# Arduino

## Examples

*See the **foundations page** for in-depth description of core concepts of the Arduino hardware and software; the **hacking page** for information on extending and modifying the Arduino hardware and software; and the **links page** for other documentation.*

### Examples

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.)

#### Digital I/O

- Blink: turn an LED on and off.
- Blink Without Delay: blinking an LED without using the delay() function.
- Button: use a pushbutton to control an LED.
- Debounce: read a pushbutton, filtering noise.
- Loop: controlling multiple LEDs with a loop and an array.

#### Analog I/O

- Analog Input: use a potentiometer to control the blinking of an LED.
- Fading: uses an analog output (PWM pin) to fade an LED.
- Knock: detect knocks with a piezo element.
- Smoothing: smooth multiple readings of an analog input.

#### Communication

*These examples include code that allows the Arduino to talk to Processing sketches running on the computer. For more information or to download Processing, see processing.org.*

- ASCII Table: demonstrates Arduino's advanced serial output functions.
- Dimmer: move the mouse to change the brightness of an LED.
- Graph: sending data to the computer and graphing it in Processing.
- Physical Pixel: turning on and off an LED by sending data from Processing.
- Virtual Color Mixer: sending multiple variables from Arduino to the computer and reading them in Processing.

#### EEPROM Library

### Other Examples

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included on the page.

#### Miscellaneous

- TwoSwitchesOnePin: Read two switches with one I/O pin
- Read a Tilt Sensor
- Controlling an LED circle with a joystick
- 3 LED color mixer with 3 potentiometers

### Timing & Millis

- Stopwatch

#### Complex Sensors

- Read an ADXL3xx accelerometer
- Read an Accelerometer
- Read an Ultrasonic Range Finder (ultrasound sensor)
- Reading the qprox qt401 linear touch sensor

#### Sound

- Play Melodies with a Piezo Speaker
- Play Tones from the Serial Connection
- MIDI Output (from ITP physcomp labs) and from Spooky Arduino

#### Interfacing w/ Hardware

- Multiply the Amount of Outputs with an LED Driver
- Interfacing an LCD display with 8 bits
    - LCD interface library
- Driving a DC Motor with an L293 (from ITP physcomp labs).
- Driving a Unipolar Stepper Motor
- Build your own DMX Master device
- Implement a software serial connection
    - RS-232 computer interface
- Interface with a serial EEPROM using SPI
- Control a digital potentiometer using SPI
- Multiple digital outs with a 595 Shift Register
- X10 output control devices over AC powerlines using X10

- EEPROM Clear: clear the bytes in the EEPROM.
- EEPROM Read: read the EEPROM and send its values to the computer.
- EEPROM Write: stores values from an analog input to the EEPROM.

**Stepper Library**

- Motor Knob: control a stepper motor with a potentiometer.

**Learning** Examples | **Foundations** | Hacking | Links

## Foundations

This page contains explanations of some of the elements of the Arduino hardware and software and the concepts behind them. Page Discussion

### Basics

- Sketch: The various components of a sketch and how they work.

### Microcontrollers

- Digital Pins: How the pins work and what it means for them to be configured as inputs or outputs.
- Analog Input Pins: Details about the analog-to-digital conversion and other uses of the pins.
- PWM: How the analogWrite() function simulates an analog output using pulse-width modulation.
- Memory: The various types of memory available on the Arduino board.

### Arduino Firmware

- Bootloader: A small program pre-loaded on the Arduino board to allow uploading sketches.

### Programming Technique

- Variables: How to define and use variables.
- Port Manipulation: Manipulating ports directly for faster manipulation of multiple pins

**Learning**   Examples | Foundations | Hacking | **Links**

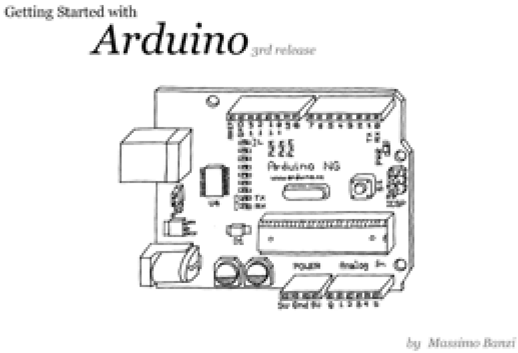
# Links

Arduino examples, tutorials, and documentation elsewhere on the web.

