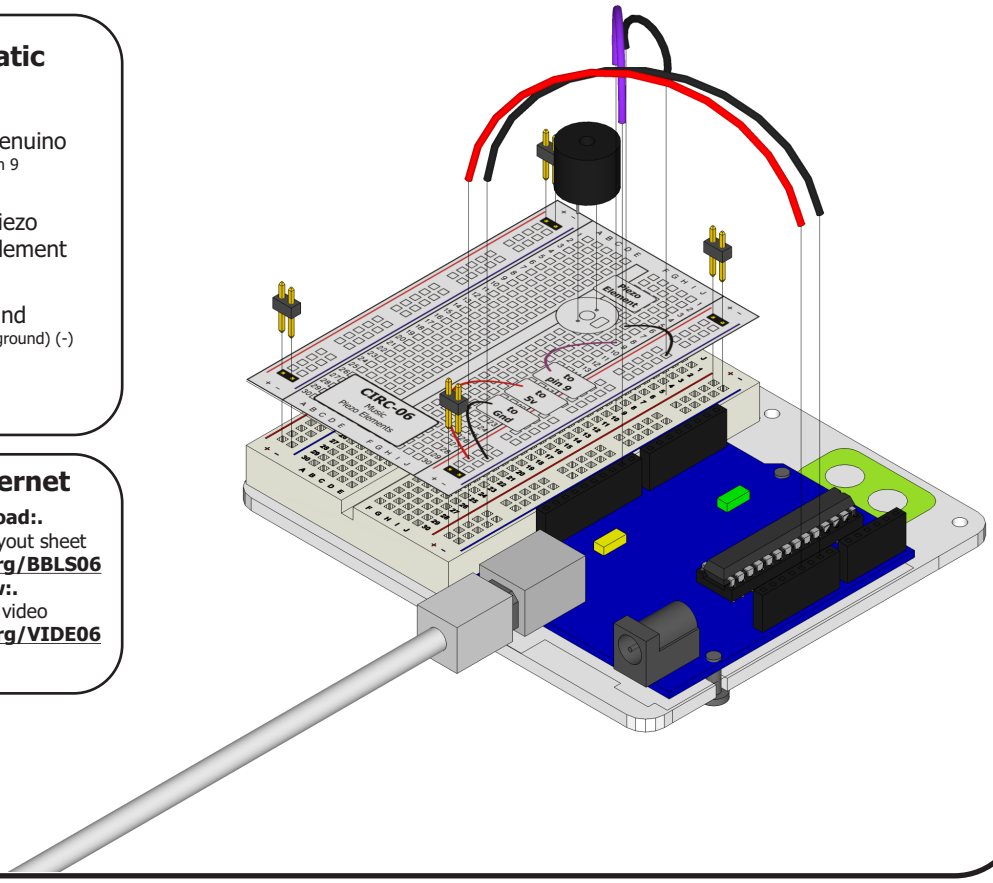
Wire

Schematic



The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS06
.:view:.
assembly video
http://ardx.org/VIDE06



CODE (no need to type everything in, just click)

Download the Code from (http://ardx.org/CODE06)

(copy the text and paste it into an empty Genuino Sketch)

```
/* Melody
 * (cleft) 2005 D. Cuartielles for K3
 *
 * This example uses a piezo speaker to play melodies. It sends
 * a square wave of the appropriate frequency to the piezo,
 * generating the corresponding tone.
 *
 * The calculation of the tones is made following the
 * mathematical operation:
 *
 *       timeHigh = period / 2 = 1 / (2 * toneFrequency)
 *
 * where the different tones are described as in the table:
 *
 * note    frequency  period     timeHigh
 * c               261 Hz               3830          1915
 * d               294 Hz               3400          1700
 * e               329 Hz               3038          1519
 * f               349 Hz               2864          1432
 * g               392 Hz               2550          1275
 * a               440 Hz               2272          1136
 * b               493 Hz               2028          1014
 * C               523 Hz               1912          956
 *
 * http://www.Genuino.cc/en/Tutorial/Melody
 */

int speakerPin = 9;
int length = 15; // the number of notes
char notes[] = "ccggaagffeeddc "; // a space represents a rest
int beats[] = { 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 4 };
int tempo = 300;

void playTone(int tone, int duration) {
  for (long i = 0; i < duration * 1000L; i += tone * 2) {
    digitalWrite(speakerPin, HIGH);
    delayMicroseconds(tone);
    digitalWrite(speakerPin,
LOW);
    delayMicroseconds(tone);
  }
}

void playNote(char note, int duration) {
  char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };
  int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956
};
    // play the tone corresponding to the note name
  for (int i = 0; i < 8; i++) {
    if (names[i] == note) {
      playTone(tones[i], duration);
    }
  }
}

void setup() {
  pinMode(speakerPin, OUTPUT);
}

void loop() {
  for (int i = 0; i < length; i++) {
    if (notes[i] == ' ') {
      delay(beats[i] * tempo); // rest
    } else {
      playNote(notes[i], beats[i] * tempo);
    }
        // pause between notes
    delay(tempo / 2);   }
}
```

NOT WORKING? (3 things to try)

No Sound

Given the size and shape of the piezo element it is easy to miss the right holes on the breadboard. Try double checking its placement.

Can't Think While the Melody is Playing?

Just pull up the piezo element whilst you think, upload your program then plug it back in.

Tired of Twinkle Twinkle Little Star?

The code is written so you can easily add your own songs, check out the code below to get started.

MAKING IT BETTER

Playing with the speed:

The timing for each note is calculated based on variables, as such we can tweak the sound of each note or the timing. To change the speed of the melody you need to change only one line.

`int tempo = 300; ---> int tempo = (new #)`

Change it to a larger number to slow the melody down, or a smaller number to speed it up.

Tuning the notes:

If you are worried about the notes being a little out of tune this can be fixed as well. The notes have been calculated based on a formula in the comment block at the top of the program. But to tune individual notes just adjust their values in the `tones[]` array up or down until they sound right. (each note is matched by its name in the `names[]` (array ie. c = 1915)

```
char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b',
'C' };
int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136,
1014, 956 };
```

Composing your own melodies:

The program is pre-set to play 'Twinkle Twinkle Little Star' however the way it is programmed makes changing the song easy. Each song is defined in one int and two arrays, the `int length` defines the number of notes, the first array `notes[]` defines each note, and the second `beats[]` defines how long each note is played. Some Examples:

Twinkle Twinkle Little Star

```
int length = 15;
char notes[] = {"ccggaagffeeddc "};
int beats[] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
                1, 1, 2, 4 };
```

Happy Birthday (first line)

```
int length = 13;
char notes[] = {"ccdcfeccdcgf "};
int beats[] = {1,1,1,1,1,2,1,1,1,1,1,2,4};
```

MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC06

CIRC-07

.:Button Pressing:.
.:Pushbuttons:.

