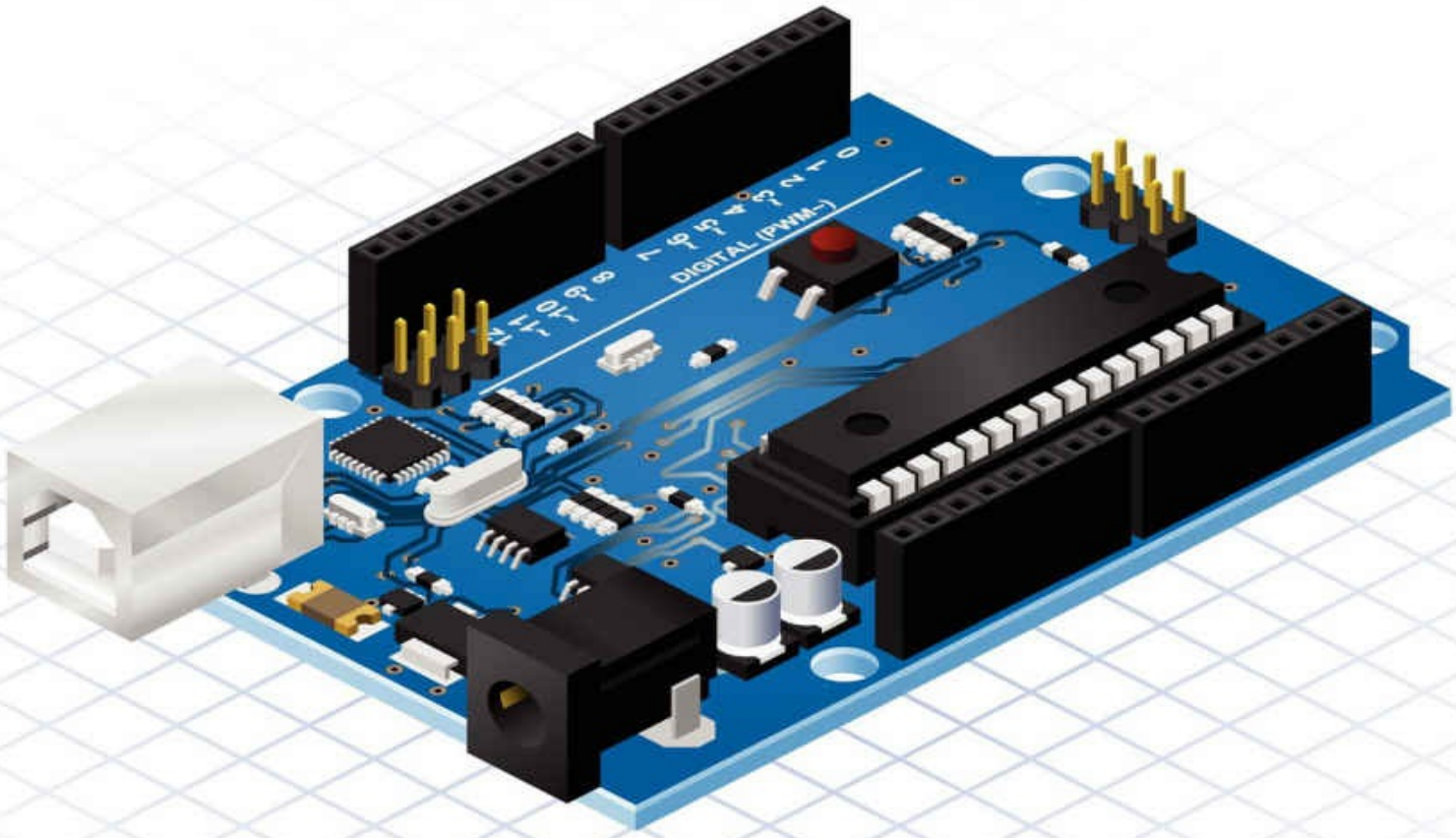
# Arduino

2ND EDITION

## ARDUINO USER GUIDE:

### OPERATING SYSTEM, PROGRAMMING, PROJECTS AND MORE!




Robert Scott

# *Arduino: Arduino User Guide for Operating system, Programming, Projects and More!*

# Contents

# Introduction

I want to thank you and congratulate you for purchasing the book, *"Arduino Board Pin Code Examples"*.

This book contains proven steps and strategies on how to get your Arduino board and compile code for projects. You learn how to use the Arduino for creating interactive devices around the house and build an in-system programmer. You can work with Arduino and become familiar with it.

Learn the basics for operating the Arduino board. You can work with prearranged boards if you are not too good with handling and assembling electronic components. You also have information on various Arduino hot selling products that you may use for your personal projects.

Thanks again for purchasing this book, I hope you enjoy it!

## Chapter 1

# USE ARDUINO TO INTERACT AND RUN CODE

Arduino is the latest sensation in the computer world. Forging a congenial environment through the use of a separate programming language, this interactive computing tool helps programmers detect and devise new methods for assessing and regulating various parameters. The system envisages the use of microcontrollers to assist in physical computing.

Arduino is a creative tool that helps you build something more with just ordinary desktop computer or circuits. It senses and thereby helps you control things in your physical world. It merges all needed software writing-development environments with a simple yet effective microcontroller board. In this e-book we will see Arduino products, hot selling items from the Arduino store and the method of programming the Arduino to create your own home automation system.

## Chapter 2

# ARDUINO PRODUCTS

Arduino produces boards, accessories, kits and shields. Currently they have 21 boards, 8 shields, 2 kits and 5 accessories. The boards are named *Arduino Uno, Arduino Yun, Arduino Tre, Arduino Esplora, Arduino Mega ADK, Arduino Robot, Arduino Mini, LilyPad Arduino Simple, Fio, Leonardo, LilyPad Arduino, Zero, Micro, Due, Arduino Ethernet, Mega 2560, LilyPad Arduino USB, Gemma, Nano, Arduino Pro Mini* and *Arduino Pro*.

The shields are *Arduino GSM Shield, Arduino WiFi Shield, Arduino USB Host shield, Arduino Wireless Proto shield, Arduino Ethernet shield, Arduino Wireless SD shield, Arduino Motor shield* and *LilyPad Arduino SimpleSnap*.

The kits are the *Arduino Starter kit* and *the Arduino Materia 101*. Accessories available are *USB/Serial Light adapter, mini USB/Serial adapter, TFT LCD screen, Arduino ISP* and *Arduino Proto shield*.

Arduino AtHeart is versatile and highly adaptable program for configuring programs and devices to control various parameters you need for your project. These programs benefit makers who want something more from Arduino. By joining in the program, you get the Arduino logo on your product, the support of the Arduino community and the chance to display your product on the Arduino store. You can describe your product on the Arduino website. Small makers have to remit a small fee for licensing the product while wholesalers would have to pay royalty. The supported processors are:

| Chip Name | Clocked at MHz |
|---|---|
| AtMega328 | 8 or 16 |
| AtMega1280 | 16 |
| AtMega2560 | 16 |
| AtMega32U4 | 16 |
| SAM3X | |

Arduino AtHeart devices have the basic simplicity to allow users to assemble

various models on their own. The functionality of these devices goes beyond the reach of ordinary computer based applications. Here is a look at some Arduino AtHeart products that are in hot demand at the Arduino store.

- **Primo**

Primo is an Arduino AtHeart product aimed at teaching children in the learning phase the basics of programming. This physical interface programming device teaches children in the age group 4 – 7 years programming logic. Children will guide a smiling robot to a prescribed destination by passing on instructions. These instructions are delivered in the form of colorful blocks with messages. It makes children involve in a developmental experience that is both exciting and challenging. Children learn through playing with their Cubetto Playset.

- **Tsunami**

Tsunami is an Arduino AtHeart product signal generator powerful yet flexible and gives users a way to experiment with projects using analog signals. It is amazingly straightforward since it combines the Direct Digital Synthesis chip with Arduino Leonardo. When combined with a software library that is easy to use and an input-output circuitry that is incredibly flexible, working with analog signals becomes a snip. *Arachnid Labs* designed the Tsunami and is currently in use in over a thousand projects covering fields like music, design dance, publishing, theater, technology, photography and fashion. To preorder the Tsunami go here.

- **Nix Color Sensor**

This Arduino AtHeart device will tickle the senses of art lovers. Nix color sensor allows one to collect colors from any object they see in real life. Take the sensor close to the object and touch it and you will find the color automatically registered inside the Nix color sensor.

Organize them into palettes using your smartphone. You can share these over

the social media and make comparisons to other colors. This proves very useful for those in the fashion, style or review of products sections of the media. Never go wrong when you choose the color of the paint for your walls.

## • Bare conductive touch board

Serious people who like to 'wire' the place up will definitely fall for this one. Here in this Arduino AtHeart product you can make any surface respond to touch sensation. Okay even fun lovers will have a ball with the Touch Board.

All you have to do is connect one of the twelve electrodes to anything conductive and then use the signal to trigger some sound that will play through an MP3 player. Think of all the fun you can have with the electric paint that you can apply on your wall, your girl's teddy bear…This is called the Touch Board and… better think before you touch it.

## • EZBoard Home Automation

Control the various devices around the house with your *EZBoard*. Think thermostat, light control, controlling sprinkler and temperature besides your garage door. All the necessary code, libraries and basic programs have been installed making it very simple for you to use this Arduino AtHeart product. You have a connected home at very affordable cost.

This Arduino AtHeart home automation device comes in a box, and nothing is as simple getting the package out, setting up the connection to the Wi-Fi and…begin to control the appliances. The board does not draw much power and you can use it extended periods just on batteries pegging 3.7V. It uses an Ethernet controller and consumes very less power. In addition, it has a power relay, temperature sensor, and microSD card socket. Its Lelylan's platform helps you interact with any hardware one normally uses in one's house.

Lelylan makes use of the same protocol Facebook uses to update application that are used in mobile messaging known as MQTT protocol. You can adapt this program for any platform.

# • NEWTC Prototyping Board

You can either buy a preassembled board from NEWTC or you can get a board and assemble the individual parts yourself. The second option obviously is more exciting and offers you the chance to dabble in some experiments in designing. Don't worry if things come unstuck, the Arduino community are always ready to back you and answer all your questions.

The CPU used in the Prototyping board is the ATMEGA328P-PU. You will also find that the Arduino Uno bootloader is already burned into the chip. Typical behavior of the Do-It-Yourself kit is as listed below.

- Has the prototyping board self assembled variety Arduino AtHeart having Arduino Uno bootloader pre-burned into it.
- Gives support for HC-06 Bluetooth connector interface
- TTL Level (5V) 4-pin connector
- Supports MCU and compatible connectors
- Power external 12 V with 3.3V and 5 V internal

Get the individual parts and begin to solder them. You can order your kit from online stores. You can solder the individual parts on the board and test it with DM-USB2Serial. Upload the program for the DM-USTYLE V1.0 DIY version board from the sketch software. Plug the board in and click on **Program Compile** and **Upload** menu.

For those who wish to use the board as an AVR board, you have a choice of compilers ranging from WINAVR and IAR to CodeVision AVR and ICC AVR. Normally, users will employ the board ISP to download the output generated by the compiler. You can use the AD-ISPRO or the AD-USBISP for doing the download.

The former is useful for parallel port type STK200/300 while the latter proves useful for USB type STK500.

## Chapter 3

# USING THE IMAGECRAFT IDE

Now we will see how to establish the compiler to render code in a hex file and how we should download it to the device. This example will show you how to use the STK200 starter kit for the AT90S23 13. You can get more details from ImageCraft website. Be sure to get the "unlock code" from ICC. This makes your program licensed. You then have to install the program to a default drive say C, the short cut to the program -- ICC AVR will be installed to the "c:\icc" directory. Search in the **Menu** program and find the group **ImageCraft Development Tools**.

## Begin your project

When you start up the ICC AVR program, you will see three distinct areas – **project file list**, **status window** and **editors**. At the beginning, all are empty. All project files are stored in the project file list. Every editor window that is syntax aware will be allocated one a separate tab in the editor section. During the project building process every message will be displayed in your status window. You can maximize space for your project file list by choosing not to display the status window or the editor.

Select Project New and in the dialog box make a new folder c:\icctest. Type in **First** for the file name in the file name window. Your project now has the name **First**. To write and compile code, your compiler must have the correct options. Projects need to have settings and you can choose the options from the Options **tab under the Projects**.

From the options box, choose the **Paths** tab. The directory path for the output directory, libraries and header files will be specified here. The paths for library path and include path will automatically get set to "c:\icc\lib"and "c:\icc\include".

In the output directory you will find files related to the project along with the output. Unless specified as absolute, it remains relative to project directory. In the edit box, if you enter "objs", IDE will "c:\icctest\first\objs" for outputting files.

## Compiler

Go to the compiler page in the dialog box and choose the option. In the list box, click on **Output Format** and click on **Intel HEX**. Keep the box next to the "**Accept C++ Comments**" checked.

## Target

Next click on the **target** tab. Here you can make options specific to the target. In this particular tutorial, you have to click on **Device Configuration** tab and select 2313. This ImageCraft IDE becomes very simplified as it as it shows the options that are required alone for the entire tool chain in the compiler. This will include linker, assembler and compiler.

A simple program will be in use by the application note to increment PORTB where you have the 8 LEDs attached. One 8-bit timer will generate delay for each increment. You will now be able to see the lights flashing. Begin by creating a new file and type:

```
#include <io2313.h>
void initialization (void);
void delay (void);

void initialization (void)
{
DDRB = 0xff;                                    // set PORTB as output
TCCR0 = 0x05;                                   // Count clock/1024
}
void delay (void)                               //Produce a delay of 65 ms
at 4 MHz
{
while (! (TIFR&0x02));
                                    // Wait for timer () overflow flag to be set
TIFR = 0x02;                                    // Clear overflow flag
}

void main (void)
{
initialization ();                              // Initialize Peripherals
while (1)                                       // Forever
{
PORTB++;                                        //Increment PORTB
delay ();                                       //Short delay
}
}
```

Save this file as “avrc031.c” under the **File**. In addition, you can also save it to project folder “c:\icctest\First” but this is optional.

## About the program

The program runs in three modes. The main loop is separated from the other two operations delay and initialization. Main clock divided by 1024 begins counting TIMER () while PORTB acts as the output. Controller waits in the delay subroutine until the **Overflow Flag** of TIMER () gets set. It then clears this flag and exits. Main loop begins with the increment in PORTB of the content. Then the delay is called so that the change is visible in the PORTB.

## Make The Inclusion For The Source File In The Project

To include the source code for any project, open the **Add Files** under **Projects**. Open the working file “avr031.c” inside “c:\icctest\first”and click OK.

Compiling the code: Select **Make Project** under **Project**. The toolbar will display the icon for **Build Project**. If the code does not have any errors then the code will compile and the HEX code will get allocated to “c:\icctest\first\objs\first.hex”. Now all you have to do is to place the file inside the starter kit.

## Loading File Into STK200 Starter Kit

Before we run the code, we convert the file into AT90S2313 in the STK200 starter kit. We use software known as AVR ISP. Open a new project once you have mounted the STK dongle in a parallel port. Select **New Project** under **Project**. Highlight and click **OK** on AT90S2313. You have extra options for setting “ **Lock Bit**” and “**Fuse**” under “**Project Manager**” but it will not be useful for this tutorial. Click the title frame and activate **Program Memory** window.