### Books and Manuals

Make
Making Things Talk
Practical Methods for Connecting Physical Objects
PROJECTS AND IDEAS TO CREATE TALKING OBJECTS FROM ANYTHING
O'REILLY
Tom Igoe

Making Things Talk (by Tom Igoe): teaches you how to get your creations to communicate with one another by forming networks of smart devices that carry on conversations with you and your environment.



Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

### Community Documentation

Tutorials created by the Arduino community. Hosted on the publicly-editable playground wiki.

Board Setup and Configuration: Information about the components and usage of Arduino hardware.

Interfacing With Hardware: Code, circuits, and instructions for using various electronic components with an Arduino board.

- Output
- Input
- Interaction
- Storage
- Communication

Interfacing with Software: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

Code Library and Tutorials: Arduino functions for performing specific tasks and other programming tutorials.

Electronics Techniques: tutorials on soldering and other electronics resources.

### Other Examples and Tutorials

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Tom Igoe's Physical Computing Site: lots of information on electronics, microcontrollers, sensors, actuators, books, etc.

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Wiring electronics reference: circuit diagrams for connecting a variety of basic electronic components.

Schematics to circuits: from Wiring, a guide to transforming circuit diagrams into physical circuits.

Examples from Tom Igoe

Examples from Jeff Gray

**Learning** Examples | Foundations | Hacking | Links

# Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide.

## Examples

### Digital Output

- Blinking LED
- Blinking an LED without using the delay() function
- Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- More complex dimming/color crossfader
- Knight Rider example
- Shooting star
- PWM all of the digital pins in a sinewave pattern

### Digital Input

- Digital Input and Output (from ITP physcomp labs)
- Read a Pushbutton
- Using a pushbutton as a switch
- Read a Tilt Sensor

### Analog Input

- Read a Potentiometer
- Interfacing a Joystick
- Controlling an LED circle with a joystick
- Read a Piezo Sensor
- 3 LED cross-fades with a potentiometer
- 3 LED color mixer with 3 potentiometers

### Complex Sensors

- Read an Accelerometer
- Read an Ultrasonic Range Finder (ultrasound sensor)
- Reading the qprox qt401 linear touch sensor
- Use two Arduino pins as a capacitive sensor

### Sound

- Play Melodies with a Piezo Speaker
- More sound ideas
- Play Tones from the Serial Connection
- MIDI Output (from ITP physcomp labs) and from Spooky Arduino

## Interfacing with Other Software

- Introduction to Serial Communication (from ITP physcomp labs)
- Arduino + Flash
- Arduino + Processing
- Arduino + PD
- Arduino + MaxMSP
- Arduino + VVVV
- Arduino + Director
- Arduino + Ruby
- Arduino + C

## Tech Notes (from the **forums** or **playground**)

- Software serial (serial on pins besides 0 and 1)
- L297 motor driver
- Hex inverter
- Analog multiplexer
- Power supplies
- The components on the Arduino board
- Arduino build process
- AVRISP mkII on the Mac
- Non-volatile memory (EEPROM)
- Bluetooth
- Zigbee
- LED as light sensor (en Francais)
- Arduino and the Asuro robot
- Using Arduino from the command line

### Interfacing w/ Hardware

- Multiply the Amount of Outputs with an LED Driver
- Interfacing an LCD display with 8 bits
    - LCD interface library
- Driving a DC Motor with an L293 (from ITP physcomp labs).
- Driving a Unipolar Stepper Motor
- Implement a software serial connection
    - RS-232 computer interface
- Interface with a serial EEPROM using SPI
- Control a digital potentiometer using SPI
- Multiple digital outs with a 595 Shift Register
- Multiple digital inputs with a CD4021 Shift Register