What We're Doing:

Up to this point we have focused entirely on outputs, time to get our Genuino to listen, watch and feel. We'll start with a simple pushbutton. Wiring up the pushbutton is simple. There is one component, the pull up resistor, that might seem out of place. This is included because an Genuino doesn't sense the same way we do (i.e. button pressed, button unpressed). Instead it looks at the voltage on the pin and decides whether it is HIGH or LOW. The button is set up to pull the Genuino's pin LOW when it is pressed, however, when the button is unpressed the voltage of the pin will float (causing occasional errors). To get the Genuino to reliably read the pin as HIGH when the button is unpressed, we add the pull up resistor.

(note: the first example program uses only one of the two buttons)

The Circuit:

Parts:


CIRC-07 Breadboard Sheet x1


2 Pin Header x4


Pushbutton x2

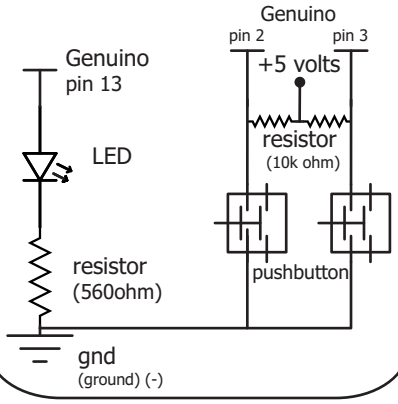

Wire

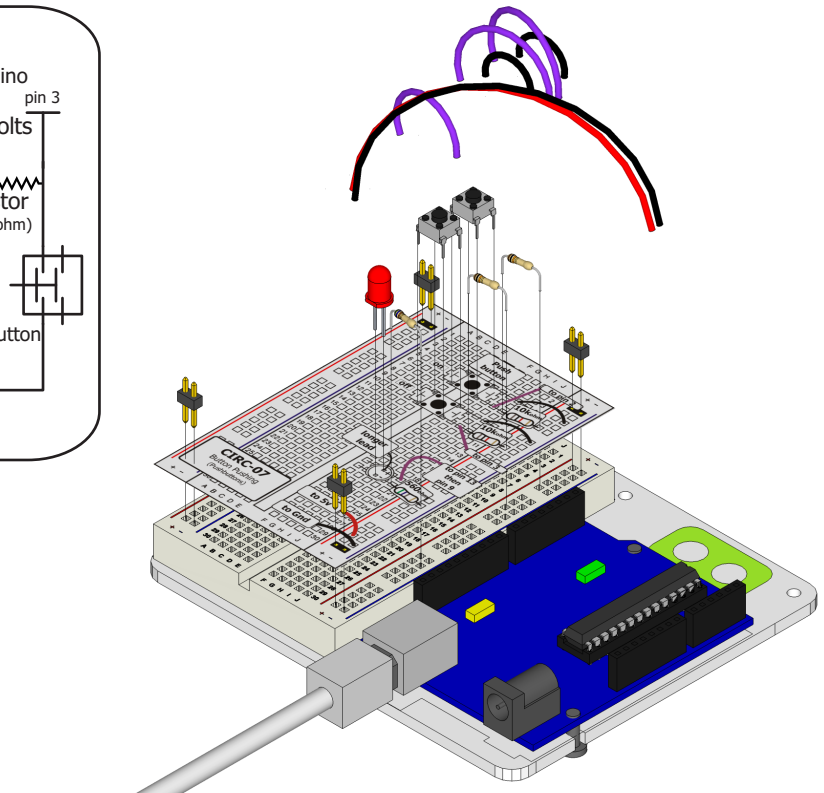

10k Ohm Resistor Brown-Black-Orange x2


560 Ohm Resistor Green-Blue-Brown x1


Red LED x1

Schematic





The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS07
.:view:.
assembly video
http://ardx.org/VIDE07

CODE (no need to type everything in, just click)

File > Examples > 2.Digital > Button

(example from the great Genuino.cc site, check it out for other great ideas)

```
/*
 * Button
 * by DojoDave <http://www.0j0.org>
 *
 * Turns on and off a light emitting diode(LED) connected to digital
 * pin 13, when pressing a pushbutton attached to pin 7.
 * http://www.Genuino.cc/en/Tutorial/Button
 */
 int ledPin = 13;                // choose the pin for the LED
int inputPin = 2;                // choose the input pin (for a pushbutton)
int val = 0;                     // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);       // declare LED as output
  pinMode(inputPin, INPUT);      // declare pushbutton as input
}

void loop(){
  val = digitalRead(inputPin);  // read input value
  if (val == HIGH) {            // check if the input is HIGH
    digitalWrite(ledPin, LOW);  // turn LED OFF
  } else {
    digitalWrite(ledPin, HIGH); // turn LED ON
  }
}
```

NOT WORKING? (3 things to try)

Light Not Turning On

The pushbutton is square and because of this it is easy to put it in the wrong way. Give it a 90 degree twist and see if it starts working.

Light Not Fading

A bit of a silly mistake we constantly made, when you switch from simple on off to fading remember to move the LED wire from pin 13 to pin 9.

Underwhelmed?

No worries these circuits are all super stripped down to make playing with the components easy, but once you throw them together the sky is the limit.

MAKING IT BETTER

On button off button:

The initial example may be a little underwhelming (i.e. I don't really need an Genuino to do this), let's make it a little more complicated. One button will turn the LED on the other will turn the LED off. Change the code to:

```
int ledPin = 13;        // choose the pin for the LED
int inputPin1 = 3;      // button 1
int inputPin2 = 2;      // button 2

void setup() {
  pinMode(ledPin, OUTPUT);   // declare LED as output
  pinMode(inputPin1, INPUT); // make button 1 an input
  pinMode(inputPin2, INPUT); // make button 2 an input
}

void loop(){
  if (digitalRead(inputPin1) == LOW) {
    digitalWrite(ledPin, LOW);  // turn LED OFF
  } else if (digitalRead(inputPin2) == LOW) {
    digitalWrite(ledPin, HIGH); // turn LED ON
  }
}
```

Upload the program to your board, and start toggling the LED on and off.