Go to **File** tab and click on **Load**. A dialog box will open where you can see the “c:\icctest\first\objs” file. Open this and click on the “first.hex” file. Now load this program on the AT90S2313 in the starter kit by selecting the **Auto-Program**

option under **Program**. Check the boxes next to "**Reload Files**", "**Program Device**" and "**Erase Device**". Click on **OK** now. The LED lights should be functioning well now.

# RECAP OF THE TUTORIAL

Begin by creating a destination folder. **Project** New

Write Project name and path **Project Options**

Enter **Objs**

Compiler Options: under output format check both **Intel HEX** and **Accept C++ Comments**

Click **Device Configurations** and choose **2313**

Write source code

**Project** Add **Files** and add source code to project; choose the file you just wrote

Click on **build** icon and compile the code; alternatively you can compile by selecting **Make Project** under **Project**

Open AVR ISP. Download hex file present in "objs" folder.

For those who need to download the Arduino software go here or use the links below.

Windows OS

Mac OS (10.7 Mountain Lion or later)

Linux 32 bits

Release notes

Linux 64 bits

Source Codes

Checksums

For detailed instruction on installation, refer this page.

# Chapter 4

## BUILD THE IN-SYSTEM PROGRAMMER USING ARDUINO

Here you will learn how to use your Arduino board to be your In-System Programmer. That means you make use of the board as AVR ISP. When you do this you get the capacity to burn the ATMEGA328 or ATMEGA168 bootloader on some AVR. Randall Bohn designed the mega-isp firmware on which we base the code for this tutorial. Here are the instructions step by step.

- Plug your Arduino board in and go to **Examples**.
- Click on **ArduinoISP Firmware** (Please note that those who use Arduino 1.0 have to make one change in their ArduinoISP code. Check the heartbeat () function. You will see that it says delay (40). Make it delay (20).
- According to the board you use, select items in **Serial Port** and **Board** menus under the **Tools**.
- Now upload Arduino sketch to the board.
- Use the figure shown below to wire the Arduino board. (Please note that if you use Arduino Uno, a 10 µF capacitor will be required between Ground and Reset.)

**MISO = - VTG**

**SCK - - MOSI**

**RST - - GND**

**ISP6PIN**

- When you next select the **Board** under the **Tools**, it actually means you are selecting the board on which you intend to burn the bootloader. It does not mean the board you are currently using to work the program.
- Then click on the **Arduino as ISP** under **Burn Bootloader**.

# Chapter 5
# WORK WITH ARDUINO

## I) EXAMPLES FROM BRIDGE LIBRARY

Here are the examples (read more at Arduino Learning Examples) that will help you work with your Arduino. All the examples written here are for Arduino 1.0 and later versions. This will help you develop an understanding of how your Arduino can adapt to different circumstances and how you can develop your skills in this direction.



## A) CHECK WIFI STATUS OF ARDUINO YUN

For this example, you need to have Arduino Yun microprocessor board. It comes with Ethernet, digital pins for input—output numbering 20 and WiFi so that you can begin straight away, unless there are legal restrictions in your country for use of WiFi-enabled devices.

Arduino Yun has slot for micro-SD card, another for micro-USB-connectivity, then 3 reset-buttons and the UCSP header. The *Atheros* processor in your Yun supports Linux distribution OpenWrt-Yun. This allows the users good degree of computer networking power along with working ease brought by Arduino.

**Start Check Status of Yun WiFi project**

When you run the script "pretty-wifi-info.lua" depicted in the sketch given in your Yun in folder /usr/bin, you will get information about the status of your Yun WiFi. Be careful however, to connect the Yun in serial connection otherwise, it will not print. Connect the computer USB cable to the Yun and choose the right port from the Port Menu.

**Hardware and Circuitry**

You do not have to make any connections or circuits for this project. You will require these pieces of hardware.

a) Wireless network
b) Arduino Yun

**Code for the WiFi Status project**

First, you need to include Process class in header.

```
#include <Process.h>
```

Next, you need to start Bridge and serial communication. If you fail to connect the computer in serial, sketch will fail to run.

```
void setup () {
        Serial.begin (9600);
        while (!Serial)
  Serial.println("Starting bridge...\n");
  pinMode(13,OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin();
  digitalWrite(13, HIGH);  // Led on pin 13 turns on when the bridge is ready

  delay(2000);
}
```

New process is initialized in loop (). This will run the check script for the WiFi. Script is run when you call runShellCommand () after setting path to script.

```
void loop () {
  Process wifiCheck;
  wifiCheck.runShellCommand ("/usr/bin/pretty-wifi-info.lua");
```

Check the characters returned to serial monitor by the script. After a few seconds,

run it again.

```
while (wifiCheck.available() > 0) {
    char c = wifiCheck.read();
    Serial.print(c);
  }

  Serial.println();

  delay(5000);
}
```

Here is the entire code once again.

```
/*
 Status of WiFi

 This sketch runs a script called "pretty-wifi-info.lua"
 installed on your Yún in folder /usr/bin.
 It will print information regarding status of your wifi connection.

 It uses Serial to print, so you need to connect your Yún to your
 computer using a USB cable and select the appropriate port from
 the Port menu

 created  18 June 2013
 By Federico Fissore

 This example code is in the public domain.
 */
```

```
#include <Process.h>

void setup() {
  Serial.begin(9600);  // initialize serial communication
  while(!Serial);      // do nothing until the serial monitor is opened

  Serial.println("Starting bridge...\n");
  pinMode(13,OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin();  // make contact with the Linux processor
  digitalWrite(13, HIGH);  // Led on pin 13 turns on when the bridge is ready

  delay(2000);  // wait 2 seconds
}

void loop() {
  Process wifiCheck;  // initialize a new process

   wifiCheck.runShellCommand("/usr/bin/pretty-wifi-info.lua");  // command you want to run

  // while there's any characters coming back from the
  // process, print them to the serial monitor:
  while (wifiCheck.available() > 0) {
    char c = wifiCheck.read();
    Serial.print(c);
  }

  Serial.println();

  delay(5000);
}
```

# B) FILE WRITE SCRIPT

This example tells you how to use FileIO classes to write to some file within filesystem of Yun. First in /tmp a shell script file will be created. Then you execute it.

**Hardware and Circuit required**

You do not have to make any circuits and only require your Arduino Yun board.

**Code for File Write Script**

To communicate with filesystem, you have to include FileIO header.

#include <FileIO.h>

Next, you have to initialize FileSystem Serial communication and Bridge in setup (). Once the serial connection has been made, start loop () after you upload file after you initiate a custom call function uploadScript ().

```
void setup () {
  Bridge.begin ();
  Serial.begin (9600);

  while (!Serial);  // wait for Serial port to connect.
  Serial.println ("File Write Script example\n\n");

  FileSystem.begin();

  uploadScript ();
}
```

One more custom function will run your script. This runScript () will execute

every 5 seconds.

```
void loop () {
  runScript ();
  delay (5000);
}
```

Create a file with uploadScript () function. The shell script is created inside Linux file system. This will check your WiFi interface network traffic. After you have finished creating the file, you may open it by creating instance of File@ class. You call it by invoking FileSystem.open ()@ the precise location where you create the script is indicated by @. Put the script in /tmp inside RAM. This will preserve the cycles of read/write that are limited in FLASH memory.

```
void uploadScript() {
  File script = FileSystem.open("/tmp/wlan-stats.sh", FILE_WRITE);
```

Print the header #!/bin/s.

Then print utility *ifconfig*.

The *@ifconfig@* command line utility controls network interfaces. This brings you to the WiFi interface *wlano*

Write script contents using File.print ().

The output from ifconfig will be sought out by grep utility.

Since you require the received number of bytes look for keywords *RX bytes*. Now you can close the file.

```
script.print("#!/bin/sh\n");
  script.print("ifconfig wlan0 | grep 'RX bytes'\n");
  script.close();  // close the file
```

In order to make script executable, you must instantiate some Process.

Now, chmod@ command changes file modes; send chmod@ with filepath and

command.  Shell script now will run like an application.

```
Process chmod;
  chmod.begin("chmod");      // chmod: change mode
  chmod.addParameter("+x");  // x stays for executable
  chmod.addParameter("/tmp/wlan-stats.sh");
  chmod.run();
}
```

Create Process through runScript () function. This will run your script and then print the result to Serial Monitor. Start the script after creating it by calling the two functions Process.begin (filepath) along with Process.run ().

```
void runScript() {
  Process myscript;
  myscript.begin("/tmp/wlan-stats.sh");
  myscript.run();
```

Now use String function to read and hold output.

```
String output = "";

  while (myscript.available()) {
    output += (char)myscript.read();
  }
```

Now get rid of blank spaces that precede or are present at the string end and then print the output to serial monitor.

```
output.trim();
  Serial.println(output);
  Serial.flush();
}
```

The complete code will look like this:

```
/*
  Write to file using FileIO classes.

 This sketch demonstrates how to write file into the Yún filesystem.
 A shell script file is created in /tmp, and it is executed afterwards.

 Created 7 June 2010
 by Cristian Maglie

 This example code is in the public domain.

 */

#include <FileIO.h>

void setup() {
  // Setup Bridge (needed every time we communicate with the Arduino Yún)
  Bridge.begin();
  // Initialize the Serial
  Serial.begin(9600);

  while(!Serial);  // wait for Serial port to connect.
  Serial.println("File Write Script example\n\n");

  // Setup File IO
  FileSystem.begin();

  // Upload script used to gain network statistics
  uploadScript();
}

void loop() {
```

```
    // Run stats script every 5 secs.
    runScript();
    delay(5000);
  }

  // this function creates a file into the Linux processor that contains a shell script
  // to check the network traffic of the WiFi interface
  void uploadScript() {
    // Write our shell script in /tmp
    // Using /tmp stores the script in RAM this way we can preserve
    // the limited amount of FLASH erase/write cycles
    File script = FileSystem.open("/tmp/wlan-stats.sh", FILE_WRITE);
    // Shell script header
    script.print("#!/bin/sh\n");
    // shell commands:
    // ifconfig: is a command line utility for controlling the network interfaces.
    //        wlan0 is the interface we want to query
     // grep: search inside the output of the ifconfig command the "RX bytes" keyword
    //      and extract the line that contains it
    script.print("ifconfig wlan0 | grep 'RX bytes'\n");
    script.close();  // close the file

    // Make the script executable
    Process chmod;
    chmod.begin("chmod");     // chmod: change mode
    chmod.addParameter("+x");  // x stays for executable
       chmod.addParameter("/tmp/wlan-stats.sh"); // path to the file to make it executable
    chmod.run();
  }
```

```
// this function run the script and read the output data
void runScript() {
  // Run the script and show results on the Serial
  Process myscript;
  myscript.begin("/tmp/wlan-stats.sh");
  myscript.run();

  String output = "";

  // read the output of the script
  while (myscript.available()) {
    output += (char)myscript.read();
  }
  // remove the blank spaces at the beginning and the ending of the string
  output.trim();
  Serial.println(output);
  Serial.flush();
}
```

# C) HTTP CLIENT

This tutorial will show how to use the Arduino Yun for creating HTTP client. Using this you can go online and download content. In our example we go to Arduino website to download any one version of Arduino logo in the form of ASCII characters.

**Hardware and Circuit Required**

- Arduino Yun
- Connection to internet

You do not require any other circuit for this example.

**Code**

Begin by including libraries of HTTPClient as well as Bridge.

```
#include <Bridge.h>
#include <HttpClient.h>
```

Start Bridge in setup () and ensure serial connection has been made. Then you can begin the loop ().

```
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin();
  Serial.begin(9600);
  while(!Serial);
}
```

Create an instance of HTTPClient in loop (). Call the URL of your choice with client.get (URL) function.

```
void loop() {
  HttpClient client;
  client.get("http://arduino.cc/asciilogo.txt");
```

Check the memory of client buffer server and read bytes to print them in your serial monitor. Use a cycle of 5 seconds.

```
while (client.available()) {
    char c = client.read();
    Serial.print(c);
  }
  Serial.flush();
```

```
  delay(5000);
}
```

The entire code for sketch is given here.

```
#include <Bridge.h>
#include <HttpClient.h>

void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin();
  Serial.begin(9600);
  while(!Serial);
}

void loop() {
  HttpClient client;
  client.get("http://arduino.cc/asciilogo.txt");

  while (client.available()) {
    char c = client.read();
    Serial.print(c);
  }
  Serial.flush();

  delay(5000);
}
```

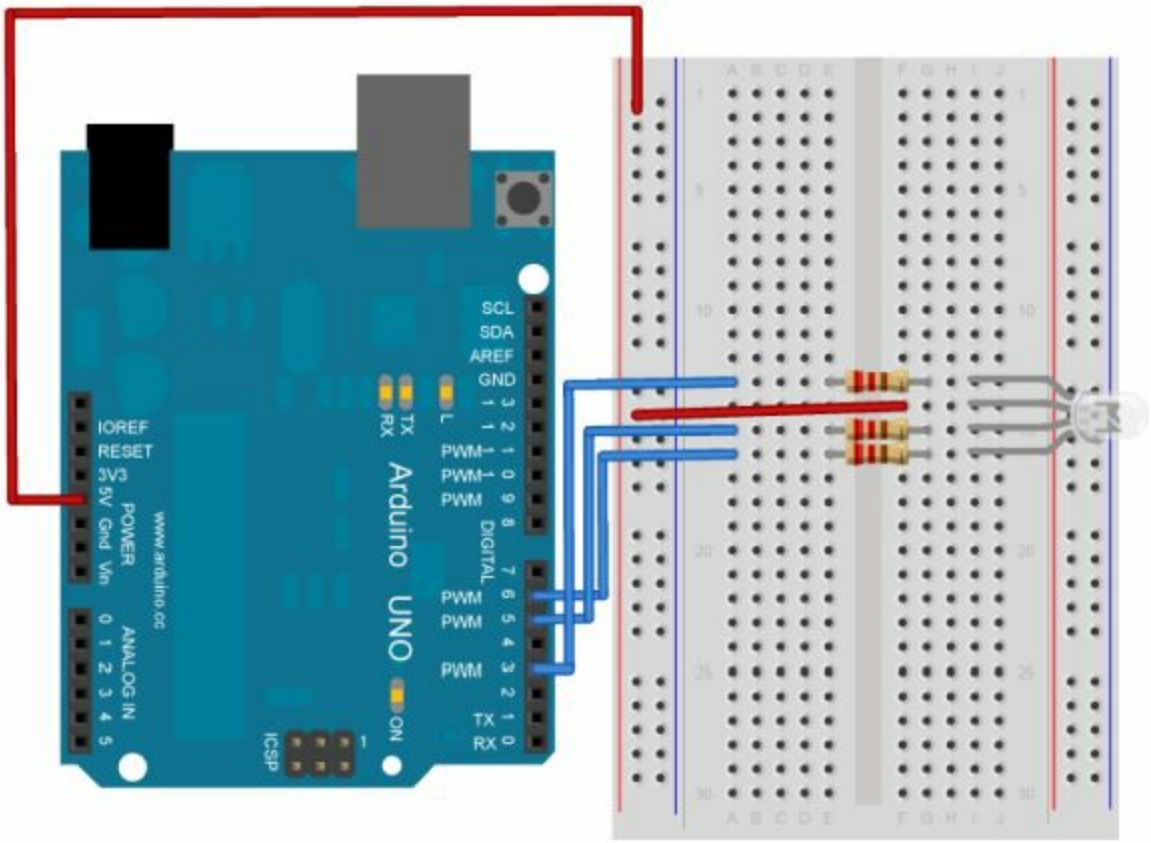
# II) EXAMPLES FROM COMMUNICATION

## A) READ ASCII STRING

In this sketch, we see how to use the Serial.parseInt () function. This helps you locate values that have alphanumeric character separation. One common method of representation is the comma-separated-value where comma is made use of to separate the different pieces of information. You can also use other characters like period or space for this purpose. You will change the RGB LED color when you pass strings having values such as “15, 140, 220”. The color changes because of the parsed values of ints.