## Other Arduino Examples

- Example labs from ITP
- Examples from Tom Igoe
- Examples from Jeff Gray

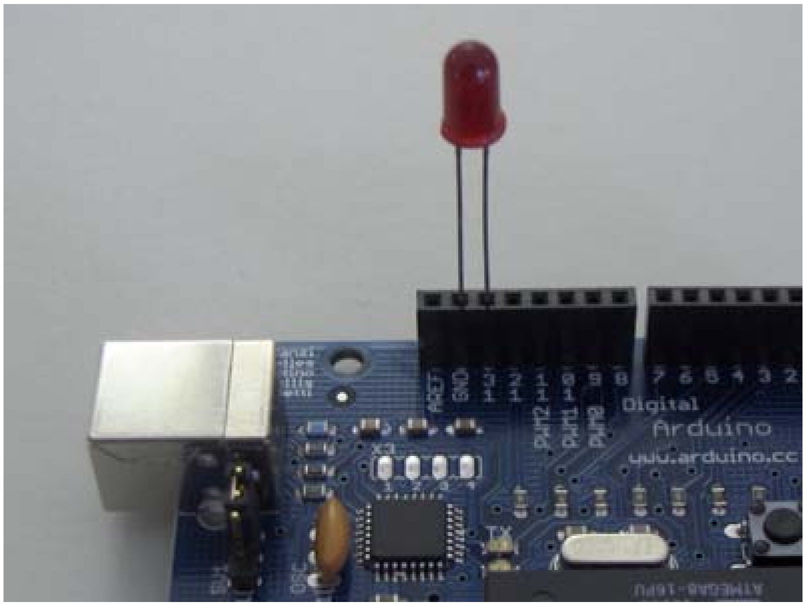
## Blink

In most programming languages, the first program you write prints "hello world" to the screen. Since an Arduino board doesn't have a screen, we blink an LED instead.

The boards are designed to make it easy to blink an LED using digital pin 13. Some (like the Diecimila and LilyPad) have the LED built-in to the board. On most others (like the Mini and BT), there is a 1 KB resistor on the pin, allowing you to connect an LED directly. (To connect an LED to another digital pin, you should use an external resistor.)

LEDs have polarity, which means they will only light up if you orient the legs properly. The long leg is typically positive, and should connect to pin 13. The short leg connects to GND; the bulb of the LED will also typically have a flat edge on this side. If the LED doesn't light up, trying reversing the legs (you won't hurt the LED if you plug it in backwards for a short period of time).

### Circuit



### Code

The example code is very simple, credits are to be found in the comments.

```
/* Blinking LED
 * ------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, in intervals of 2 seconds. Ideally we use pin 13 on the Arduino
 * board because it has a resistor attached to it, needing only an LED

 *
 * Created 1 June 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 * based on an orginal by H. Barragan for the Wiring i/o board
 */

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

void setup()
{
  pinMode(ledPin, OUTPUT);     // sets the digital pin as output
}

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

*Examples > Digital I/O*

# Blink Without Delay

Sometimes you need to blink an LED (or some other time sensitive function) at the same time as something else (like watching for a button press). That means you can't use delay(), or you'd stop everything else the program while the LED blinked. Here's some code that demonstrates how to blink the LED without using delay(). It keeps track of the last time it turned the LED on or off. Then, each time through loop() it checks if a sufficient interval has passed - if it has, it turns the LED off if it was on and vice-versa.

## Code

```
int ledPin = 13;                // LED connected to digital pin 13
int value = LOW;                // previous value of the LED
long previousMillis = 0;        // will store last time LED was updated
long interval = 1000;           // interval at which to blink (milliseconds)

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, is the difference
  // between the current time and last time we blinked the LED bigger than
  // the interval at which we want to blink the LED.
  if (millis() - previousMillis > interval) {
    previousMillis = millis();   // remember the last time we blinked the LED

    // if the LED is off turn it on and vice-versa.
    if (value == LOW)
      value = HIGH;
    else
      value = LOW;