Fading up and down:

Lets use the buttons to control an analog signal. To do this you will need to change the wire connecting the LED from pin 13 to pin 9, also change this in code.

```
int ledPin = 13; ----> int ledPin = 9;
```

Next change the loop() code to read.

```
int value = 0;
void loop(){
  if (digitalRead(inputPin1) == LOW) { value--; }
  else if (digitalRead(inputPin2) == LOW) { value++; }
  value = constrain(value, 0, 255);
  analogWrite(ledPin, value);
  delay(10);
}
```

Changing fade speed:

If you would like the LED to fade faster or slower, there is only one line of code that needs changing;

```
delay(10);  ----> delay(new #);
```

To fade faster make the number smaller, slower requires a larger number.

MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC07

CIRC-08

.:Twisting:.
.:Potentiometers:.

What We're Doing:

Along with the digital pins, the Genuino also has 6 pins which can be used for analog input. These inputs take a voltage (from 0 to 5 volts) and convert it to a digital number between 0 (0 volts) and 1023 (5 volts) (10 bits of resolution). A very useful device that exploits these inputs is a potentiometer (also called a variable resistor). When it is connected with 5 volts across its outer pins the middle pin will read some value between 0 and 5 volts dependent on the angle to which it is turned (ie. 2.5 volts in the middle). We can then use the returned values as a variable in our program.

The Circuit:

Parts:

CIRC-08 Breadboard Sheet x1

2 Pin Header x4

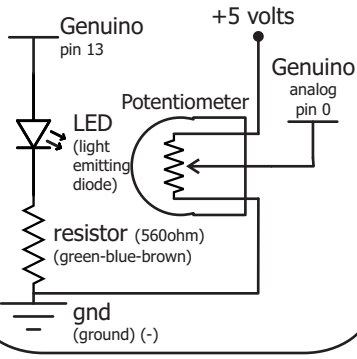
Potentiometer 10k ohm x1

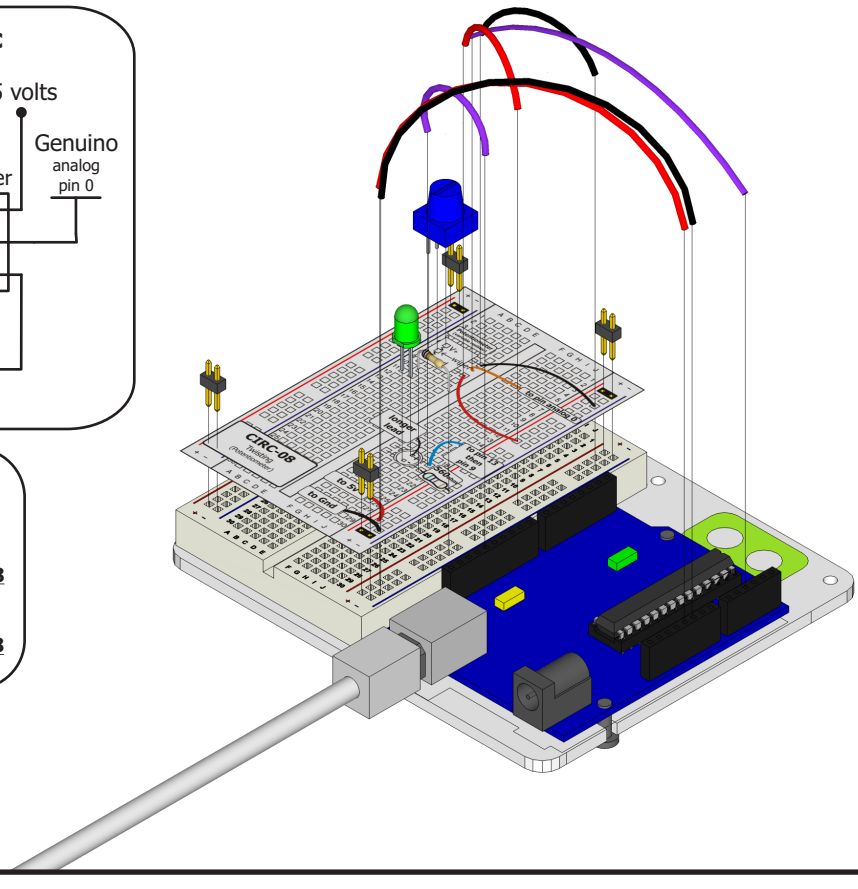
Wire

Green LED x1

560 Ohm Resistor Green-Blue-Brown x1

Schematic





The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS08
.:view:.
assembly video
http://ardx.org/VIDE08

CODE (no need to type everything in, just click)

File > Examples > 3.Analog > AnalogInput

(example from the great Genuino.cc site, check it out for other great ideas)

```
/* Analog Input
 * Demonstrates analog input by reading an analog sensor on analog
 * pin 0 and turning on and off a light emitting diode(LED)  connected to
digital pin 13.
 * The  amount of time the LED will be on and off depends on the value obtained by
 * analogRead().
 * Created by David Cuartielles
 * Modified 16 Jun 2009
 * By Tom Igoe
 * http://Genuino.cc/en/Tutorial/AnalogInput
 */

int sensorPin = 0;    // select the input pin for the potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  pinMode(ledPin, OUTPUT);  //declare the ledPin as an OUTPUT:
}


void loop() {
  sensorValue = analogRead(sensorPin);// read the value from the sensor:
  digitalWrite(ledPin, HIGH);         // turn the ledPin on
  delay(sensorValue);                 // stop the program for <sensorValue> milliseconds:
  digitalWrite(ledPin, LOW);          // turn the ledPin off:
  delay(sensorValue);                 // stop the program for for <sensorValue> milliseconds:
}
```

NOT WORKING? (3 things to try)

Sporadically Working

This is most likely due to a slightly dodgy connection with the potentiometer's pins. This can usually be conquered by taping the potentiometer down.

Not Working

Make sure you haven't accidentally connected the potentiometer's wiper to digital pin 0 rather than analog pin 0. (the row of pins beneath the power pins)

Still Backward

You can try operating the circuit upside down. Sometimes this helps.

MAKING IT BETTER

Threshold switching:

Sometimes you will want to switch an output when a value exceeds a certain threshold. To do this with a potentiometer change the `loop()` code to.

```
void loop() {
  int threshold = 512;
  if(analogRead(sensorPin) > threshold){
digitalWrite(ledPin, HIGH);}
  else{ digitalWrite(ledPin, LOW);}
}
```

This will cause the LED to turn on when the value is above 512 (about halfway), you can adjust the sensitivity by changing the `threshold` value.