## Hardware and Circuit required

- Arduino Board
- Three 220-ohm resistors
- Breadboard
- Common anode RGB LED
- Hookup wire




To make the circuit shown above, take 5 wires. Use a red wire to connect the 5V pin to one end of one of the long vertical rows present on your breadboard. Take an RGB LED and check the datasheet in order to identify the pins. To the LED common anode, connect power rail you created now.

Now use the wires that remain to make connection to pin 5 and green cathode, pin 3 and red cathode and pin 6 with blue cathode along with resistors. All RGB LEDs share power pin that is common if they have a common cathode.

Now, make a pin LOW instead of HIGH and across that diode, some voltage difference will happen. Now, if you send 255 using analogWrite () he LED will turn off, while the value 0 will turn it on full brightness. In effect, you will call analogWrite (pin, 255-brightness) instead of analogWrite (pin, 255). There is nothing complicated in the mathematics.

## Code for the Read ASCII String

You need to set up global variables for the pins to which you connect the LED. This will help you identify easily which one of them is red, which is green and which is blue when you move on to main program part.

```
const int redPin = 3;
const int greenPin = 5;
const int bluePin = 6;
```

Serial communication will begin between your computer and Arduino in setup () at 9600 bits per second. Also, configure the pins to be outputs.

```
Serial.begin(9600);
pinMode(redPin, OUTPUT);
pinMode(greenPin, OUTPUT);
pinMode(bluePin, OUTPUT);
```

You need to make sure that every data present in serial buffer is read. For this we use while function.

```
while (Serial.available() > 0) {
```

Now you come to the main portion where you note the brightness of the LED, which is the main serial information. Declare any local variable and use it to store the value. You separate every value by commas using the Serial.parseInt (). Then you read information into variables.

```
int red = Serial.parseInt();
int green = Serial.parseInt();
int blue = Serial.parseInt();
```

After this is done, you can use the newline character to proceed to the next step.

```
if (Serial.read() == '\n') {
```

In order to maintain PWM control, try to keep values within acceptable range through constrain (). So, whenever the value lies outside your accepted PWM range, it will be limited to some particular valid number. If you subtract this from 255 you will format value for use with LED with common anode. You already know that the LED will light up when there is a difference of voltage between pin connected for Arduino and anode.

```
red = 255 - constrain(red, 0, 255);
green = 255 - constrain(green, 0, 255);
blue = 255 - constrain(blue, 0, 255);
```

After formatting PWM values, make use of analogWrite () and change LED color. Since you already subtracted the value of output from 255, you get:

```
analogWrite(redPin, red);
analogWrite(greenPin, green);
analogWrite(bluePin, blue);
```

Make the entire set of value of LED into one string and send back to serial monitor.

```
Serial.print(red, HEX);
Serial.print(green, HEX);
Serial.println(blue, HEX);
```

Lastly, you have to close those brackets from your while statement, if statement and main loop.

```
    }
  }
}
```

Now that Arduino has been programmed, go back to Serial monitor. Start your

message in a newline and send values for lights between 0 and 255 as: Red, Green, Blue. Once this is done, LEDs connected will turn to the specified color and you can read HEX values from Serial monitor.

```
/*
 Reading a serial ASCII-encoded string.

 This sketch demonstrates the Serial parseInt() function.
 It looks for an ASCII string of comma-separated values.
 It parses them into ints, and uses those to fade an RGB LED.

 Circuit: Common-anode RGB LED wired like so:
 * Red cathode: digital pin 3
 * Green cathode: digital pin 5
 * blue cathode: digital pin 6
 * anode: +5V

 created 13 Apr 2012
 by Tom Igoe

 This example code is in the public domain.
 */

// pins for the LEDs:
const int redPin = 3;
const int greenPin = 5;
const int bluePin = 6;

void setup() {
  // initialize serial:
  Serial.begin(9600);
  // make the pins outputs:
  pinMode(redPin, OUTPUT);
```

```
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

}

void loop() {
  // if there's any serial available, read it:
  while (Serial.available() > 0) {

    // look for the next valid integer in the incoming serial stream:
    int red = Serial.parseInt();
    // do it again:
    int green = Serial.parseInt();
    // do it again:
    int blue = Serial.parseInt();

    // look for the newline. That's the end of your
    // sentence:
    if (Serial.read() == '\n') {
      // constrain the values to 0 - 255 and invert
        // if you're using a common-cathode LED, just use "constrain(color, 0,
255);"
      red = 255 - constrain(red, 0, 255);
      green = 255 - constrain(green, 0, 255);
      blue = 255 - constrain(blue, 0, 255);

      // fade the red, green, and blue legs of the LED:
      analogWrite(redPin, red);
      analogWrite(greenPin, green);
      analogWrite(bluePin, blue);

      // print the three numbers in one string as hexadecimal:
      Serial.print(red, HEX);
```

```
      Serial.print(green, HEX);
      Serial.println(blue, HEX);
    }
  }
}
```

## B) VIRTUAL COLOR MIXER

In this example, we see how we send different values from your Arduino board to the computer. Background color of Max/MSP patch or processing sketch gives the readings. The potentiometers set blue, green or red components.

### Hardware Required

- Arduino Board
- Breadboard
- (3) Analog Sensors (potentiometer, photocell, FSR, etc.)
- Hook-up wire
- (3) 10K ohm resistors

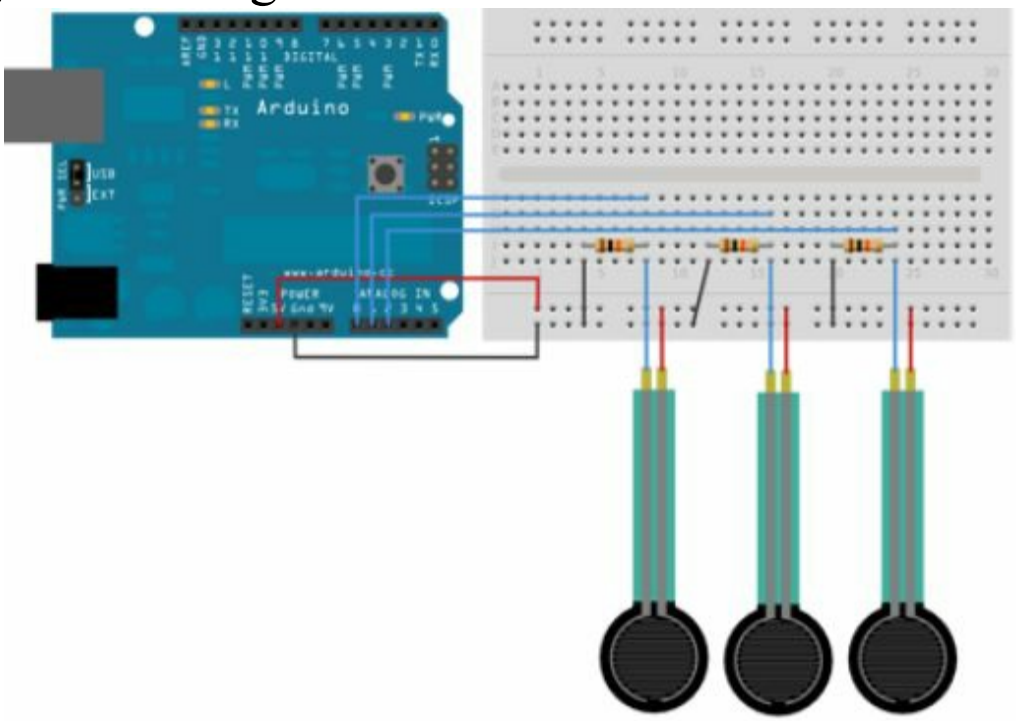
### Software Required

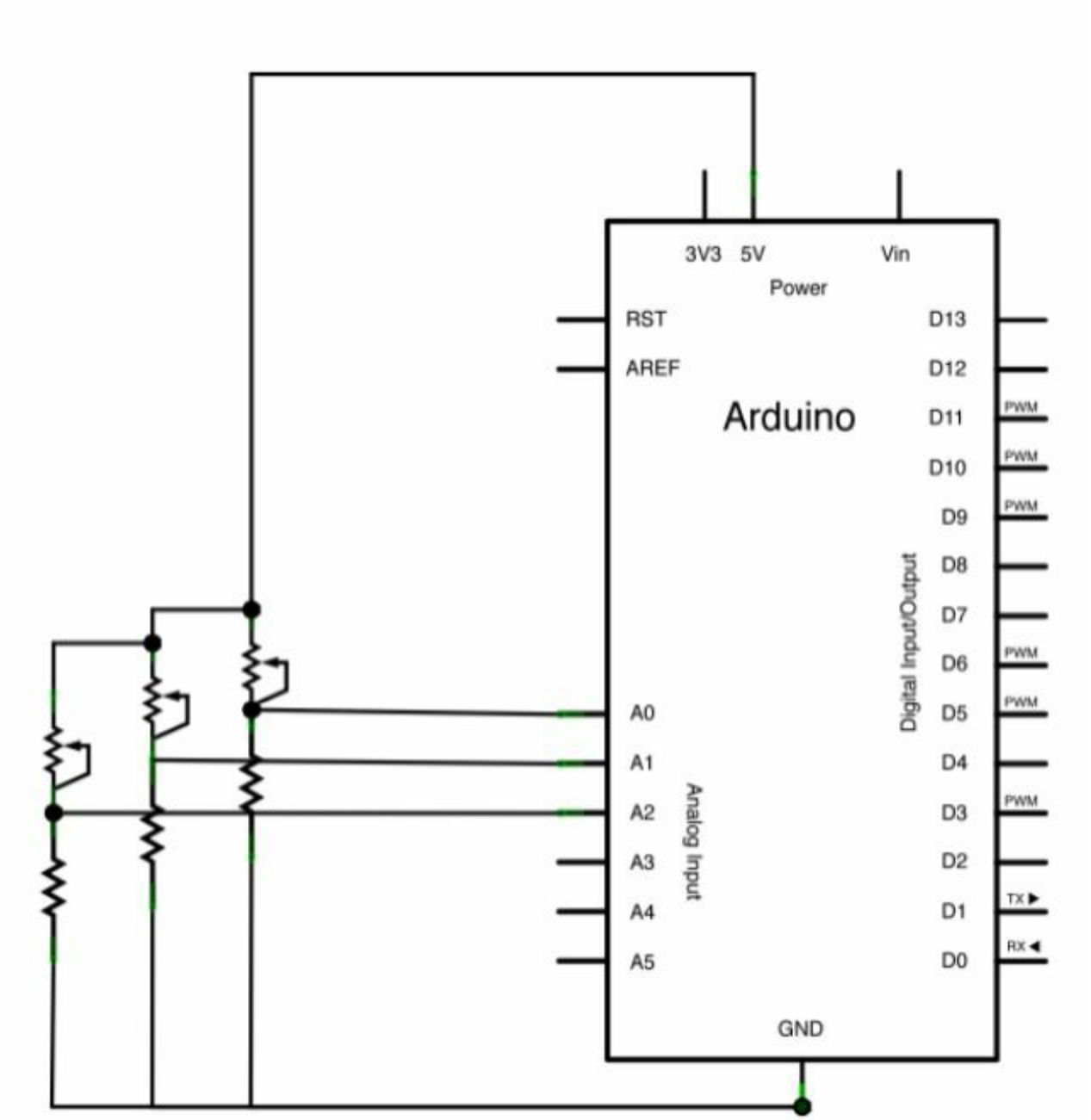
- Processor
- Max/MSP version 5

### Circuit

Here we have a circuit comprising of force-sensing resistors. Three sub-circuits that are voltage dividers generate analog voltages. Analog-Input pins 0, 1, and 2 are connected to the analog sensors. Voltage dividers have

resistors connected in series and according to their value, they have voltage in each.






# Code

Sensor values pass from Arduino to computer as decimal numbers that are ASCII-encoded. ASCII characters for each number from 0 to 9 are used to pass the number. For instance, you would get the ASCII bytes value 51, 52 and 56 for the number 348. (3 has the byte value of 51,…and so on).

```
/*
  This example reads three analog sensors (potentiometers are easiest)
  and sends their values serially. The Processing and Max/MSP programs at the
```

```
bottom
    take those three values and use them to change the background color of the
screen.

    The circuit:
    * potentiometers attached to analog inputs 0, 1, and 2

    http://www.arduino.cc/en/Tutorial/VirtualColorMixer

    created 2 Dec 2006
    by David A. Mellis
    modified 30 Aug 2011
    by Tom Igoe and Scott Fitzgerald

     This example code is in the public domain.
    */

  const int redPin = A0;      // sensor to control red color
  const int greenPin = A1;    // sensor to control green color
  const int bluePin = A2;     // sensor to control blue color

  void setup()
  {
   Serial.begin(9600);
  }

  void loop()
  {
   Serial.print(analogRead(redPin));
   Serial.print(",");
   Serial.print(analogRead(greenPin));
   Serial.print(",");
   Serial.println(analogRead(bluePin));
```

```
}

/* Processing code for this example

//  This example code is in the public domain.

 import processing.serial.*;

 float redValue = 0;        // red value
 float greenValue = 0;      // green value
 float blueValue = 0;       // blue value

 Serial myPort;

 void setup() {
 size(200, 200);

 // List all the available serial ports
 println(Serial.list());
 // I know that the first port in the serial list on my Mac
 // is always my Arduino, so I open Serial.list()[0].
 // Open whatever port is the one you're using.
 myPort = new Serial(this, Serial.list()[0], 9600);
 // don't generate a serialEvent() unless you get a newline character:
 myPort.bufferUntil('\n');
 }

 void draw() {
 // set the background color with the color values:
 background(redValue, greenValue, blueValue);
 }

 void serialEvent(Serial myPort) {
```