    digitalWrite(ledPin, value);
  }
}
```

## Button

The pushbutton is a component that connects two points in a circuit when you press it. The example turns on an LED when you press the button.

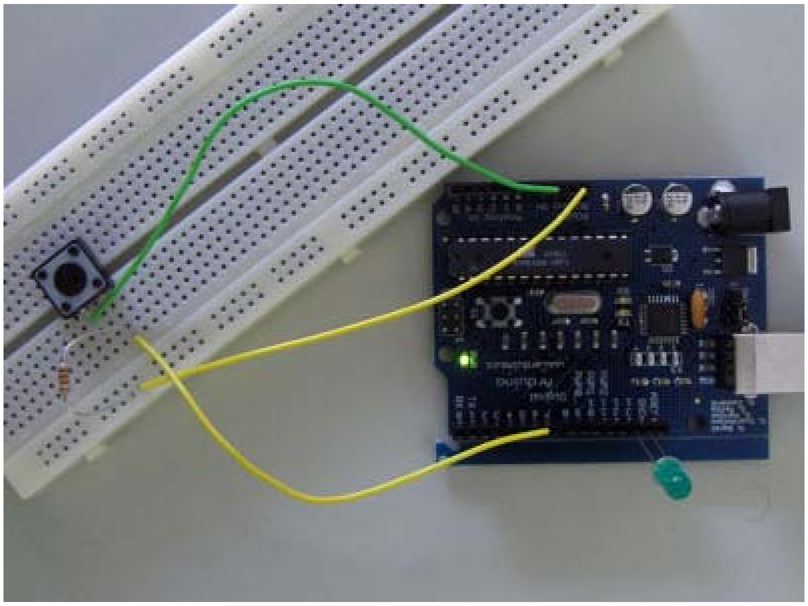
We connect three wires to the Arduino board. The first goes from one leg of the pushbutton through a pull-up resistor (here 2.2 KOhms) to the 5 volt supply. The second goes from the corresponding leg of the pushbutton to ground. The third connects to a digital i/o pin (here pin 7) which reads the button's state.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to 5 volts (through the pull-up resistor) and we read a HIGH. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to ground, so that we read a LOW. (The pin is still connected to 5 volts, but the resistor in-between them means that the pin is "closer" to ground.)

You can also wire this circuit the opposite way, with a pull-down resistor keeping the input LOW, and going HIGH when the button is pressed. If so, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will more-or-less randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resister in the circuit.

### Circuit



### Code

```
int ledPin = 13; // choose the pin for the LED
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

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

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

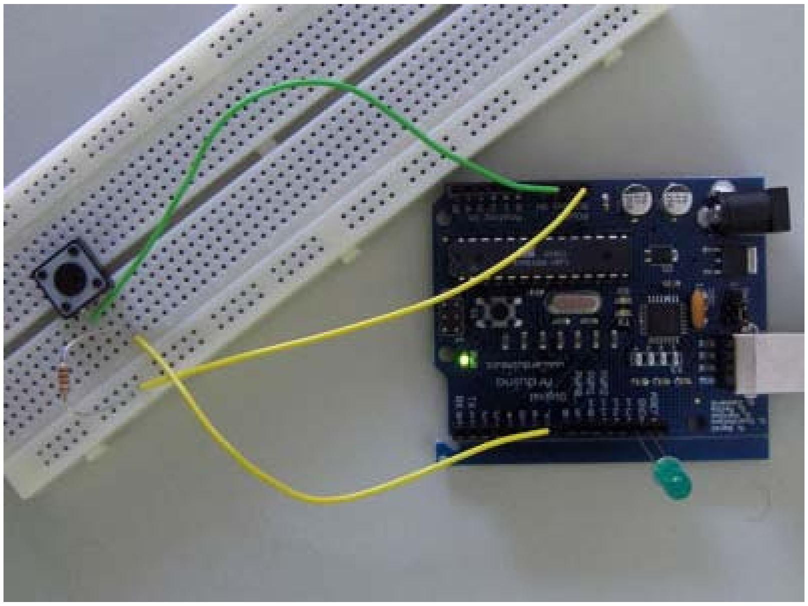
*Examples > Digital I/O*

# Debounce

This example demonstrates the use of a pushbutton as a switch: each time you press the button, the LED (or whatever) is turned on (if it's off) or off (if on). It also debounces the input, without which pressing the button once would appear to the code as multiple presses. Makes use of the millis() function to keep track of the time when the button is pressed.

## Circuit

A push-button on pin 7 and an LED on pin 13.