Fading:

Let's control the brightness of an LED directly from the potentiometer. To do this we need to first change the pin the LED is connected to. Move the wire from pin 13 to pin 9 and change one line in the code.

```
int ledPin = 13; ----> int ledPin = 9;
```

Then change the loop code to.

```
void loop() {
 int value = analogRead(sensorPin) / 4;
 analogWrite(ledPin, value);
}
```

Upload the code and watch as your LED fades in relation to your potentiometer spinning. (Note: the reason we divide the value by 4 is the analogRead() function returns a value from 0 to 1023 (10 bits), and analogWrite() takes a value from 0 to 255 (8 bits))

Controlling a servo:

This is a really neat example and brings a couple of circuits together. Wire up the servo like you did in CIRC-04, then open the example program Knob (**File > Examples > Servo > Knob**), then change one line of code.

```
int sensorPin = 0; ----> int sensorPin = 2;
```

Upload to your Genuino and then watch as the servo shaft turns as you turn the potentiometer.

MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC08

CIRC-09

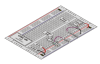
.:Light:.
.:Photo Resistors:.

What We're Doing:

Whilst getting input from a potentiometer can be useful for human controlled experiments, what do we use when we want an environmentally controlled experiment? We use exactly the same principles but instead of a potentiometer (twist based resistance) we use a photo resistor (light based resistance). The Genuino cannot directly sense resistance (it senses voltage) so we set up a voltage divider (http://ardx.org/VODI). The exact voltage at the sensing pin is calculable, but for our purposes (just sensing relative light) we can experiment with the values and see what works for us. A low value will occur when the sensor is well lit while a high value will occur when it is in darkness.

The Circuit:

Parts:


CIRC-09 Breadboard Sheet x1


2 Pin Header x4


Photo-Resistor x1

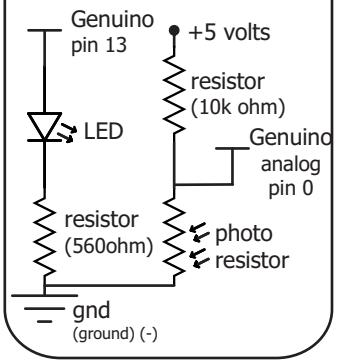

Wire


10k Ohm Resistor Brown-Black-Orange x1


560 Ohm Resistor Green-Blue-Brown x1


Green LED x1

Schematic



The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS09
.:view:.
assembly video
http://ardx.org/VIDE09

CODE (no need to type everything in, just click)

Download the Code from (http://ardx.org/CODE09)

(copy the text and paste it into an empty Genuino Sketch)

```
/*
 * A simple programme that will change the
 * intensity of an LED based  on the amount of
 * light incident on the photo resistor.
 *
 */

//PhotoResistor Pin
int lightPin = 0; //the analog pin the
                  //photoresistor is
                  //connected to
                  //the photoresistor is not
                  //calibrated to any units so
                  //this is simply a raw sensor
                  //value (relative light)
//LED Pin
int ledPin = 9;//the pin the LED is connected to
               //we are controlling brightness so
               //we use one of the PWM (pulse
               //width modulation pins)

void setup()
{
  pinMode(ledPin, OUTPUT); //sets the led pin to
//output
}
 /*
 * loop() - this function will start after setup
 * finishes and then repeat
 */
void loop()
{
 int lightLevel = analogRead(lightPin); //Read the
                                        // lightlevel
 lightLevel = map(lightLevel, 0, 900, 0, 255);
         //adjust the value 0 to 900 to 0 to 255
  lightLevel = constrain(lightLevel, 0, 255);
      //make sure the value is between 0 and 255
  analogWrite(ledPin, lightLevel);  //write the value
}
```

NOT WORKING? (3 things to try)

LED Remains Dark

This is a mistake we continue to make time and time again, if only they could make an LED that worked both ways. Pull it up and give it a twist.

It Isn't Responding to Changes in Light.

Given that the spacing of the wires on the photo-resistor is not standard, it is easy to misplace it. Double check its in the right place.

Still not quite working?

You may be in a room which is either too bright or dark. Try turning the lights on or off to see if this helps. Or if you have a flashlight near by give that a try.

MAKING IT BETTER

Reverse the response:

Perhaps you would like the opposite response. Don't worry we can easily reverse this response just change:

```
analogWrite(ledPin, lightLevel); ---->
analogWrite(ledPin, 255 - lightLevel);
```

Upload and watch the response change:

Night light:

Rather than controlling the brightness of the LED in response to light, let's instead turn it on or off based on a threshold value. Change the loop() code with.

```
  void loop(){
     int threshold = 300;
     if(analogRead(lightPin) > threshold){
       digitalWrite(ledPin, HIGH);
     }else{
       digitalWrite(ledPin, LOW);
     }
}
```

Light controlled servo:

Let's use our newly found light sensing skills to control a servo (and at the same time engage in a little bit of Genuino code hacking). Wire up a servo connected to pin 9 (like in CIRC-04). Then open the Knob example program (the same one we used in CIRC-08) **File > Examples > Servo > Knob**. Upload the code to your board and watch as it works unmodified.

Using the full range of your servo:

You'll notice that the servo will only operate over a limited portion of its range. This is because with the voltage dividing circuit we use the voltage on analog pin 0 will not range from 0 to 5 volts but instead between two lesser values (these values will change based on your setup). To fix this play with the val = map(val, 0, 1023, 0, 179); line. For hints on what to do, visit http://Genuino.cc/en/Reference/Map .

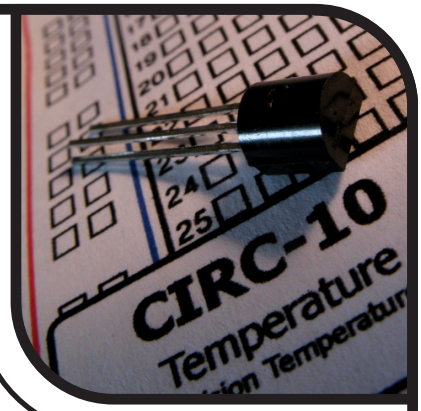
MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC09

.:Temperature:.

.:TMP36 Precision Temperature Sensor:.



What We're Doing:

What's the next phenomena we will measure with our Genuino? Temperature. To do this we'll use a rather complicated IC (integrated circuit) hidden in a package identical to our P2N2222AG transistors. It has three pins, ground, signal and +5 volts, and is easy to use. It outputs 10 millivolts per degree centigrade on the signal pin (to allow measuring temperatures below freezing there is a 500 mV offset eg. 25° C = 750 mV, 0° C = 500mV). To convert this from the digital value to degrees, we will use some of the Genuino's maths abilities. Then to display it we'll use one of the IDE's rather powerful features, the debug window. We'll output the value over a serial connection to display on the screen. Let's get to it.