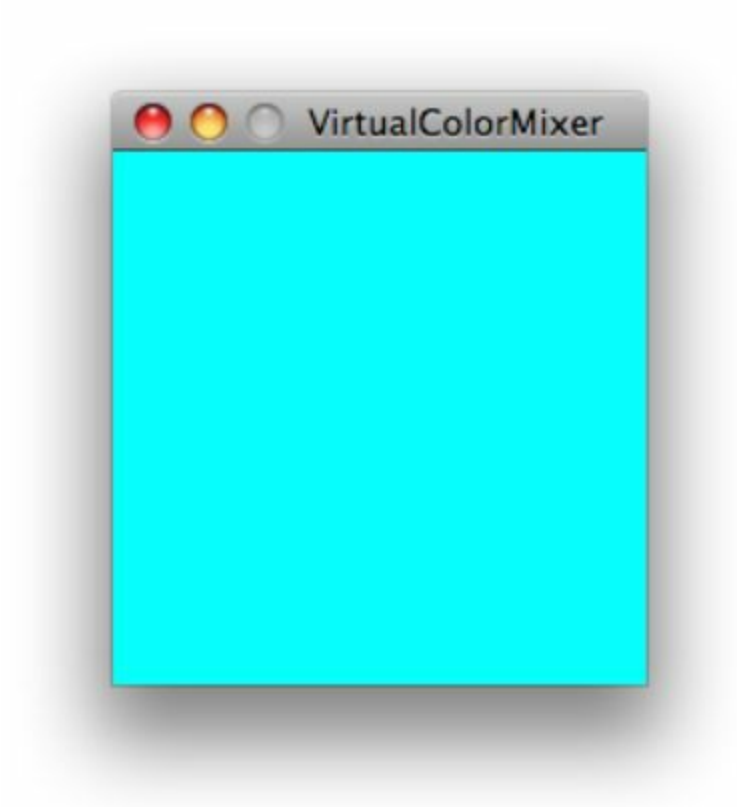
```
// get the ASCII string:
String inString = myPort.readStringUntil('\n');

if (inString != null) {
// trim off any whitespace:
inString = trim(inString);
// split the string on the commas and convert the
// resulting substrings into an integer array:
float[] colors = float(split(inString, ","));
// if the array has at least three elements, you know
// you got the whole thing.  Put the numbers in the
// color variables:
if (colors.length >=3) {
// map them to the range 0-255:
redValue = map(colors[0], 0, 1023, 0, 255);
greenValue = map(colors[1], 0, 1023, 0, 255);
blueValue = map(colors[2], 0, 1023, 0, 255);
}
}
}
 */

/* Max/MSP patch for this example

 ----------begin_max5_patcher----------
1512.3oc4Z00aaaCE8YmeED9ktB35xOjrj1aAsXX4g8xZQeYoXfVh1gqRjdT
TsIsn+2K+PJUovVVJ1VMdCAvxThV7bO7b48dIyWtXxzkxaYkSA+J3u.Sl7kK
lLwcK6MlT2dxzB5so4zRW2lJXeRt7elNy+HM6Vs61uDDzbOYkNmo02sg4euS
4BSede8S2P0o2vEq+aEKU66PPP7b3LPHDauPvyCmAvv4v6+M7L2XXF2WfCc
lURgVPKbCxzKUbZdySDUEbgABN.ia08R9mccGYGn66qGutNir27qWbg8iY+7
HDRx.Hjf+OPHCQgPdpQHoxhBlwB+QF4cbkthlCRk4REnfeKScs3ZwaugWBbj
.PS+.qDPAkZkgPlY5oPS4By2A5aTLFv9pounjsgpnZVF3x27pqtBrRpJnZaa
C3WxTkfUJYA.BzR.BhIy.ehquw7dSoJCsrlATLckR.nhLPNWvVwL+Vp1LHL.
```

*SjMG.tRaG7OxT5R2c8Hx9B8.wLCxVaGI6qnpj45Ug84kL+6YIM8CqUxJyycF*
*7bqsBRULGvwfWyRMyovElat7NvqoejaLm4f+fkmyKuVTHy3q3ldhB.WtQY6Z*
*x0BSOeSpTqA+FW+Yy3SyybH3sFy8p0RVCmaMpTyX6HdDZ2JsPbfSogbBMue*
*JLd6RMBdfRMzPjZvimuWIK2XgFA.ZmtfKoh0Sm88qc6OF4bDQ3P6kEtF6xej*
*.OkjD4H5OllyS+.3FlhY0so4xRlWqyrXErQpt+2rsnXgQNZHZgmMVzEofW7T*
*S4zORQtgIdDbRHrObRzSMNofUVZVcbKbhQZrSOo934TqRHIN2ncr7BF8TKR*
*tHDqL.PejLRRPKMR.pKFAkbtDa+UOvsYsIFH0DYsTCjqZ66T1CmGeDILLpSm*
*myk0SdkOKh5LUr4GbWwRYdW7fm.BvDmzHnSdH3biGpSbxxDNJoGDAD1ChH*
*I0DaloOTBLvkO7zPs5HJnKNoGAXbol5eytUhfyiSfnjE1uAq+Fp0a+wygGwR*
*q3ZI8.psJpkpJnyPzwmXBj7Sh.+bNvVZxlcKAm0OYHIxcIjzEKdRChgO5UMf*
*LkMPNN0MfiS7Ev6TYQct.F5IWcCZ4504rGsiVswGWWSYyma01QcZgmL+f+sf*
*oU18Hn6o6dXkMkFF14TL9rIAWE+6wvGV.p.TPqz3HK5L+VxYxl4UmBKEjr.B*
*6zinuKI3C+D2Y7azIM6N7QL6t+jQyZxymK1ToAKqVsxjlGyjz2c1kTK3180h*
*kJEYkacWpv6lyp2VJTjWK47wHA6fyBOWxH9pUf6jUtZkLpNKW.9EeUBH3ymY*
*XSQlaqGrkQMGzp20adYSmIOGjIABo1xZyAWJtCX9tg6+HMuhMCPyx76ao+U*
*UxmzUE79H8d2ZB1m1ztbnOa1mGeAq0awyK8a9UqBUc6pZolpzurTK232e5gp*
*aInVw8QIIcpaiNSJfY4Z+92Cs+Mc+mgg2cEsvGlLY6V+1kMuioxnB5VM+fsY*
*9vSu4WI1PMBGXye6KXvNuzmZTh7U9h5j6vvASdngPdgOFxycNL6ia1axUMm1*
*JIzebXcQCn3SKMf+4QCMmOZung+6xBCPLfwO8ngcEI52YJ1y7mx3CN9xKUY*
*bg7Y1yXjlKW6SrZnguQdsSfOSSDItqv2jwJFjavc1vO7OigyBr2+gDYorRk1*
*HXZpVFfu2FxXkZtfp4RQqNkX5y2sya3YYL2iavWAOaizH+pw.Ibg8f1I9h3Z*
*2B79sNeOHvBOtfEalWsvyu0KMf015.AaROvZ7vv5AhnndfHLbTgjcCK1KlHv*
*gOk5B26OqrXjcJ005.QqCHn8fVTxnxfj93SfQiJlv8YV0VT9fVUwOOhSV3uD*
*eeqCUClbBPa.j3vWDoMZssNTzRNEnE6gYPXazZaMF921syaLWyAeBXvCESA8*
*ASi6Zyw8.RQi65J8ZsNx3ho93OhGWENtWpowepae4YhCFeLErOLENtXJrOSc*
*iadi39rf4hwc8xdhHz3gn3dBI7iDRlFe8huAfIZhq*
*----------end_max5_patcher----------*

*/

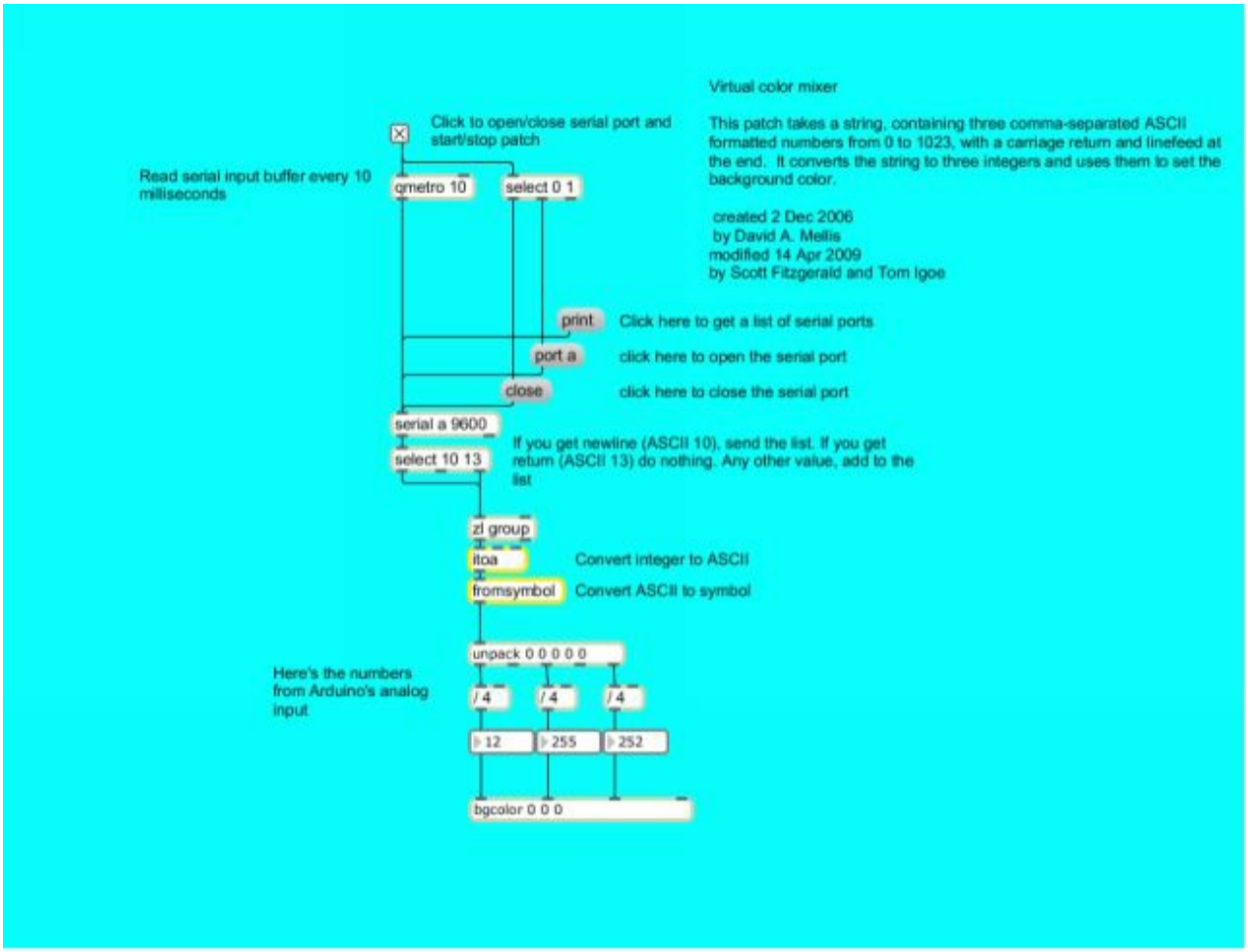
# Processing Code

Use code sample above to copy your Processing sketch. When you change the analog sensor values, the background color will keep changing.




## Max Code

You Max Patch will resemble this. Using code sample above, copy its text and paste to new Max window.




# C) DIMMER

***(All images developed by fritzing.org)***

Here in this project you will learn what to do to control how bright the LED shines. You send data as individual bytes in the range 0 to 255. When the Arduino reads these values, it sets the LED brightness. Arduino takes values from any software that can communicate with the Arduino through serial port of the computer. Here we see how it done using Max/MSP version 5 and Processing.