## Code

```
int inPin = 7;         // the number of the input pin
int outPin = 13;       // the number of the output pin

int state = HIGH;      // the current state of the output pin
int reading;           // the current reading from the input pin
int previous = LOW;    // the previous reading from the input pin

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 200;   // the debounce time, increase if the output flickers

void setup()
{
  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
}

void loop()
```

```
{
  reading = digitalRead(inPin);

  // if we just pressed the button (i.e. the input went from LOW to HIGH),
  // and we've waited long enough since the last press to ignore any noise...
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    // ... invert the output
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;

    // ... and remember when the last button press was
    time = millis();
  }

  digitalWrite(outPin, state);

  previous = reading;
}
```

Edit Page | Page History | Printable View | All Recent Site Changes

*Examples > Digital I/O*

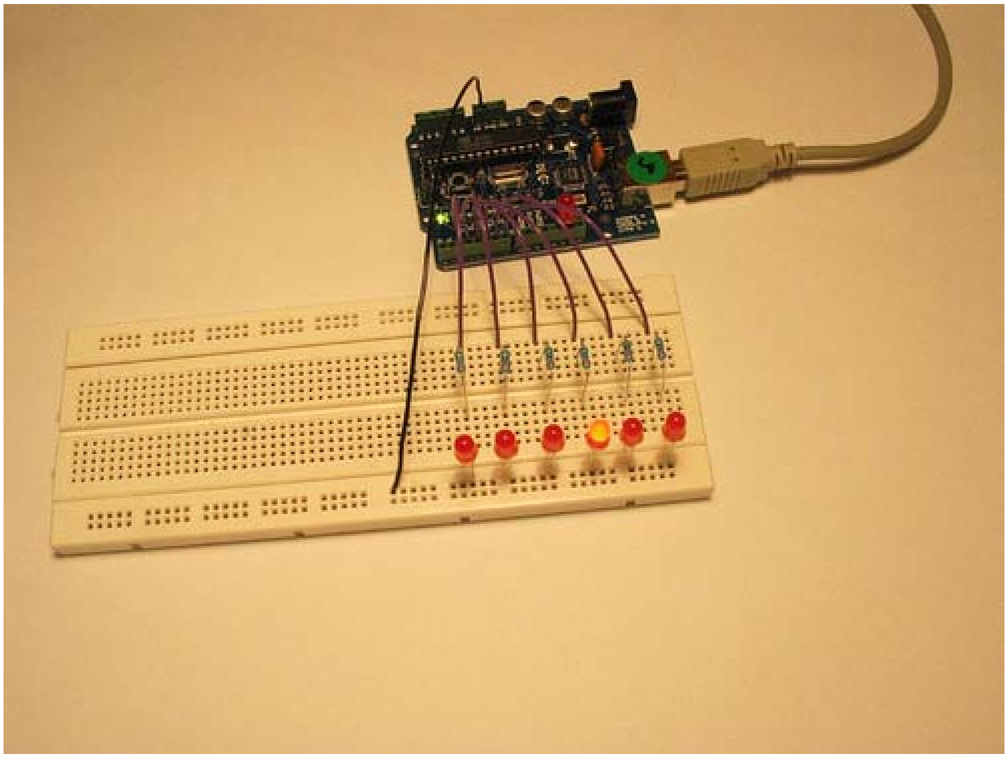
## Loop

We also call this example "Knight Rider" in memory to a TV-series from the 80's where the famous David Hasselhoff had an AI machine driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects.

Thus we decided that in order to learn more about sequential programming and good programming techniques for the I/O board, it would be interesting to use the **Knight Rider** as a metaphor.

This example makes use of 6 LEDs connected to the pins 2 - 7 on the board using 220 Ohm resistors. The first code example will make the LEDs blink in a sequence, one by one using only **digitalWrite(pinNum,HIGH/LOW)** and **delay(time)**. The second example shows how to use a **for(;;)** construction to perform the very same thing, but in fewer lines. The third and last example concentrates in the visual effect of turning the LEDs on/off in a more softer way.

### Circuit



### Code

```
int timer = 100;                   // The higher the number, the slower the timing.
int pins[] = { 2, 3, 4, 5, 6, 7 }; // an array of pin numbers
int num_pins = 6;                  // the number of pins (i.e. the length of the array)

void setup()
{
  int i;

  for (i = 0; i < num pins; i++)   // the array elements are numbered from 0 to num pins - 1
```

```
    pinMode(pins[i], OUTPUT);      // set each pin as an output
}

void loop()
{
  int i;

  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);   // turning it on,
    delay(timer);                  // pausing,
    digitalWrite(pins[i], LOW);    // and turning it off.
  }
  for (i = num_pins - 1; i >= 0; i--) {
    digitalWrite(pins[i], HIGH);
    delay(timer);
    digitalWrite(pins[i], LOW);
  }
}
```