One extra note, this circuit uses the Genuino IDE's serial monitor. To open this, first upload the program then click the button which looks like a magnifying glass or press **(ctrl + shift + m)**

The TMP36 Datasheet:
http://ardx.org/TMP36

The Circuit:

Parts:

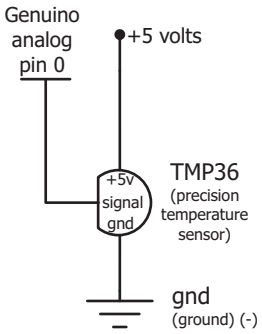
CIRC-10 Breadboard Sheet x1

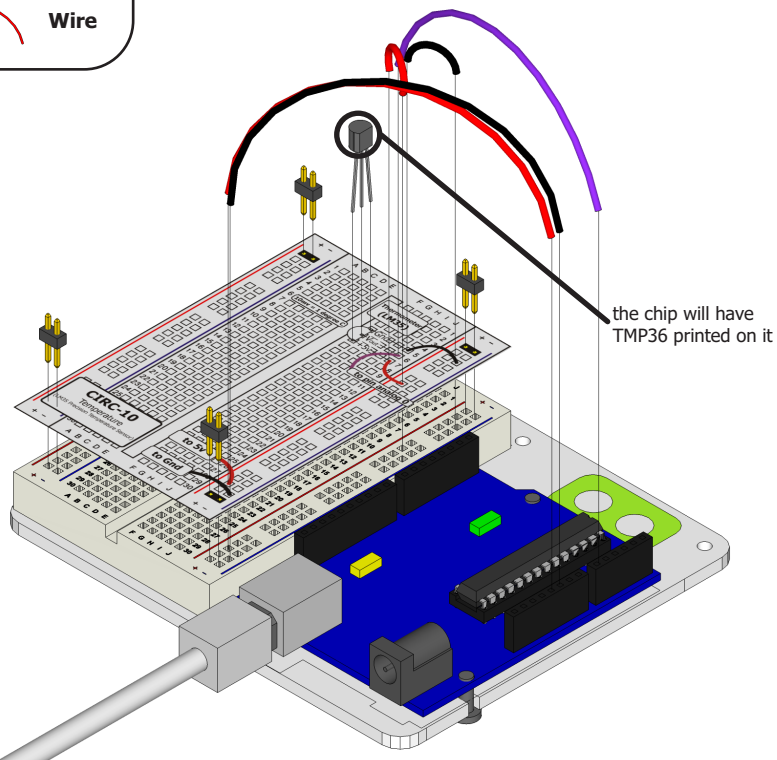
2 Pin Header x4

TMP36 Temperature Sensor x1

Wire

Schematic





The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS10
.:view:.
assembly video
http://ardx.org/VIDE10

CODE (no need to type everything in, just click)

Download the Code from (http://ardx.org/CODE10)

(copy the text and paste it into an empty Genuino Sketch)

```
/* ------------------------------------------------
 * |  Genuino Experimentation Kit Example Code |
 * |  CIRC-10 .: Temperature :.                |
 * ------------------------------------------------
 *
 *  A simple program to output the current temperature
 *  to the IDE's debug window
 *  For more details on this circuit:


//TMP36 Pin Variables
int temperaturePin = 0;//the analog pin the TMP36's
                       //Vout pin is connected to
                       //the resolution is
                       //10 mV / degree centigrade
                       //(500 mV offset) to make
                       //negative temperatures an
option

void setup()
{
  Serial.begin(9600);  //Start the serial connection
                       //with the computer
                       //to view the result open the
                       //serial monitor
                       //last button beneath the file
                       //bar (looks like a box with an
                       //antenna)
}

void loop()
// run over and over again
{
 float temperature = getVoltage(temperaturePin);
           //getting the voltage reading from the
           //temperature sensor


temperature = (temperature - .5) * 100;//converting from 10
mv
                       //per degree wit 500 mV offset to
                       //degrees ((volatge - 500mV) times
100)
 Serial.println(temperature); //printing the result
 delay(1000);                 //waiting a second
}

/*
 * getVoltage() - returns the voltage on the analog input
 * defined by pin
 */
float getVoltage(int pin){
 return (analogRead(pin) * .004882814);//converting from a 0
                       //to 1024 digital range
                       // to 0 to 5 volts
                       //(each 1 reading equals ~ 5
millivolts
}
```

NOT WORKING? (3 things to try)

Nothing Seems to Happen

This program has no outward indication it is working. To see the results you must open the Genuino IDE's serial monitor. (instructions on previous page)

Gibberish is Displayed

This happens because the serial monitor is receiving data at a different speed than expected. To fix this, click the pull-down box that reads "*** baud" and change it to "9600 baud".

Temperature Value is Unchanging

Try pinching the sensor with your fingers to heat it up or pressing a bag of ice against it to cool it down.

MAKING IT BETTER

Outputting voltage:

This is a simple matter of changing one line. Our sensor outputs 10mv per degree centigrade so to get voltage we simply display the result of getVoltage().

delete the line `temperature = (temperature - .5) * 100;`

Outputting degrees Fahrenheit:

Again this is a simple change requiring only maths. To go degrees C ----> degrees F we use the formula:

(F = C * 1.8) + 32)

add the line

```
temperature =
(((temperature - .5) * 100)*1.8) + 32;
```

before `Serial.println(temperature);`

More informative output:

Let's add a message to the serial output to make what is appearing in the Serial Monitor more informative. To do this first revert to the original code then change:

```
Serial.println(temperature);
            ---->
Serial.print(temperature);
Serial.println(" degrees centigrade");
```

The change to the first line means when we next output it will appear on the same line, then we add the informative text and a new line.

Changing the serial speed:

If you ever wish to output a lot of data over the serial line time is of the essence. We are currently transmitting at 9600 baud but much faster speeds are possible. To change this change the line:

`Serial.begin(9600); ----> Serial.begin(115200);`

Upload the sketch turn on the serial monitor, then change the speed from 9600 baud to 115200 baud in the pull down menu. You are now transmitting data 12 times faster.

MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC10

CIRC-11

.:Larger Loads:.
.:Relays:.