## Hardware and Circuit Required

- Arduino Board

- 220 ohm resistor
- LED

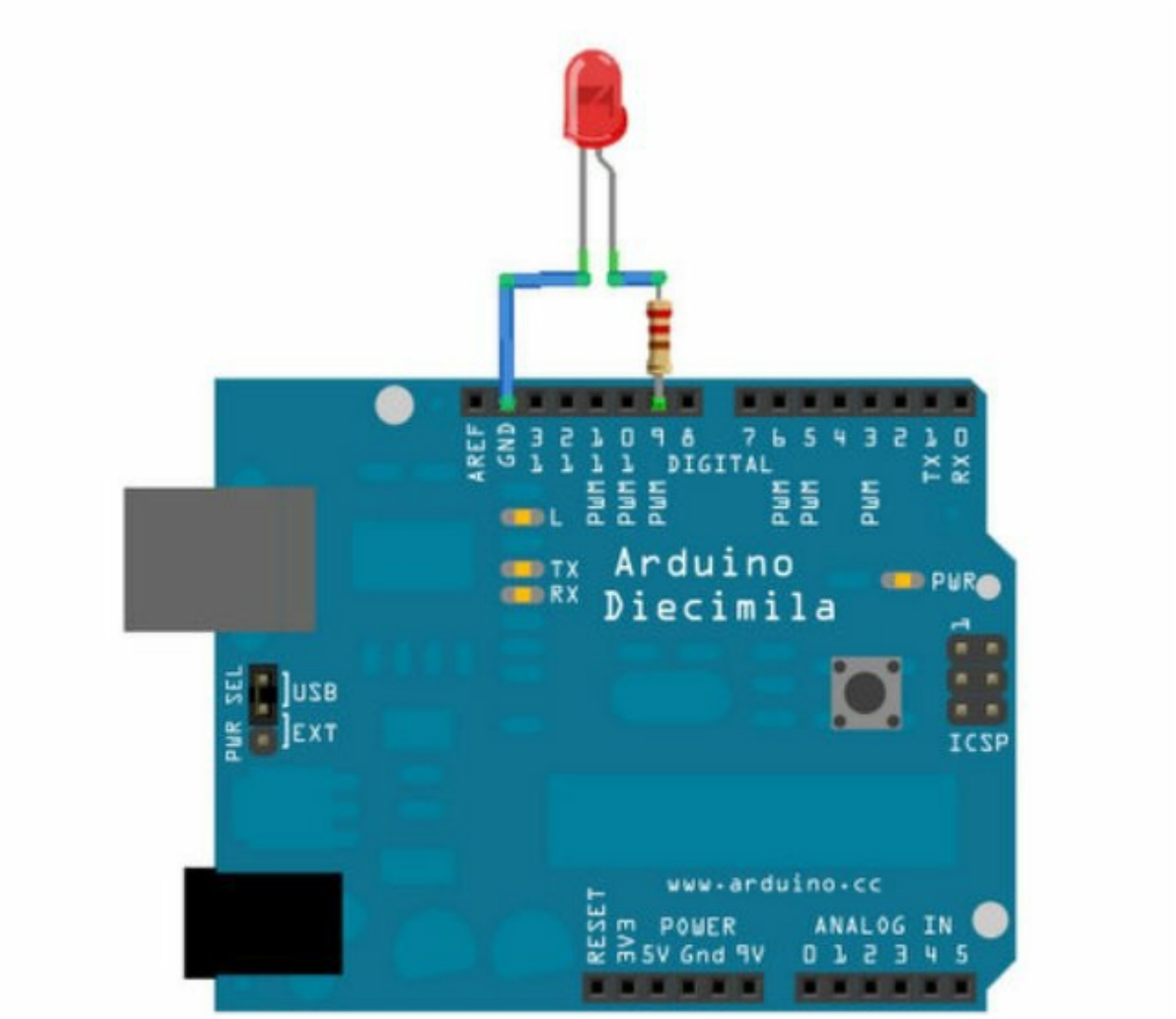
## Software Required

Max/MSP version 5 or

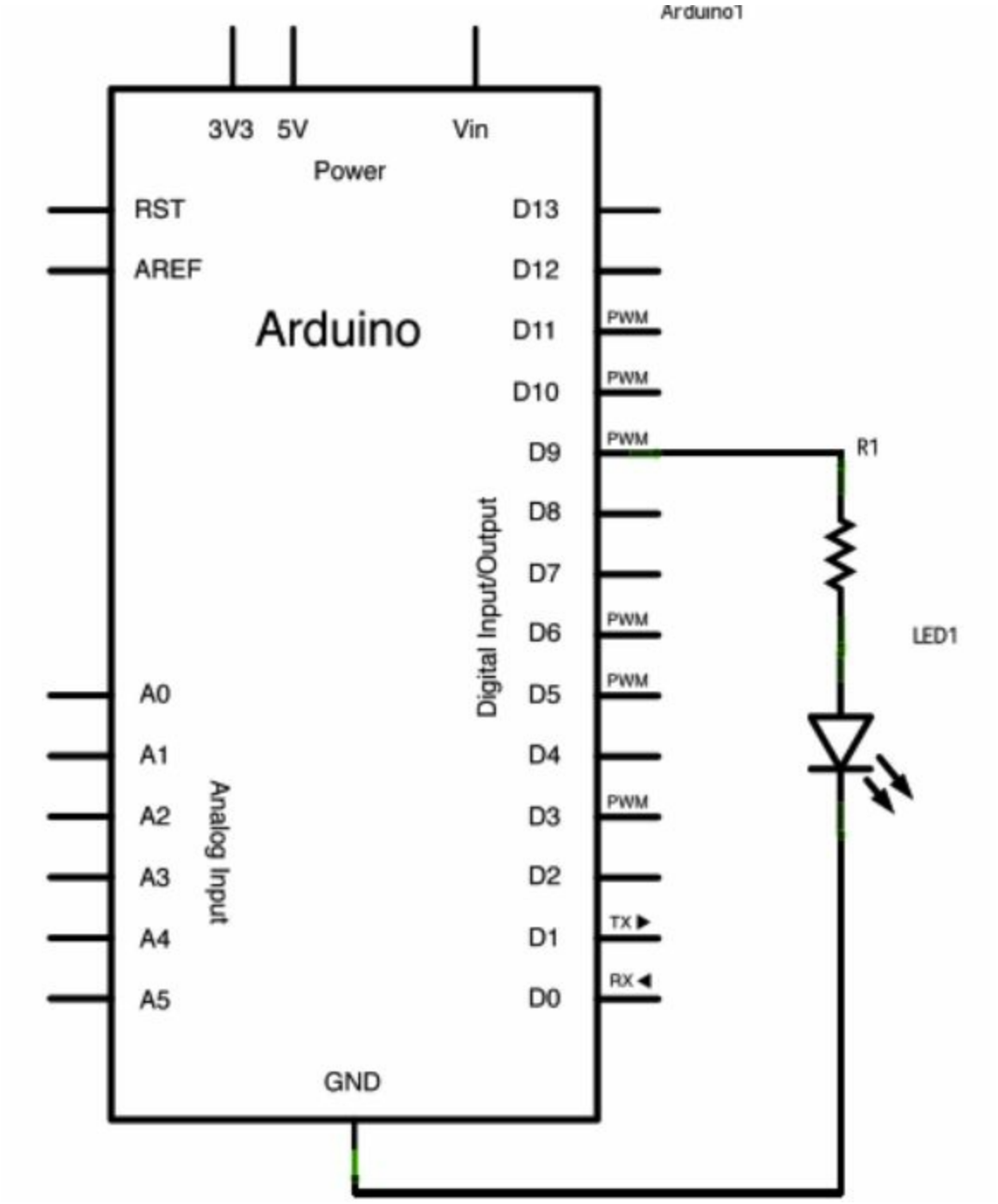
Processing

## Circuit

Using a suitable resistor connect pin 9 to an LED. For most cases, a 220 Ω to 330 Ω resistor will do fine.



# Schematic




# Code

```
/*
  Dimmer

  Demonstrates the sending data from the computer to the Arduino board,
  in this case to control the brightness of an LED.  The data is sent
  in individual bytes, each of which ranges from 0 to 255.  Arduino
```

```
 reads these bytes and uses them to set the brightness of the LED.

 The circuit:
 LED attached from digital pin 9 to ground.
 Serial connection to Processing, Max/MSP, or another serial application

 created 2006
 by David A. Mellis
 modified 30 Aug 2011
 by Tom Igoe and Scott Fitzgerald

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/Dimmer

 */

const int ledPin = 9;      // the pin that the LED is attached to

void setup()
{
  // initialize the serial communication:
  Serial.begin(9600);
  // initialize the ledPin as an output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  byte brightness;

  // check if data has been sent from the computer:
  if (Serial.available()) {
    // read the most recent byte (which will be from 0 to 255):
```

```
    brightness = Serial.read();
    // set the brightness of the LED:
    analogWrite(ledPin, brightness);
  }
}

/* Processing code for this example
// Dimmer - sends bytes over a serial port
// by David A. Mellis
//This example code is in the public domain.

import processing.serial.*;
Serial port;

void setup() {
size(256, 150);

println("Available serial ports:");
println(Serial.list());

// Uses the first port in this list (number 0).  Change this to
// select the port corresponding to your Arduino board.  The last
// parameter (e.g. 9600) is the speed of the communication.  It
// has to correspond to the value passed to Serial.begin() in your
// Arduino sketch.
port = new Serial(this, Serial.list()[0], 9600);

// If you know the name of the port used by the Arduino board, you
// can specify it directly like this.
//port = new Serial(this, "COM1", 9600);
}

void draw() {
```

```
// draw a gradient from black to white
for (int i = 0; i < 256; i++) {
stroke(i);
line(i, 0, i, 150);
}

// write the current X-position of the mouse to the serial port as
// a single byte
port.write(mouseX);
}
*/

/* Max/MSP v5 patch for this example

----------begin_max5_patcher----------
1008.3ocuXszaiaCD9r8uhA5rqAeHIa0aAMaAVf1S6hdoYQAsDiL6JQZHQ2M
YWr+2KeX4vjnjXKKkKhhiGQ9MeyCNz+X9rnMp63sQvuB+MLa1OlOalSjUvrC
ymEUytKuh05TKJWUWyk5nE9eSyuS6jesvHu4F4MxOuUzB6X57sPKWVzBLXiI
xZtGj6q2vafaaT0.BzJfjj.p8ZPukazsQvpfcpFs8mXR3plh8BoBxURIOWyK
rxspZ0YI.eTCEh5Vqp+wGtFXZMKe6CZc3yWZwTdCmYW.BBkdiby8v0r+ST.W
sD9SdUkn8FYspPbqvnBNFtZWiUyLmleJWo0vuKzeuj2vpJLaWA7YiE7wREui
FpDFDp1KcbAFcP5sJoVxp4NB5Jq40ougIDxJt1wo3GDZHiNocKhiIExx+owv
AdOEAksDs.RRrOoww1Arc.9RvN2J9tamwjkcqknvAE0l+8WnjHqreNet8whK
z6mukIK4d+Xknv3jstvJs8EirMMhxsZIusET25jXbX8xczIl5xPVxhPcTGFu
xNDu9rXtUCg37g9Q8Yc+EuofIYmg8QdkPCrOnXsaHwYs3rWx9PGsO+pqueG2
uNQBqWFh1X7qQG+3.VHcHrfO1nyR2TlqpTM9MDsLKNCQVz6KO.+Sfc5j1Yk
jzkn2jwNDRP7LVb3d9LtoWBAOnvB92Le6yRmZ4UF7YpQhiFi7A5Ka8zXhKdA
4r9TRGG7V4COiSbAJKdXrWNhhF0hNUh7uBa4Mba0l7JUK+omjDMwkSn95Iz
TOwkdp7W.oPRmNRQsiKeu4j3CkfVgt.NYPEYqMGvvJ48vIlPiyzrIuZskWIS
xGJPcmPiWOfLodybH3wjPbMYwlbFIMNHPHFOtLBNaLSa9sGk1TxMzCX5KT
WIH2ocxSdngM0QPqFRxyPHFsprrhGc9Gy9xoBjz0NWdR2yW9DUa2F85jG2v9
FgTO4Q8qiC7fzzQNpmNpsY3BrYPVJBMJQ1uVmoItRhw9NrVGO3NMNzYZ+zS
3WTvTOnUydG5kHMKLqAOjTe7fN2bGSxOZDkMrBrGQ9J1gONBEy0k4gVo8q
```

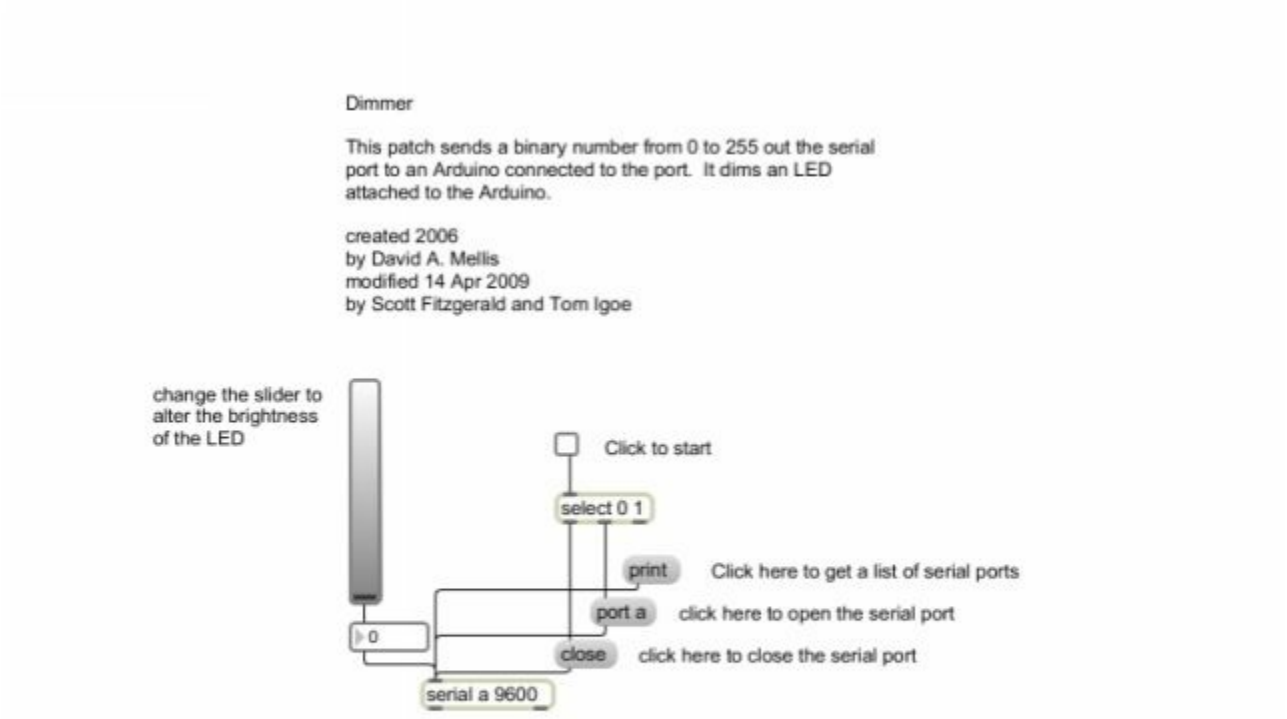
*cxmfxVihWz6a3yqY9NazzUYkua9UnynadOtogW.JfsVGRVNEbWF8I+eHtcwJ*
*+wLXqZeSdWLo+FQF6731Tva0BISKTx.cLwmgJsUTTvkg1YsnXmxDge.CDR7x*
*D6YmX6fMznaF7kdczmJXwm.XSOOrdoHhNA7GMiZYLZZR.+4lconMaJP6JOZ8*
*ftCs1YWHZI3o.sIXezX5ihMSuXzZtk3ai1mXRSczoCS32hAydeyXNEu5SHyS*
*xqZqbd3ZLdera1iPqYxOm++v7SUSz*
*----------end_max5_patcher----------*
**/*

# Processing Code

Use the connection shown in code sample. This will send the bytes that dim the LED. Connection is through serial port to Arduino.

# Max Code

The Max/MSP patch is shown of the above code sample. You have to copy and paste it to new window.



# III) EXAMPLES FROM ESPLORA LIBRARY

## LED SHOW

In this tutorial, we learn how to get the values directly from joystick. The Serial monitor displays output in terms of RGB LED color.

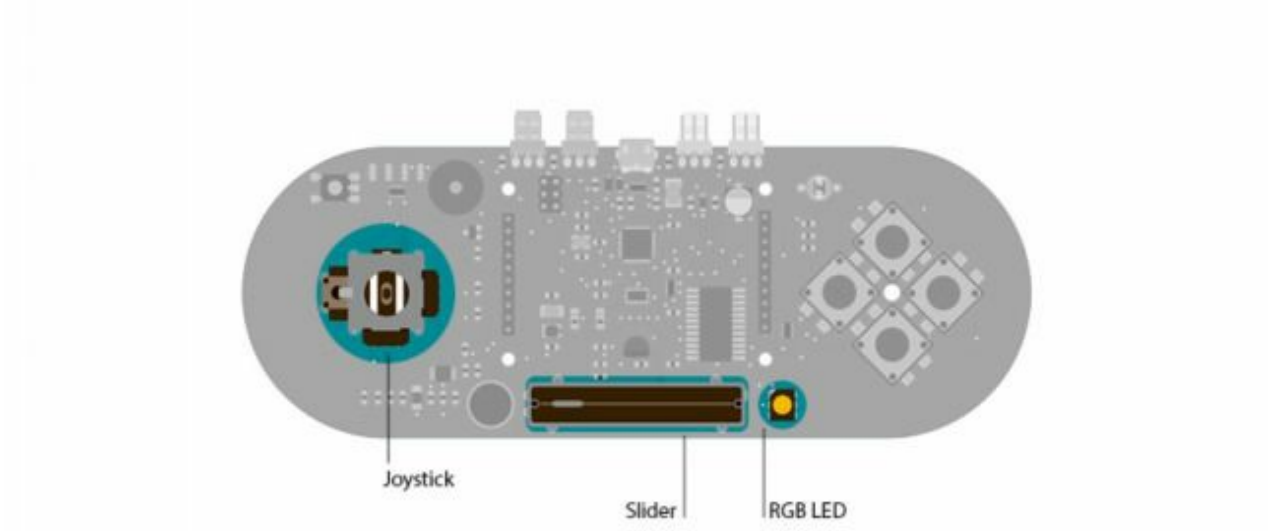
You can move the joystick along two perpendicular axes X or Y. Each one of the axes controls one color that the RGB LED displays – X- axis depicts red while Y – axis depicts green. Blue element will have its control by relative position of linear potentiometer.

### Hardware and Circuit Required

Arduino Esplora

### Circuit

You will not need any circuit for this example.




### Code

The brightness of the RGB LED light may be controlled individually through the

Esplora Library functions. The light consists of three colors Red, Green, and Blue.

writeRed ();
writeBlue ();
writeGreen ();

You may also control all of the colors with one single command using writeRGB (); Values generated will differ when the joystick moves. You can also shift the position of the linear potentiometer to create new outputs. One output is by the serial monitor while the other you see as the RGB LED light.

```
/*
  Esplora LED Show

  Makes the RGB LED bright and glow as the joystick or the
  slider are moved.

  Created on 22 November 2012
  By Enrico Gueli <enrico.gueli@gmail.com>
  Modified 22 Dec 2012
  by Tom Igoe
*/
#include <Esplora.h>

void setup() {
  // initialize the serial communication:
  Serial.begin(9600);
}

void loop() {
  // read the sensors into variables:
  int xAxis = Esplora.readJoystickX();
  int yAxis = Esplora.readJoystickY();
  int slider = Esplora.readSlider();
```

```
  // convert the sensor readings to light levels:
  byte red   = map(xAxis, -512, 512, 0, 255);
  byte green = map(yAxis, -512, 512, 0, 255);
  byte blue  = slider / 4;

  // print the light levels:
  Serial.print(red);
  Serial.print(' ');
  Serial.print(green);
  Serial.print(' ');
  Serial.println(blue);

  // write the light levels to the LED.
  Esplora.writeRGB(red, green, blue);

  // add a delay to keep the LED from flickering:
  delay(10);
}
```

# IV) EXAMPLES FROM ROBOT LIBRARY

## A) FOLLOWING THE LINE

This is a fun filled project where you can race your robot against others. The means of controlling your robot is by a line drawn on a large paper. Your robot will search for the line and then it will follow it. Think of the fun you can have when you have more robots!