*Examples > Analog I/O*

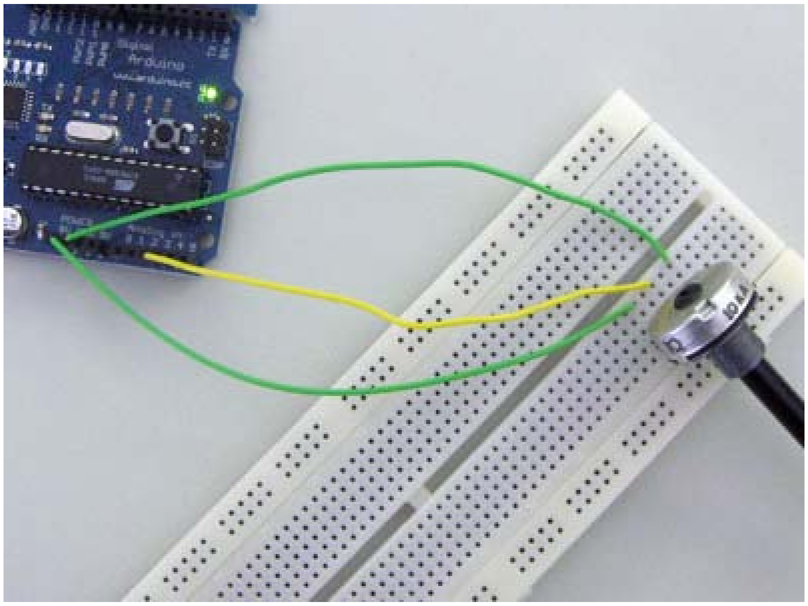
## Analog Input

A potentiometer is a simple knob that provides a variable resistance, which we can read into the Arduino board as an analog value. In this example, that value controls the rate at which an LED blinks.

We connect three wires to the Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, we change the amount of resistence on either side of the wiper which is connected to the center pin of the potentiometer. This changes the relative "closeness" of that pin to 5 volts and ground, giving us a different analog input. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft is turned all the way in the other direction, there are 5 volts going to the pin and we read 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

### Circuit



### Code

```
/*
 * AnalogInput
 * by DojoDave <http://www.0j0.org>
 *
 * Turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead(). In the easiest case we connect
 * a potentiometer to analog pin 2.
 */

int potPin = 2;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor

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

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  digitalWrite(ledPin, HIGH);  // turn the ledPin on
  delay(val);                  // stop the program for some time
  digitalWrite(ledPin, LOW);   // turn the ledPin off
  delay(val);                  // stop the program for some time
}
```

**Learning** Examples | Foundations | Hacking | Links

*Examples > Analog I/O*

# Fading

Demonstrates the use of analog output (PWM) to fade an LED.

## Circuit

An LED connected to digital pin 9.

## Code

```
int value = 0;                            // variable to keep the actual value
int ledpin = 9;                           // light connected to digital pin 9

void setup()
{
  // nothing for setup
}

void loop()
{
  for(value = 0 ; value <= 255; value+=5) // fade in (from min to max)
  {
    analogWrite(ledpin, value);           // sets the value (range from 0 to 255)
    delay(30);                            // waits for 30 milli seconds to see the dimming effect
  }
  for(value = 255; value >=0; value-=5)    // fade out (from max to min)
  {
    analogWrite(ledpin, value);
    delay(30);
  }
}
```