What We're Doing:

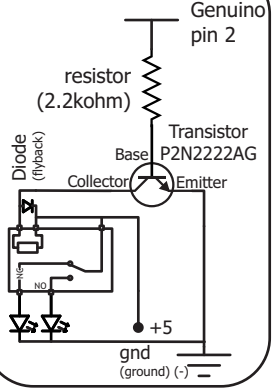
The final circuit is a bit of a test. We combine what we learned about using transistors in CIRC03 to control a relay. A relay is an electrically controlled mechanical switch. Inside the little plastic box is an electromagnet that, when energized, causes a switch to trip (often with a very satisfying clicking sound). You can buy relays that vary in size from a quarter of the size of the one in this kit up to as big as a fridge, each capable of switching a certain amount of current. They are immensely fun because there is an element of the physical to them. While all the silicon we've played with to this point is fun sometimes you may just want to wire up a hundred switches to control something magnificent. Relays give you the ability to dream it up then control it with your Genuino. Now to using today's technology to control the past. (The 1N4001 diode is acting as a flyback diode, for details on why it's there visit: http://ardx.org/4001)

The Circuit:

Parts:

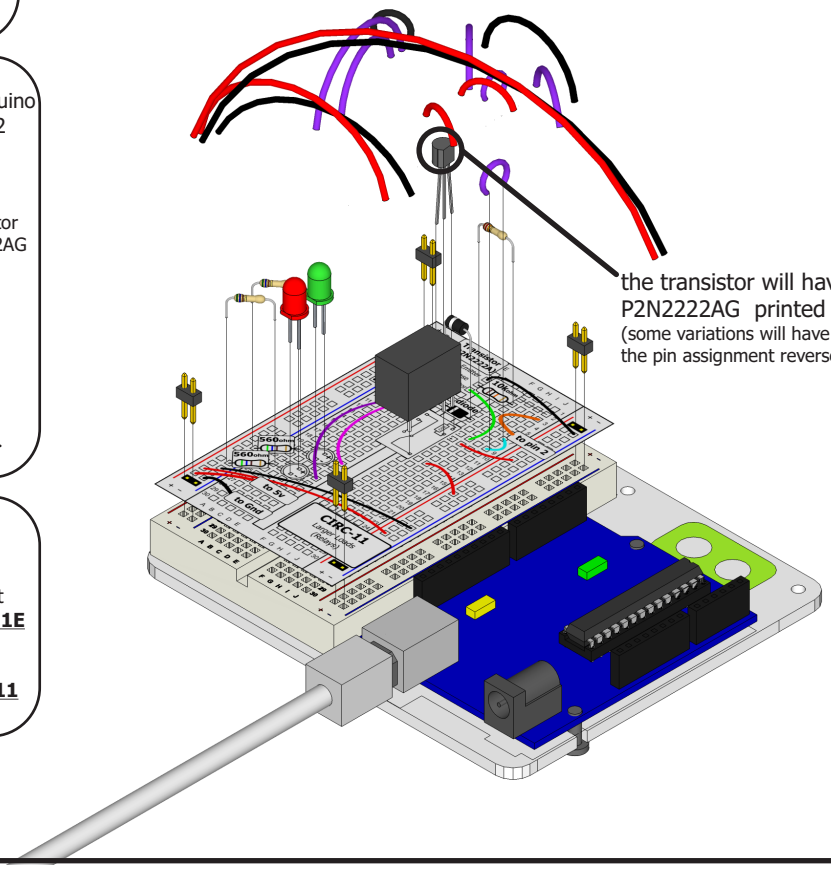
- **CIRC-11 Breadboard Sheet** x1
- **Diode (1N4001)** x1
- **Transistor P2N2222AG (TO92)** x1
- **Relay (SPDT)** x1
- **2.2k Ohm Resistor Red-Red-Red** x1
- **560 Ohm Resistor Green-Blue-Brown** x2
- **Green LED** x1
- **Red LED** x1
- **2 Pin Header** x4

Schematic



The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS11E
.:view:.
assembly video
http://ardx.org/VIDE11



the transistor will have P2N2222AG printed on it (some variations will have the pin assignment reversed)

CODE (no need to type everything in, just click)

File > Examples > 1.Basic > Blink

(example from the great Genuino.cc site, check it out for other great ideas)

```
/*
 * Blink
 *
 * The basic Genuino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Genuino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.Genuino.cc/en/Tutorial/Blink
 */

int ledPin = 2;                // ***********  CHANGE TO PIN 2 ************

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

NOT WORKING? (3 things to try)

Nothing Happens

The example code uses pin 13 and we have the relay connected to pin 2. Make sure you made this change in the code.

No Clicking Sound

The transistor or coil portion of the circuit isn't quite working. Check the transistor is plugged in the right way.

Not Quite Working

The included relays are designed to be soldered rather than used in a breadboard. As such you may need to press it in to ensure it works (and it may pop out occasionally).

MAKING IT BETTER

Watch the Back-EMF Pulse

Replace the diode with an LED. You'll see it blink each time it "snubs" the coil voltage spike when it turns off.

Controlling a Motor

In CIRC-03 we controlled a motor using a transistor. However if you want to control a larger motor a relay is a good option. To do this simply remove the red LED, and connect the motor in its place (remember to bypass the 330 Ohm resistor).

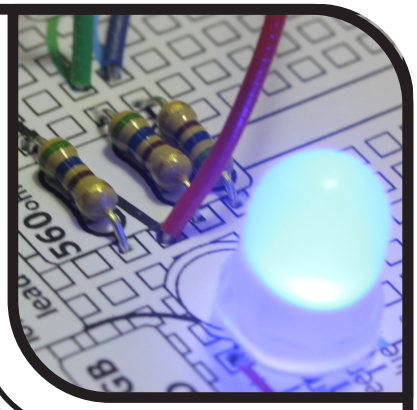
MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://ardx.org/CIRC11

CIRC-12

.:Colorful Light:.
.:RGB LEDs:.



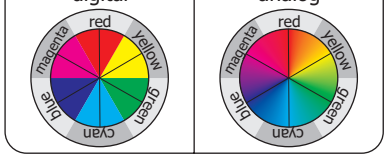
What We're Doing:

We've blinked an LED and controlled eight in sequence, now it's time to control color. Using an RGB LED (actual 3 LEDs in a single housing) we can generate any color our heart desires. We do this through color mixing, what's required is delving back to your elementary art days of playing with colored cellophane to produce different colors (if you can't remember that far back don't worry here's a color wheel to help you out).

Color Truth Table

red	green	blue	
ON	ON	OFF	yellow
OFF	ON	ON	cyan
ON	OFF	ON	magenta
ON	ON	ON	white

Color Wheels



The Circuit:

Parts:

CIRC-12 Breadboard Sheet x1

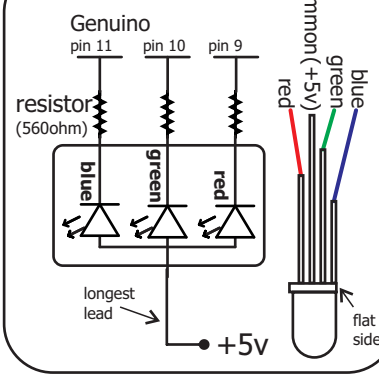
2 Pin Header x4

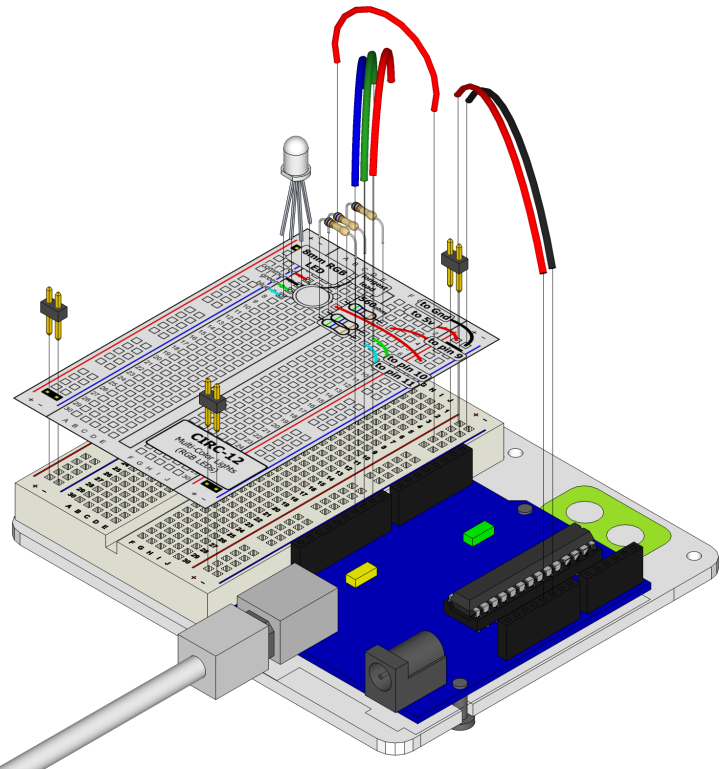
8mm RGB LED x1

Wire

560 Ohm Resistor Green-Blue-Brown x3

Schematic





The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS12E

CODE (no need to type everything in, just click)

Download the Code from (http://ardx.org/CODE12E)
(copy the text and paste it into an empty Genuino Sketch)