### Hardware and Circuit required

Arduino Robot

Large white sheet of paper

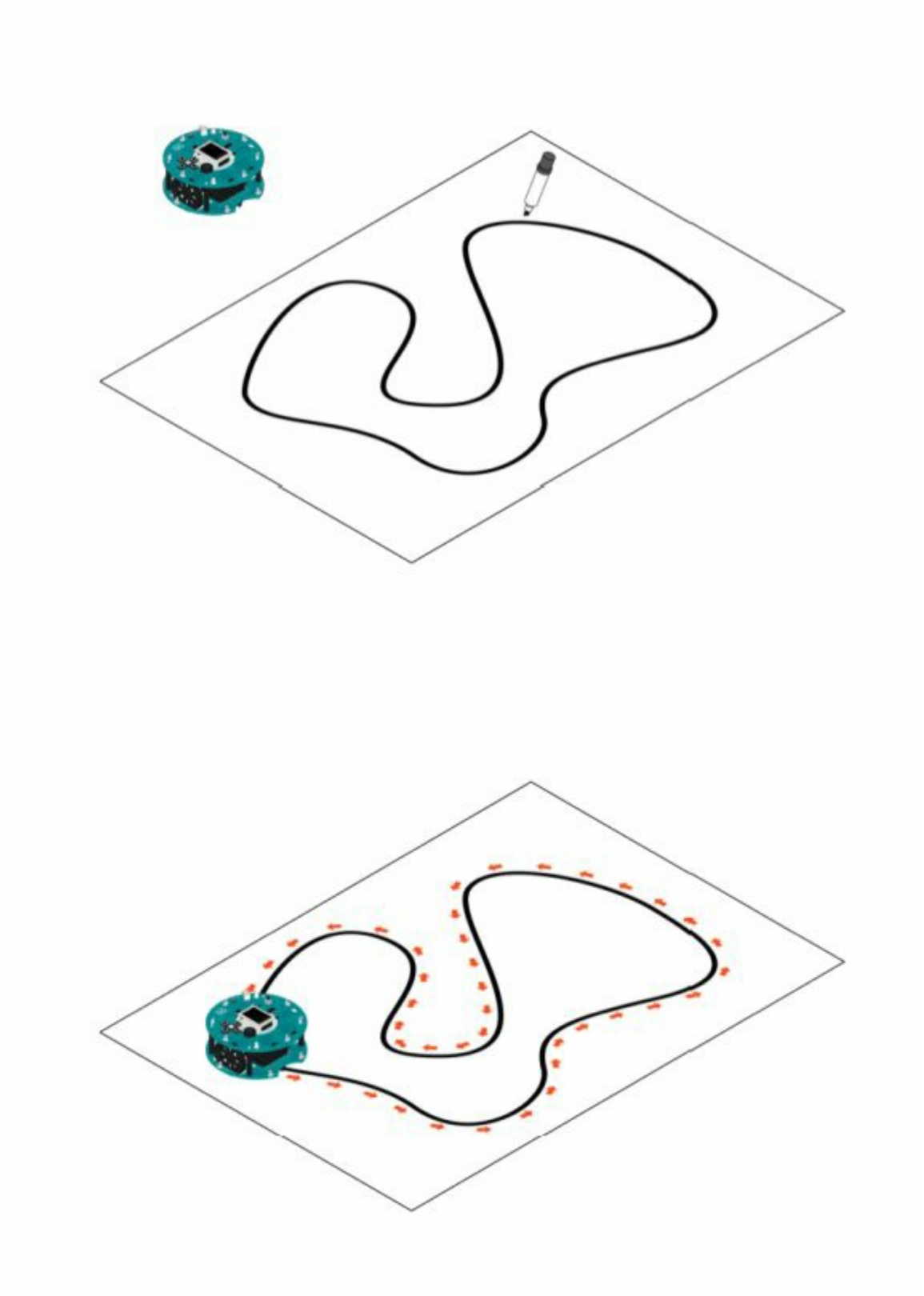
Black marker pen

Tape

# Procedure

1. Make tracks for the robot. These tracks will go winding on a piece of paper. They have to be 1 inch (about 2 ½ cm) thick. The line must not intersect and the end of the line must join the starting point. Join several pieces of paper if the track area is not big enough.
2. Not fix the paper to the ground with some tape. This will prevent it from slipping when the robot runs around on it. Try to keep the surface light in color.
3. Upload the code, unplug the USB and switch on power.
4. Put the robot on the ground such that it is facing the line.
5. For the first few trials (starting screen), the robot will make the calibrations required and then it will orient itself to detect the line.
6. Once the robot detects the line, it will carry on following it without stop. For more details on calibration check up lineFollowConfig.

# Try to run it



*Now your robot is ready to race!*

# Code

*/* Robot Line Follow*

*This sketch demonstrates the line following capabilities of the Arduino Robot. On the floor, place some black electrical tape along the path you wish the robot to follow. To indicate a stopping point, place another piece of tape perpendicular to the path.*

*Circuit:*
** Arduino Robot*

*created 1 May 2013*
*by X. Yang*
*modified 12 May 2013*
*by D. Cuartielles*

*This example is in the public domain*
**/*

```
#include <ArduinoRobot.h> // include the robot library
```

```
#include <Wire.h>
#include <SPI.h>

long timerOrigin; // used for counting elapsed time

void setup() {
  // initialize the Robot, SD card, display, and speaker
  Robot.begin();
  Robot.beginTFT();
  Robot.beginSD();
  Robot.beginSpeaker();

  // show the logots on the TFT screen
  Robot.displayLogos();

  Robot.drawBMP("lf.bmp", 0, 0); // display background image

  Robot.playFile("chase.sqm");  // play a song from the SD card

  // add the instructions
  Robot.text("Line Following\n\n place the robot on\n the track and \n see it run",
5, 5);
  Robot.text("Press the middle\n button to start...", 5, 61);
  Robot.waitContinue();

  // These are some general values that work for line following
  // uncomment one or the other to see the different behaviors of the robot
  //Robot.lineFollowConfig(14, 9, 50, 10);
  Robot.lineFollowConfig(11, 7, 60, 5);



  //set the motor board into line-follow mode
  Robot.setMode(MODE_LINE_FOLLOW);
```

```
  // start
  Robot.fill(255, 255, 255);
  Robot.stroke(255, 255, 255);
  Robot.rect(0, 0, 128, 80); // erase the previous text
  Robot.stroke(0, 0, 0);
  Robot.text("Start", 5, 5);

  Robot.stroke(0, 0, 0); // choose color for the text
  Robot.text("Time passed:", 5, 21); // write some text to the screen

  timerOrigin = millis(); // keep track of the elapsed time

  while (!Robot.isActionDone()) { //wait for the finish signal
    Robot.debugPrint(millis() - timerOrigin, 5, 29); // show how much time has passed
  }

  Robot.stroke(0, 0, 0);
  Robot.text("Done!", 5, 45);
}
void loop() {
  //nothing here, the program only runs once. Reset the robot
  //to do it again!
}
```

## Working Principle

Robot senses the ground beneath it with five sensors. Light reflects back to the sensors. In order to detect the line, the wheels of the robot must turn. This it does when it does not detect any line and all five sensors produce white light.

## Stopping

When the robot detects black in all five sensors, it will stop. This is the method to stop the robot.

# B) WRITING A LIBRARY FOR ARDUINO

In this tutorial, we will see what goes into the creation of an Arduino library. First, we write a sketch that will flash Morse code and then use the function to create the library. This way, people can easily use your code and update it. It will help the library to improve. You will get more information in API Style Guide.

Let us begin with the simple sketch that does the Morse code.

```
int pin = 13;

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

void loop()
{
  dot(); dot(); dot();
  dash(); dash(); dash();
  dot(); dot(); dot();
  delay(3000);
}

void dot()
{
  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
```

```
  delay(250);
}

void dash()
{
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(250);
}
```

The code above when run will flash the distress code SOS through pin 13. The sketch does not have any significantly different parts other than the functions dot () and dash (), which of course do the actual blinking. Now, the functions use the ledPin () variable that will tell functions, which pin to use. Finally, one has the pinMode () a call to initialize pin as output. You have started your library from the sketch!

Every library has at least two files – the header file that has extension .h and source file having extension .cpp. Header file comprises of definitions stating everything that the library contains. Source file comprises of the actual code. Let us name our library *Morse*. This will make our header file Morse.h…should be interesting to see its contents. You will find it difficult to understand it all at the beginning, but it will get easier as you go along.

Header file core will be made of one line representation for every function in the library. It is represented in class together with any variables you might need.

```
class Morse
{
  public:
    Morse(int pin);
    void dot();
    void dash();
```

```
  private:
    int _pin;
};
```

To understand class, you need to think of it as function collection alongside variables where you can easily access it. When this agglomeration can be accessed by anyone using the library, it is termed *public*. When it is private, it can only be accessed from within that class alone.

Special function constructor exists within a class to help create instance of the class. Name of the constructor will remain the same as the class name but it will not have return value.

To complete the header file, you will also need #include statement in order to let you access constants and standard types in Arduino language. In sketches, this is normally included but it is not in libraries. This is how you write it.

```
#include "Arduino.h"
```

Lastly, you wrap your header file in the construct:

```
#ifndef Morse_h
#define Morse_h

// the #include statement and code go here...

#endif
```

This helps you avert problems like repetition as it happens with two #includes. You round up your library by putting a comment on top of it. This will tell what is present, the library name, name of author, the date and license.

The complete header file would be like this.

```
/*
  Morse.h - Library for flashing Morse code.
  Created by David A. Mellis, November 2, 2007.
  Released into the public domain.
*/
#ifndef Morse_h
#define Morse_h

#include "Arduino.h"

class Morse
{
  public:
    Morse(int pin);
    void dot();
    void dash();
  private:
    int _pin;
};

#endif
```

Let us now see what goes into the making of the source file – the Morse.cpp. You begin with include statements that give the code you write access to Arduino standard functions as defined inside in the library.

```
#include "Arduino.h"
#include "Morse.h"
```

After this, you make constructors. This lets users choose specific instances of classes. They do this by choosing a pin that is configured as an output.

```
Morse::Morse (int pin)
{
  pinMode (pin, OUTPUT);
  _pin = pin;
}
```

In these lines of code, you will notice that there is a slightly different convention of naming. One is the :: operator, which is the scope operator. It gives a global visibility to the chosen variable. Another change you see is the underscore that is assigned before the pin class. Adding underscore helps you name a class or variable without breaking the naming convention. You get a clearer view of your code and help you group all those classes you want with underscore so that you can work easier. For instance, you may need some group to be private. You can add underscore in front of their class names. Now, proceed with the code for the library.

```
void Morse::dot()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(1000);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

You then round off your code by writing the comment on top. It helps you understand what you have written.

```
/*
  Morse.cpp - Library for flashing Morse code.
  Created by David A. Mellis, November 2, 2007.
  Released into the public domain.
*/

#include "Arduino.h"
#include "Morse.h"

Morse::Morse(int pin)
{
  pinMode(pin, OUTPUT);
  _pin = pin;
}

void Morse::dot()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(1000);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

Now that you have written the code, you must know how to access and use it.

Begin by making a Morse directory in the library subdirectory in your sketch. Copy Morse.h along with Morse.cpp files and do the Arduino environment launch. It means just open the Arduino and then go to Sketch>Import Library menu. Here you will find Morse inside.

```
#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
  morse.dot(); morse.dot(); morse.dot();
  morse.dash(); morse.dash(); morse.dash();
  morse.dot(); morse.dot(); morse.dot();
  delay(3000);
}
```

Let us see what differences exist after we made the duplication. First thing to notice is the #include statement at the top. Doing this includes this sketch in code that will be sent to the board and make the library accessible to the sketch. You do not have to include library. Delete the #include statement for library to save space. Next, you can see a new instance of Morse class has been created named morse.

```
Morse morse (13);
```

During the execution process, the constructor class is called even before setup () function and it will be given an argument, 13 in this example. The setup () will not have anything because the call taking place to the pinMode () occurs within the library.

In a similar fashion, we need to prefix *morse* in order to call the functions dot ()

and dash (). For each instance of Morse class, you can add the pin and store it in the private variable _pin. When we call some particular instance of any function, we are specifying exactly what the variable should be for that instance for the duration of the call. So, if we had:

```
Morse morse(13);
Morse morse2(12);
```

a call to morse2.dot () would have the value 12. You might notice too that Arduino does not automatically recognize your definitions in the library and update the environment and color. In order to do this, you make use of keywords.txt file that you create within that Morse directory.

```
Morse   KEYWORD1
dash    KEYWORD2
dot     KEYWORD2
```

Each class you mention is written in this way. Name of the class and the keyword you assign to it. Then use a tab without space to go to next class. Here again you write the next keyword and so on. KEYWORD1 will all have orange color while the KEYWOR2 will take up brown color. Restart Arduino environment and the new keywords will take effect.

Now if you plan to include working examples within your sketch, do place example directory within Morse directory. Now, move the sketch (say SOS) to the examples directory. It is easy to negotiate to this by pulling down Sketch>Show Sketch Folder.

Again restart Arduino and you will see Library-Morse inside File>Sketchbook>Examples. You can use comments to explain what your example does. You can download Arduino for your operating system.

# V) EXAMPLES FROM SENSORS

# A) KNOCK

This tutorial will tell you how to use a Piezo element for detecting knock on any surface – a table, the door or the top of a device. As you might have guessed the Piezo would be any electronic device, you use for detecting the knock. How does it do this? The Piezo element possesses the property of generating electricity when you vibrate it…meaning you knock it. The output is a sound with a particular tone. Therefore, you can use the piezo element to both generate and detect tones.

We use analog-to-digital conversion (ADC) process that outputs numerals from 0 —1023 for voltage ranges between 0V and 5V. The sketch reads the voltage with analogRead () function. In case the output is greater than the prescribed threshold, Arduino sends a string "Knock!" which you can see by opening serial monitor.