*Examples > Analog I/O*

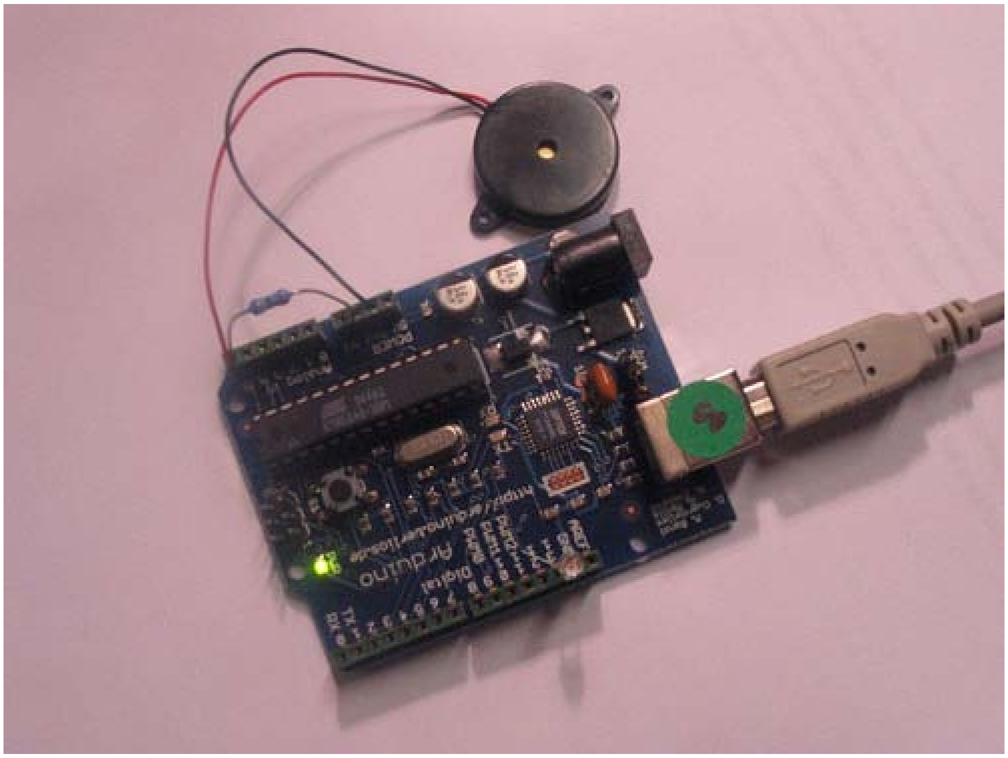
## Knock

Here we use a Piezo element to detect sound, what will allow us to use it as a knock sensor. We are taking advantage of the processors capability to read analog signals through its ADC - analog to digital converter. These converters read a voltage value and transform it into a value encoded digitally. In the case of the Arduino boards, we transform the voltage into a value in the range 0..1024. 0 represents 0volts, while 1024 represents 5volts at the input of one of the six analog pins.

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the analog input pin number 0, that supports the functionality of reading a value between 0 and 5volts, and not just a plain HIGH or LOW.

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the input. We also have to connect a resistor in the range of the Megaohms in parallel to the Piezo element; in the example we have plugged it directly in the female connectors. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc and are easier to use as input sensors.

The code example will capture the knock and if it is stronger than a certain threshold, it will send the string "Knock!" back to the computer over the serial port. In order to see this text you can use the Arduino serial monitor.



*Example of connection of a Piezo to analog pin 0 with a resistor*

```
/* Knock Sensor
 * by DojoDave <http://www.0j0.org>
 *
 * Program using a Piezo element as if it was a knock sensor.
 *
 * We have to basically listen to an analog pin and detect
 * if the signal goes over a certain threshold. It writes
 * "knock" to the serial port if the Threshold is crossed,
 * and toggles the LED on pin 13.
 *
 * http://www.arduino.cc/en/Tutorial/Knock
 */

int ledPin = 13;      // led connected to control pin 13
int knockSensor = 0;  // the knock sensor will be plugged at analog pin 0
byte val = 0;         // variable to store the value read from the sensor pin
int statePin = LOW;   // variable used to store the last LED status, to toggle the light
int THRESHOLD = 100;  // threshold value to decide when the detected sound is a knock or not

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

void loop() {
  val = analogRead(knockSensor);    // read the sensor and store it in the variable "val"
  if (val >= THRESHOLD) {
    statePin = !statePin;           // toggle the status of the ledPin (this trick doesn't use time cycles)
    digitalWrite(ledPin, statePin); // turn the led on or off
    Serial.println("Knock!");       // send the string "Knock!" back to the computer, followed by newline
    delay(10);                      // short delay to avoid overloading the serial port
   }
}
```

*Examples > Analog I/O*

## Smoothing

Reads repeatedly from an analog input, calculating a running average and printing it to the computer. Demonstrates the use of arrays.

### Circuit

Potentiometer on analog input pin 0.