```
//RGB LED pins
int ledDigitalOne[] = {9, 10, 11};
   //the three digital pins of the digital LED
   //9 = redPin, 10 = greenPin, 11 = bluePin

const boolean ON = LOW;
   //Define on as LOW (this is because we use
   //a common Anode RGB LED (common pin is
   //connected to +5 volts)
const boolean OFF = HIGH;
  //Define off as HIGH

//Predefined Colors
const boolean RED[] = {ON, OFF, OFF};
const boolean GREEN[] = {OFF, ON, OFF};
const boolean BLUE[] = {OFF, OFF, ON};
const boolean YELLOW[] = {ON, ON, OFF};
const boolean CYAN[] = {OFF, ON, ON};
const boolean MAGENTA[] = {ON, OFF, ON};
const boolean WHITE[] = {ON, ON, ON};
const boolean BLACK[] = {OFF, OFF, OFF};

//An Array that stores the predefined colors
const boolean* COLORS[] =
  {RED, GREEN, BLUE,YELLOW, CYAN, MAGENTA,
   WHITE, BLACK};
```

```
void setup(){
  for(int i = 0; i < 3; i++){
   pinMode(ledDigitalOne[i], OUTPUT);
     //Set the three LED pins as outputs
  }
}

void loop(){
  setColor(ledDigitalOne, CYAN);
      //Set the color of the LED

  //randomColor()

}

void randomColor(){
  int rand = random(0, sizeof(COLORS) / 2);
    //get a random number within the range of
    //colors
  setColor(ledDigitalOne, COLORS[rand]);
    //Set the color of led one to a random color
  delay(1000);
}

void setColor(int* led, boolean* color){
 for(int i = 0; i < 3; i++){
   digitalWrite(led[i], color[i]);
 }
}
```

NOT WORKING? (3 things to try)

LED Remains Dark or Shows Incorrect Color

With the four pins of the LED so close together, it's sometimes easy to misplace one. Try double checking each pin is where it should be.

Seeing Red

The red diode within the RGB LED may be a bit brighter than the other two. To make your colors more balanced, try using a higher ohm resistor (or two resistors in series).

Looking For More?

(shameless plug)
If you're looking to do more why not check out all the lovely extra bits and bobs available from http://www.seeedstudio.com

MAKING IT BETTER

More Colors

I imagine you are less than impressed by the cyan glowing LED before you. To display a different color change the color in the code to one of the others.

```
setColor(ledDigitalOne, CYAN); ---->
       setColor(ledDigitalOne, **NEW COLOR**);
```

Display a Random Color

Of course we can do more than display a constant color, to see how we cycle through random colors change the `loop()` code to.