## Hardware and Circuit Required

Arduino board

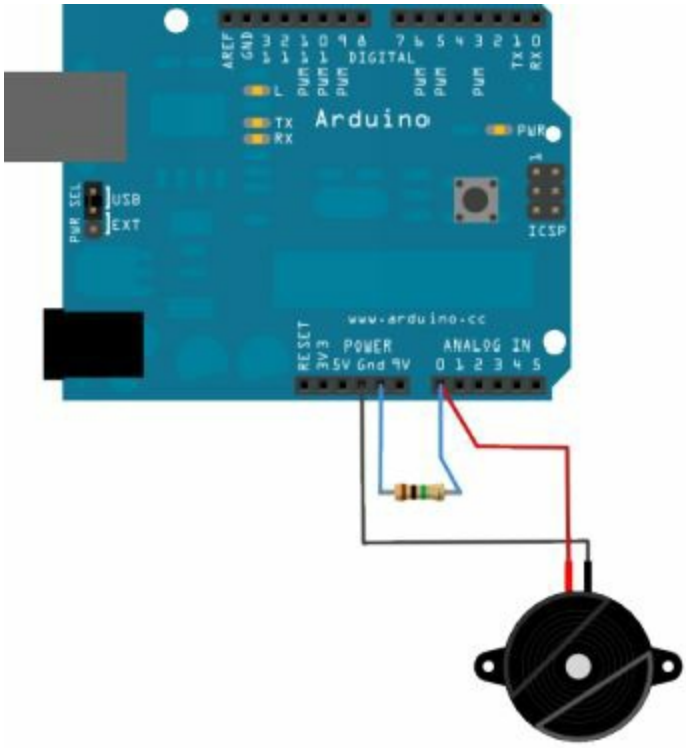
Megohm resistor – 1

Piezo-electric disc –1

Any solid surface (to knock)

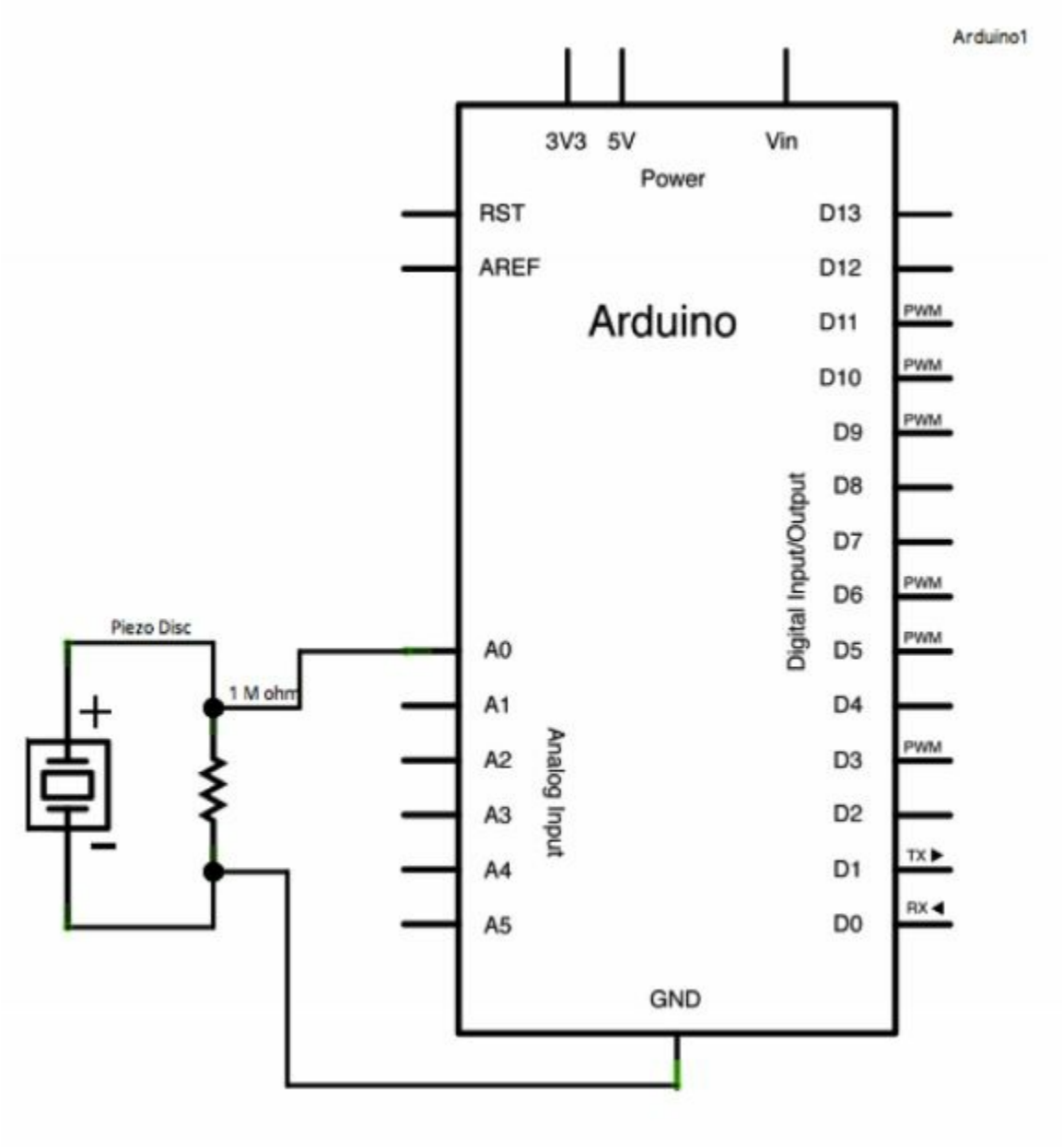
## Circuit

Piezos have polarized structures. This allows current to pass through them in one direction only. Make a connection with the black wire (lower voltage) to the ground and the higher voltage red wire to the 0 analog pin. In order to limit voltage/current produced by piezo, use a 1-megohm resistor in parallel. If possible, get piezo elements that do not have plastic coverings. Now tape the bare metallic body to the surface you want to take the reading from.




*(Fritzing develops these images.)*

# Schematic




*1-Megohm resistor connected to piezo at pin 0*

# Code

User sets a threshold and studies data generated by the piezo element. One can change the threshold value to modify the sensitivity of the element.

*/* Knock Sensor*

*You can use this sketch to detect knocking sound with a piezo element. Reading from the analog pin compares results that user predetermines for some threshold. When the result shows bigger value, 'knock' will be written to serial port and the pin 13 LED will be toggled.*

```
 Details of circuit:
        In analog at 0 piezo positive connection is attached
        Ground attaches to – piezo connection
        1-megohm connected between ground and 0 in analog

  http://www.arduino.cc/en/Tutorial/Knock

  created 25 Mar 2007
  by David Cuartielles <http://www.0j0.org>
  modified 30 Aug 2011
  by Tom Igoe

  This example code is in the public domain.

 */


// these constants won't change:
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


// these variables will change:
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 OUTPUT
 Serial.begin(9600);       // use the serial port
}
```

```
void loop() {
  // read the sensor and store it in the variable sensorReading:
  sensorReading = analogRead(knockSensor);

  // if the sensor reading is greater than the threshold:
  if (sensorReading >= threshold) {
    // toggle the status of the ledPin:
    ledState = !ledState;
    // update the LED pin itself:
    digitalWrite(ledPin, ledState);
    // send the string "Knock!" back to the computer, followed by newline
    Serial.println("Knock!");
  }
  delay(100);  // delay to avoid overloading the serial port buffer
```

---

# VI) EXAMPLES FROM STRINGS

## A) STRING ADDITION OPERATOR

Concatenation is defined as the process of combining a character or several characters or an ASCII representation (constant or variable) to a string and outputting the result. This resultant string is longer than either the length of characters used or the original string. To add Strings you use the + operator.

```
// adding string with a constant integer:
  stringThird =  stringFirst + 678;

  // adding string with a constant long integer:
  stringThird = stringFirst + 975312468;

  // adding string with a constant character:
```

```
stringThird =  stringFirst + 'D';

// adding string with a constant string:
stringThird =  stringFirst +  "gfd";

// adding together two Strings:
stringThird = stringFirst + stringSecond;
```

Another thing you can do is to add the result of some operation to some string. The only restriction is that the data type returned by the string must be valid.

```
stringThird = stringFirst + millis();
```

Here millis () returns long integer, which may be added to any string. You can write the code like this too.

```
stringThird = stringFirst + analogRead(A0);
```

In the last, analogRead () returns integer that can be added to string. This kind of concatenation proves useful in situations where you need to display more than one value as in LCD displays for advertising products. They need to display the price and details including the name and packaging weight.

```
int sensorValue = analogRead(A0);
String stringFirst = "Sensor value: ";
String stringThird = stringFirst + sensorValue;
Serial.println(stringThird);
```

This will give you some sensor value output. However, if you run the code below, it will not give you any valid output.

```
int sensorValue = analogRead(A0);
```

```
String stringThird = "Sensor value: " + sensorValue;
Serial.println(stringThird);
```

The result will be unpredictable since you have not initially given value to stringThird. Check another kind of improper initialization here.

```
Serial.println("I want " + analogRead(A0) + " donuts");
```

The operator precedence is wrong and so the code will not compile. The following example will compile but the results will not be what are expected.

```
int sensorValue = analogRead(A0);
String stringThird = "I want " + sensorValue;
Serial.println (stringThird  + " donuts");
```

The reason for the error is again clear. The stringThird never was initialized.

# Hardware and Circuit required

Arduino board

You do not require any other circuit for this example.

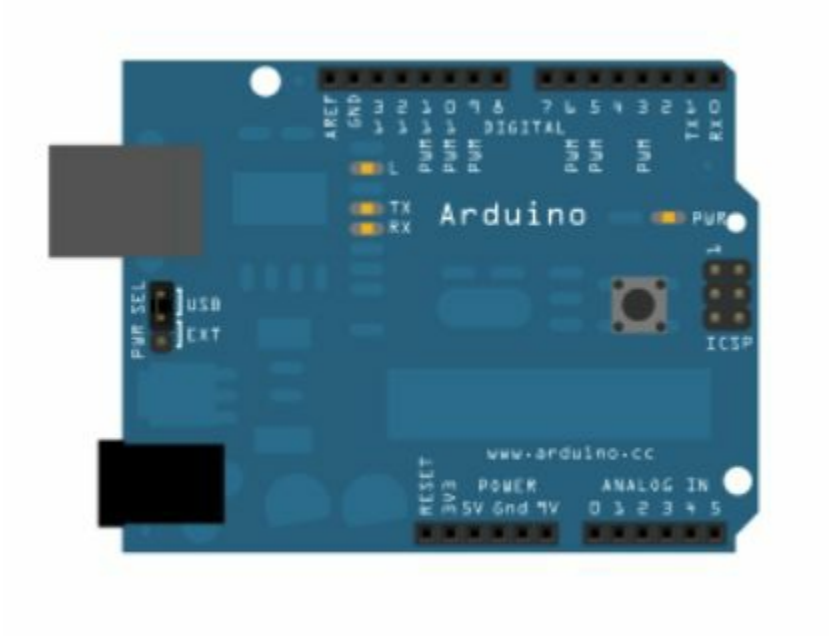



*Image developed by Fritzing*

# Code

Here is the code for several examples of concatenation types.

```
/*
 Adding Strings together

 Examples of how to add strings together
 You can also add several different data types to string, as shown here:

 created 27 July 2010
 modified 2 Apr 2012
 by Tom Igoe

 http://arduino.cc/en/Tutorial/StringAdditionOperator

 This example code is in the public domain.
 */

// declare three strings:
String stringOne, stringTwo, stringThree;

void setup() {
  // initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  stringOne = String("stringThree = ");
  stringTwo = String("this string");
  stringThree = String ();
  // send an intro:
  Serial.println("\n\nAdding strings together (concatenation):");
```

```
  Serial.println();
 }

 void loop() {
  // adding a constant integer to a string:
  stringThree =  stringOne + 123;
  Serial.println(stringThree);    // prints "stringThree = 123"

  // adding a constant long integer to a string:
  stringThree = stringOne + 123456789;
  Serial.println(stringThree);    // prints " You added 123456789"

  // adding a constant character to a string:
  stringThree =  stringOne + 'A';
  Serial.println(stringThree);    // prints "You added A"

  // adding a constant string to a string:
  stringThree =  stringOne +  "abc";
  Serial.println(stringThree);    // prints "You added abc"

  stringThree = stringOne + stringTwo;
  Serial.println(stringThree);    // prints "You added this string"

  // adding a variable integer to a string:
  int sensorValue = analogRead(A0);
  stringOne = "Sensor value: ";
  stringThree = stringOne  + sensorValue;
   Serial.println(stringThree);    // prints "Sensor Value: 401" or whatever value
analogRead(A0) has

  // adding a variable long integer to a string:
  long currentTime = millis();
  stringOne="millis() value: ";
```

```
  stringThree = stringOne + millis();
  Serial.println(stringThree);   // prints "The millis: 345345" or whatever value currentTime has

  // do nothing while true:
  while(true);
}
```

# B) STRING COMPARISON OPERATORS

You can make comparisons between various strings using operators for String comparison such as equals () or equalIgnoresCase () and ==, <=, >, !=, >=, <. Sorting and sequencing are important activities when creating databases.

Many functions are interchangeable such as equals () and == operators. You may use either one since the answer will be the same. That is why you will that:

```
if (stringOne.equals(stringTwo)) {
```

will perform the same operation as

```
if (stringOne ==stringTwo) {
```

Greater than and less than operators will check the first alphabet or number in the string. This is why you get "a" < "b" and "1" < "2", but "999"> "1000" because 9 comes after 1.

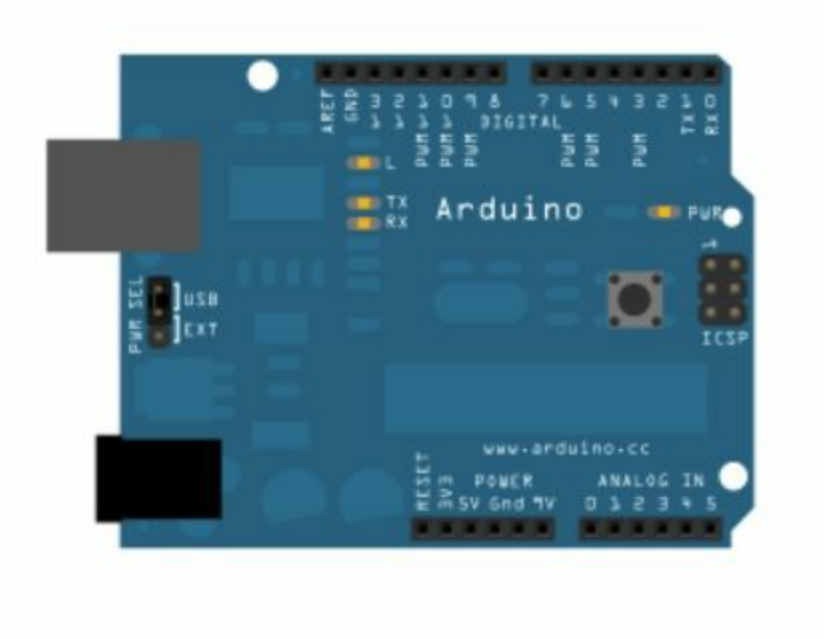
Take note: When you use operators for string comparison, do not think the strings as numbers. If you want mathematical values, you have to use variable such as floats, ints.

## Hardware and Circuit Required

Arduino board

You will not require any circuit in this tutorial. The only connection you need to

make is between the Arduino board and your computer.




# Code

```
/*
  Comparing Strings

  Examples of how to compare strings using the comparison operators

  created 27 July 2010
  modified 2 Apr 2012
  by Tom Igoe

  http://arduino.cc/en/Tutorial/StringComparisonOperators

  This example code is in the public domain.
  */

String stringOne, stringTwo;

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
```

```
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }


  stringOne = String("this");
  stringTwo = String("that");
  // send an intro:
  Serial.println("\n\nComparing Strings:");
  Serial.println();

}

void loop() {
  // two strings equal:
  if (stringOne == "this") {
    Serial.println("StringOne == \"this\"");
  }
  // two strings not equal:
  if (stringOne != stringTwo) {
    Serial.println(stringOne + " =! " + stringTwo);
  }

  // two strings not equal (case sensitivity matters):
  stringOne = "This";
  stringTwo = "this";
  if (stringOne != stringTwo) {
    Serial.println(stringOne + " =! " + stringTwo);
  }
  // you can also use equals() to see if two strings are the same:
  if (stringOne.equals(stringTwo)) {
    Serial.println(stringOne + " equals " + stringTwo);
  }
```

```
else {
  Serial.println(stringOne + " does not equal " + stringTwo);
}

// or perhaps you want to ignore case:
if (stringOne.equalsIgnoreCase(stringTwo)) {
  Serial.println(stringOne + " equals (ignoring case) " + stringTwo);
}
else {
  Serial.println(stringOne + " does not equal (ignoring case) " + stringTwo);
}

// a numeric string compared to the number it represents:
stringOne = "1";
int numberOne = 1;
if (stringOne.toInt() == numberOne) {
  Serial.println(stringOne + " = " + numberOne);
}



// two numeric strings compared:
stringOne = "2";
stringTwo = "1";
if (stringOne >= stringTwo) {
  Serial.println(stringOne + " >= " + stringTwo);
}

// comparison operators can be used to compare strings for alphabetic sorting too:
stringOne = String("Brown");
if (stringOne < "Charles") {
  Serial.println(stringOne + " < Charles");
}
```

```
if (stringOne > "Adams") {
  Serial.println(stringOne + " > Adams");
}

if (stringOne <= "Browne") {
  Serial.println(stringOne + " <= Browne");
}


if (stringOne >= "Brow") {
  Serial.println(stringOne + " >= Brow");
}

// the compareTo() operator also allows you to compare strings
// it evaluates on the first character that's different.
// if the first character of the string you're comparing to
// comes first in alphanumeric order, then compareTo() is greater than 0:
stringOne = "Cucumber";
stringTwo = "Cucuracha";
if (stringOne.compareTo(stringTwo) < 0 ) {
  Serial.println(stringOne + " comes before " + stringTwo);
}
else {
  Serial.println(stringOne + " comes after " + stringTwo);
}

delay(10000);  // because the next part is a loop:

// compareTo() is handy when you've got strings with numbers in them too:

while (true) {
  stringOne = "Sensor: ";
```

```
    stringTwo= "Sensor: ";

    stringOne += analogRead(A0);
    stringTwo += analogRead(A5);

    if (stringOne.compareTo(stringTwo) < 0 ) {
      Serial.println(stringOne + " comes before " + stringTwo);
    }
    else {
      Serial.println(stringOne + " comes after " + stringTwo);

    }
  }
}
```

## Chapter 6

# ARDUINO IN HARDWARE INTERFACE

Arduino comes into play for interfacing devices that are used as input, output or user interfaces. They also help devices used in communication and an ISP programmer. In order to use the kit or assemble it, you have to be aware of the full capabilities of this revolutionary computing device.

See all that is possible and begin a project that interests you. They help in governing physical systems and in regulating the audio systems. You can use them in multiplexing and controlling and distributing electrical power. They are used as input devices in power, audio and automobiles.

Specific examples for each of these are given here. Scientists use them in environmental sciences and distance sensing devices. Other uses include their use in human interfacing, game controllers, keypads and light sensors. Here we are going to see how to use the Arduino for some basic functions.

## SIMPLE BASIC CONTROLS

### A) Fading an LED

For this experiment, you need the following equipment.

- Breadboard
- 220 ohm resistor
- Arduino board
- An LED

Select the 9 pin of your Arduino and solder a 220 ohm resistor to it. Connect the other end of the resistor to anode the positive leg of your LED. The shorter cathode leg in the LED should be directly connected to the ground. Now to write the code.

First declare the ledPin is your 9 pin. This will be all there is in your setup () portion of the code. Next comes the analog write () function. Here you need to pass two functions – one that tells which pin it has to write to and one that tells what PWM value it has to write.

To make the LED fade off, you need to increase the PWM value that is at 0 all the way to 255. Now decrease the value all the way back to 0. You can use any variable to change the PWM value. In this example, we use fullOrOff. For each loop, the increment is done by a variable goDark.

When the fullOrOff is at one end of the value scale, either 0 or 255, the goDark value is changed to negative. This means that if the fullOrOff is at 10, then goDark is assigned -10 and if it reaches 250, then the goDark value is given +10 value. The analogWrite will do the steps very fast. To control the speed of your fade, you use delay at the sketch end. To see the difference actually, change the delay value and check how the program responds. Here is the program.

```
/*
 Fade
 You will see in this example how you fade the LED on the pin 9
      You use the analogWrite () function
      You are reading the function written in public domain.

*/

int led = 9;           // the pin that the LED is attached to
int fullOrOff = 0;    // how bright the LED is
int goDark = 5;    // how many points to fade the LED by

// when you press the reset the setup routine runs

    void setup () {
```

```
    // declare pin 9 to be an output:
  pinMode (led, OUTPUT);
}

// the loop will not stop executing, it goes on
    void loop () {

    // set the fullOrOff of pin 9:

    analogWrite (led, fullOrOff);

  // change the fullOrOff for next time through the loop:
  fullOrOff = fullOrOff + goDark;

  // when the loop reaches the end reverse its direction of fading

      if (fullOrOff == 0 || fullOrOff == 255) {
    goDark = -goDark;
  }

      // dimming will occur at the end of 30 seconds
      delay (30);
}
```

## B) Blink the LED

In this example too we use the same material as before. You can use 220 Ω to 1000 Ω resistance. You may sometimes find that the LED is already attached to your Arduino board. Otherwise, solder the LED to the 13 pin like before.

To begin the code, initialize the 13 pin to be your output pin:

pinMode (13, OUTPUT);

To turn the LED on , use this code in the main loop.

digitalWrite(13, HIGH);

Once the voltage comes to the 13 pin, it will light up the LED. Now you have to turn off the voltage by using the line:

digitalWrite(13, LOW);

Now the voltage in pin 13 drops back to zero and the LED light will switch off. If you want to really appreciate the work your program does, set the delay for 1000 milliseconds. During this period, nothing happens. Take a look at the program:

```
/*
 Blink
 You will see a repeat of this cycle – ON for one second, OFF the other
 You are reading this example code written expressly in the public domain
*/

// Most Arduino boards have pin 13 attached to LED
    // give it a name:
int led = 13;

// on pressing reset the setup routine starts again.
    void setup() {
 // initialize the digital pin as an output.
 pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
 digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
 delay(1000);               // wait for a second
 digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
 delay(1000);               // wait for a second
```

}

## C) Set up Button control

Button is used to connect two points when you press on it. In this example, we see how to make use of a button to turn on a LED connected to pin 13. For this example, you will need:

- Momentary button
- Breadboard
- Arduino board
- 10K Ω resistor
- Hook-up wire

You have to make three wire connection to begin your button control. The breadbox side has two rows that are long and vertical. This will be the red and black wire connecting the 5V ground and supply. From pin 2 connect one wire to pushbutton leg. From the same button leg, make a connection to 10K Ω pull down resistor to ground. The other leg button will be given the 5V supply.

When the button is not depressed, the connection between the legs of the button remains open. The pin has connection only to the ground and so it returns the LOW reading. When you press the button down, the reading now goes to 5V and so you get a HIGH reading.

Note that you can reverse the functionality of the Button by reversing the connection. This makes the LED button to always be on but will go off when you press the button. If you disconnect the i/o pin entirely, you will see the light blinking very randomly. The reason is that the input has now got a floating value and thus returns HIGH or LOW randomly. It is for this reason we include the pull-down resistor.

```
/*
 Button

Turns on and off a light emitting diode(LED) connected to digital
pin 13, when pressing a pushbutton attached to pin 2.


The circuit:
* LED attached from pin 13 to ground
* pushbutton attached to pin 2 from +5V
* 10K resistor attached to pin 2 from ground

* Note: on most Arduinos there is already an LED on the board
attached to pin 13.


created 2005
by DojoDave <http://www.0j0.org>
modified 30 Aug 2011
by Tom Igoe

This example code is in the public domain.

http://www.arduino.cc/en/Tutorial/Button
*/

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
```

```
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

## D)Simple Keyboard

This example will see the use of the tone function. Tone () command will generate sounds with different pitches. Here you will need the following material.

- 3 numbers force-resisting sensors
- 1 number 100 Ω resistor
- 3 numbers 10K Ω resistors

- Hook up wire
- 8 Ω speaker
- Breadboard

Connect the three force resisting sensors to 5V supply in parallel. Connect each of them to the pins 0 –2 with 10K Ω resistor to ground. When you run the code, it will check whether any of the FSRs are above the threshold assigned to it. The sketch will assign one note to each FSR from an array of notes. Here is the code.

```
/*
 keyboard

 Plays a pitch that changes based on a changing analog input

 circuit:
* 3 force-sensing resistors from +5V to analog in 0 through 5
* 3 10K resistors from analog in 0 through 5 to ground
* 8-ohm speaker on digital pin 8

 created 21 Jan 2010
 modified 9 Apr 2012
 by Tom Igoe

This example code is in the public domain

 http://arduino.cc/en/Tutorial/Tone3

 */

#include "pitches.h"

const int threshold = 10;    // minimum reading of the sensors that generates a note
```

```
// notes to play, corresponding to the 3 sensors:
int notes[] = {
  NOTE_A4, NOTE_B4,NOTE_C3 };

void setup() {

}

void loop() {
  for (int thisSensor = 0; thisSensor < 3; thisSensor++) {
    // get a sensor reading:
    int sensorReading = analogRead(thisSensor);

    // if the sensor is pressed hard enough:
    if (sensorReading > threshold) {
      // play the note corresponding to this sensor:
      tone(8, notes[thisSensor], 20);
    }
  }
}
```

You will see that there is a file pitches.h that contains all the pitches for each note. The notes are denoted by their key and a number according to the ascending of the pitch. This table was originally made by Brett Hagmann who is famous for his work that resulted in the tone () function. You have to make a new tab and past the following code to make the file.

```
/*************************************************
                * Public Constants

*************************************************/

#define NOTE_B0  31
```

```
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
```

```
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
```

```
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

## E) Ultrasonic Sensor NewPing

You can use the Arduino to construct a simple ultrasonic sensor. This model is very slow and cannot be used with interrupt code. You can set a delay of 1 second if that is absolutely necessary. This program returns the distance (inches) and you can use the result for other calculations. Jason Ch has compiled the code written here.

```
unsigned long echo = 0;

int ultraSoundSignal = 9; // Ultrasound signal pin
```

```
unsigned long ultrasoundValue = 0;


void setup ()
{
Serial.begin(9600);
pinMode(ultraSoundSignal, OUTPUT);
}


unsigned long ping(){
pinMode(ultraSoundSignal, OUTPUT); // Switch signalpin to output
digitalWrite(ultraSoundSignal, LOW); // Send low pulse
delayMicroseconds(2); // Wait for 2 microseconds
digitalWrite(ultraSoundSignal, HIGH); // Send high pulse
delayMicroseconds(5); // Wait for 5 microseconds
digitalWrite(ultraSoundSignal, LOW); // Holdoff
pinMode(ultraSoundSignal, INPUT); // Switch signalpin to input
digitalWrite(ultraSoundSignal, HIGH); // Turn on pullup resistor
  // please note that pulseIn has a 1sec timeout, which may
  // not be desirable.  Depending on your sensor specs, you
```

```
  // can likely bound the time like this -- marcmerlin
  // echo = pulseIn(ultraSoundSignal, HIGH, 38000)
  echo = pulseIn(ultraSoundSignal, HIGH); //Listen for echo
  ultrasoundValue = (echo / 58.138) * .39; //convert to CM then to inches
return ultrasoundValue;
}

void loop()
{
int x = 0;
x = ping();
  Serial.println(x);
  delay(250); //delay 1/4 seconds.
}
```

## F) Example Sketch for NewPing

(does 20 pings per second)

(Go here for more information.)

```
#include <NewPing.h>
```

```
#define TRIGGER_PIN  12  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     11  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters).  Maximum sensor distance is rated at 400-500cm.

    NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.


void setup() {
  Serial.begin(115200); // Open serial monitor at 115200 baud to see ping results.
}


void loop() {
  delay(50);                     // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest delay between pings.
  unsigned int uS = sonar.ping(); // Send ping, get ping time in microseconds (uS).
  Serial.print("Ping: ");
  Serial.print(uS / US_ROUNDTRIP_CM); // Convert ping time to distance and print result (0 = outside set distance range, no ping echo)
  Serial.println("cm");
}
```

## Chapter 7

# SOME MORE SPECIAL INFORMATION

## Conditions For Building A Commercial Product

You can fashion your own commercial product but be sure to follow these conditions.

- If you make use of the design from some Arduino board Eagle files, then you are obliged to resubmit the modified files under the same Creative Commons Attribution share-Alike license.
- The GPL covers the Arduino environment and this does not limit the sale of the software thus derived or selling the product commercially, the only condition being that it must keep any modification under the same Creative Commons license.
- Design of the product need not be open source if you use an Arduino board in your commercial product. You do not have to disclose any information regarding the product.
- You can use the libraries and the Arduino core without being under any obligation to disclose information regarding the firmware. However, if you make modifications to the libraries or the core or if you require updates from future version Arduino libraries or cores, you need to you need to submit modification under LGPL and make available object files which you use for relinking the firmware against updated versions.

## Arduino Counterfeit Boards Learning To Spot The Fake

First off, check the color. Color must be teal. You can turn the board around and see the graphics on the back. The typical logo will be an eight lying on its side with a negative and positive sign within the two orbs. Duplicates will be blue to deep blue in color. The font will not be Silk as it is in the original. The printing is clear and symmetrical in the original. If the printing is not clear or does not have symmetry, then it is probably a fake. You will also see the map of Italy. This also must be clear.

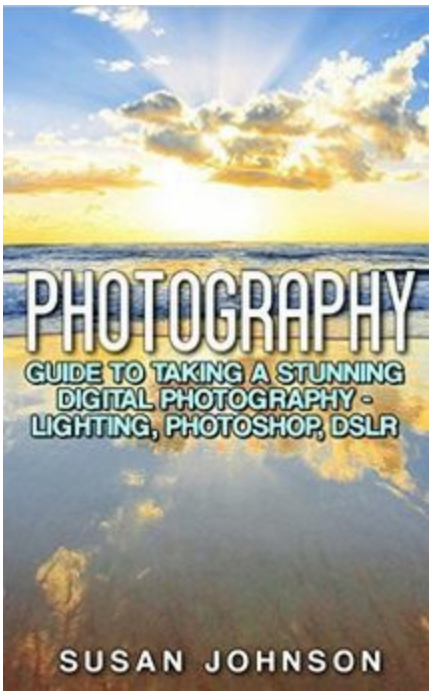
Look for the 501K component. Most fakes will have a green colored component that is available over the counter. The brown component is specially made for Arduino. Lastly check if the Arduino is sold by an authorized dealer.

## Use C to program your Arduino board

Arduino language for most part consists of C/C++ functions that you call. The sketch will be changed in a minor way (for example when you create function prototypes automatically) and is then taken up by the C/C++ compiler avr-g++. So every C or C++ construct that is working for avr-g++ will work on Arduino. You can get more details on this here.

For more information, please visit this page.

# Other Recommendations




## Photography: Guide to Taking a Stunning Digital Photography - Lighting, Photoshop, DSLR

**Learn Things Like...**

- Basic foundation of your camera - starting things right
- Secret rules that is used by the author to take astonishing photos
- What to keep in mind when building your own indoor studio
- Checking your images and using Photoshop


# Apps: Beginner's Guide for App Programming, App Development, App Design

*Do you want to learn how to program your own app? Are you read to create something that could potentially change the world?*

Download “ Apps: Beginner's Guide for App Programming, App Development, App Design ” and learn the basic foundations of App programming so you can start programming your own app starting from tomorrow! What are you waiting for? Take action right now and become a programmer

If the links do not work, for whatever reason, you can simply search them on Amazon website

# Free Bonus($9.99): Get My Latest Kindle E-Book “Top 10 Gadgets of 2015” for Free

As a “Thank You” for downloading, and reading my book I would like to send you my latest E-book “Top 10 Gadgets of 2015” for F.R.E.E. This is no strings attached offer, just my gift to you for being a great customer. Just Click on the image below


or type: https://cracklifecode.leadpages.net/technology/