```
void loop(){
   //setColor(ledDigitalOne, CYAN);
   randomColor()
}
```

Analog Color Control

While switching between colors is good fun RGB LEDs really come into their own when mixed with analog control. Using PWM (pulse width modulation) it's possible to produce nearly any color and fade between them. Sadly the code for this is a bit too long for the section above, for an example program (with lots of comments).

Download the code from:
http://ardx.org/**MABE12E**

MORE, MORE, MORE:

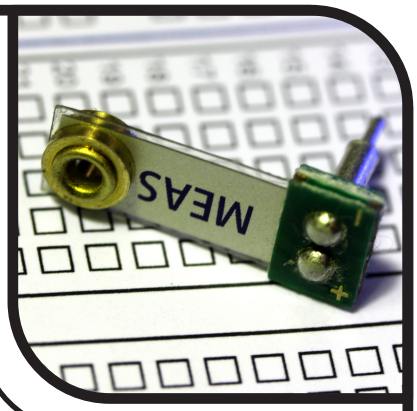
More details, where to buy more parts, where to ask more questions:

http://seeedstudio.com

CIRC-13

.:Vibration:.

.:Piezo Vibration Sensor:.



What We're Doing:

Earlier (CIRC-06) we used a piezo element as an output. But piezo elements have dual functionality, pulsing them with electricity produces a click and manipulating them produces a voltage. Using the Genuino's analog input we can measure the voltage generated and use it to determine whether any movement has occured. The piezo vibration sensor included in this kit, has an additional weight added this means it will flex when shaken, this lets us write programs that can respond to movement.

.: or for all the technical details a datasheet can be found here:.

.: http://ardx.org/**VIBS** :.

The Circuit:

Parts:

CIRC-13 Breadboard Sheet x1

2 Pin Header x4

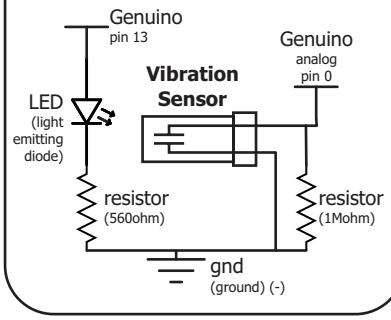
Piezo Vibration Sensor Minisense 100 x1

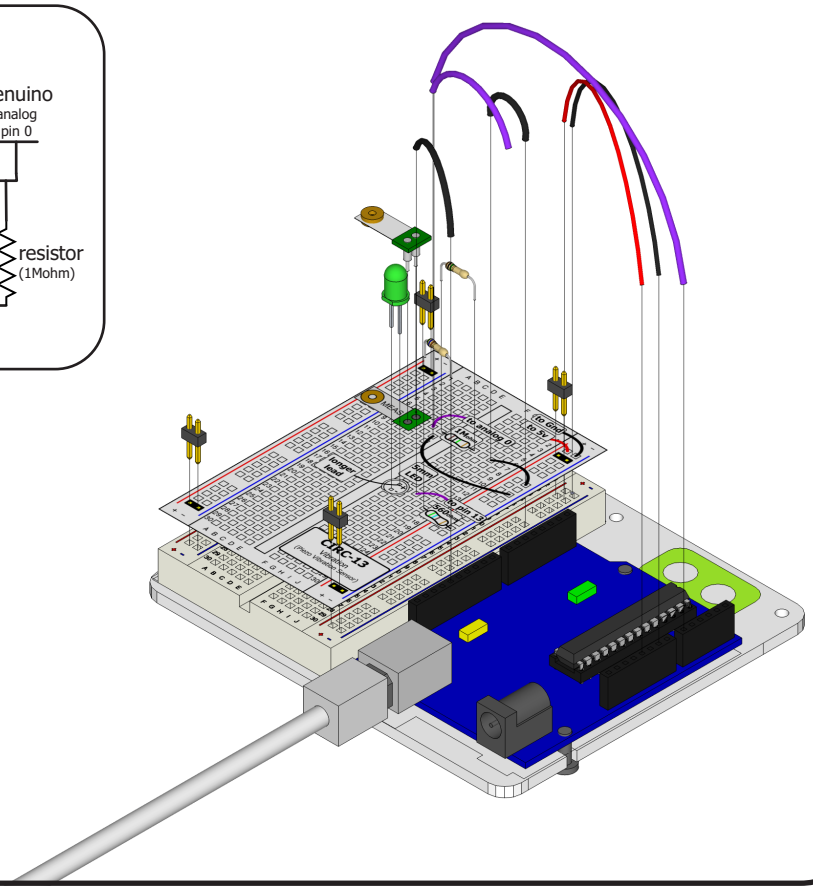
Green Led x1

560 Ohm Resistor Green-Blue-Brown x1

1 M Ohm Resistor Brown-Black-Green x1

Schematic





The Internet

.:download:.
breadboard layout sheet
http://ardx.org/BBLS13E

CODE (no need to type everything in, just click)

Download the Code from (http://ardx.org/CODE13E)
(copy the text and paste it into an empty Genuino Sketch)

```
/* Knock Sensor

   This sketch reads a piezo element to
   detect a knocking sound.
 */


const int ledPin = 13; // led connected to
                       // digital pin 13
const int knockSensor = A0;
     // the piezo is connected to analog pin 0
const int threshold = 100;
     // threshold value to decide when the
     // detected sound is a knock or not


int sensorReading = 0;
      // variable to store the value read
      // from the sensor pin
int ledState = LOW;
      // variable used to store the last LED
      //  status, to toggle the light

void setup() {
 pinMode(ledPin, OUTPUT);
            // declare the ledPin as an OUTPUT
 Serial.begin(9600);
            // use the serial port
}

void loop() {

  sensorReading = analogRead(knockSensor);
        //read the sensor

  // if the sensor reading is greater than the
  //  threshold:
  if (sensorReading >= threshold) {

    ledState = !ledState;
        // toggle the status of the ledPin:
    digitalWrite(ledPin, ledState);
        // update the LED pin itself:
    Serial.println("Knock!");
        // send the string "Knock!"
    delay(100);
        // delay to avoid overloading the serial
        // port buffer
  }
}


   *** This code is a modified version of the ***
            *** included example code ***
        *** Examples > 6. Sensors > Knock ***
```

NOT WORKING? (3 things to try)

LED Not Changing?

The vibration sensor is designed to be soldered rather than used in a breadboard. Because of this sometimes it isn't quite settled. Double check its connection.

No Response

The default threshold value is quite high. Sometimes you may need to flick the sensor, or shake it quite hard to see a response.

Looking For More?

(shameless plug)
If you're looking to do more why not check out all the lovely extra bits and bobs available from http://www.seeedstudio.com

MAKING IT BETTER

Adjusting the Sensitivity
To adjust the sensitivity simply change one line in the code:

```
      const int threshold = 100;
                      |
  const int threshold = *new number*;
```

For greater sensitivity use a smaller number, (as low as 1 will work but you may get some false readings)

Using the Piezo Buzzer
If you'd prefer to sense a knock rather than vibration, you can use the piezo element supplied with the kit (used in CIRC-06). To do this simply remove the vibration sensor and put the piezo element in its place. The pin spacing is different so you will need to move a couple of wires. Once this is done, tap the plastic casing to see the LED toggle. (If it doesn't try turning it 180 degrees or changing the sensitivity in the code).

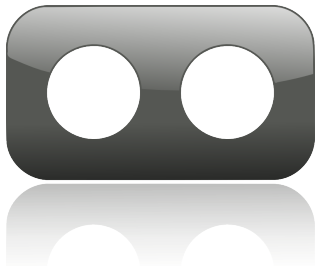
Fine Tuning
If you're interested in the exact values being received you can change the code to output the readings to the serial monitor. Simply download the code from
http://ardx.org/**MABE13E**
Upload to your Genuino and open the Serial Monitor (Ctrl+Shift+m)

MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

http://seeedstudio.com

www.oomlout.com



seeed studio
Open Hardware Facilitator