### Code

```
// Define the number of samples to keep track of.  The higher the number,
// the more the readings will be smoothed, but the slower the output will
// respond to the input.  Using a #define rather than a normal variable lets
// use this value to determine the size of the readings array.
#define NUMREADINGS 10

int readings[NUMREADINGS];               // the readings from the analog input
int index = 0;                           // the index of the current reading
int total = 0;                           // the running total
int average = 0;                         // the average

int inputPin = 0;

void setup()
{
  Serial.begin(9600);                    // initialize serial communication with computer
  for (int i = 0; i < NUMREADINGS; i++)
    readings[i] = 0;                     // initialize all the readings to 0
}

void loop()
{
  total -= readings[index];              // subtract the last reading
  readings[index] = analogRead(inputPin); // read from the sensor
  total += readings[index];              // add the reading to the total
  index = (index + 1);                   // advance to the next index

  if (index >= NUMREADINGS)              // if we're at the end of the array...
    index = 0;                           // ...wrap around to the beginning

  average = total / NUMREADINGS;         // calculate the average
  Serial.println(average);               // send it to the computer (as ASCII digits)
}
```

**Learning** Examples | Foundations | Hacking | Links

*Examples > Communication*

## ASCII Table

Demonstrates the advanced serial printing functions by generating a table of characters and their ASCII values in decimal, hexadecimal, octal, and binary.

### Circuit

None, but the Arduino has to be connected to the computer.

### Code

```
// ASCII Table
// by Nicholas Zambetti <http://www.zambetti.com>

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

  // prints title with ending line break
  Serial.println("ASCII Table ~ Character Map");

  // wait for the long string to be sent
  delay(100);
}

int number = 33; // first visible character '!' is #33

void loop()
{
  Serial.print(number, BYTE);    // prints value unaltered, first will be '!'

  Serial.print(", dec: ");
  Serial.print(number);          // prints value as string in decimal (base 10)
  // Serial.print(number, DEC);  // this also works

  Serial.print(", hex: ");
  Serial.print(number, HEX);     // prints value as string in hexadecimal (base 16)

  Serial.print(", oct: ");
  Serial.print(number, OCT);     // prints value as string in octal (base 8)

  Serial.print(", bin: ");
  Serial.println(number, BIN);   // prints value as string in binary (base 2)
                                 // also prints ending line break

  // if printed last visible character '~' #126 ...
  if(number == 126) {
    // loop forever
    while(true) {
      continue;
    }
  }
  number++; // to the next character

  delay(100); // allow some time for the Serial data to be sent
}
```

### Output

```
ASCII Table ~ Character Map
!, dec: 33, hex: 21, oct: 41, bin: 100001
", dec: 34, hex: 22, oct: 42, bin: 100010
#, dec: 35, hex: 23, oct: 43, bin: 100011
$, dec: 36, hex: 24, oct: 44, bin: 100100
%, dec: 37, hex: 25, oct: 45, bin: 100101
&, dec: 38, hex: 26, oct: 46, bin: 100110
', dec: 39, hex: 27, oct: 47, bin: 100111
(, dec: 40, hex: 28, oct: 50, bin: 101000
...
```

*Examples > Communication*

# 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.

## Circuit

An LED connected to pin 9 (with appropriate resistor).

## Code

```
int ledPin = 9;

void setup()
{
  // begin the serial communication
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  byte val;

  // check if data has been sent from the computer
  if (Serial.available()) {
    // read the most recent byte (which will be from 0 to 255)
    val = Serial.read();
    // set the brightness of the LED
    analogWrite(ledPin, val);
  }
}
```

## Processing Code

```
// Dimmer - sends bytes over a serial port
// by David A. Mellis

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);
}
```

**Learning** Examples | Foundations | Hacking | Links

*Examples > Communication*

# Graph

A simple example of communication from the Arduino board to the computer: the value of an analog input is printed. We call this "serial" communication because the connection appears to both the Arduino and the computer as an old-fashioned serial port, even though it may actually use a USB cable.

You can use the Arduino serial monitor to view the sent data, or it can be read by Processing (see code below), Flash, PD, Max/MSP, etc.

## Circuit

An analog input connected to analog input pin 0.

## Code

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

void loop()
{
  Serial.println(analogRead(0));
  delay(20);
}
```

## Processing Code

```
// Graph
// by David A. Mellis
//
// Demonstrates reading data from the Arduino board by graphing the
// values received.
//
// based on Analog In
// by <a href="http://itp.jtnimoy.com">Josh Nimoy</a>.

import processing.serial.*;

Serial port;
String buff = "";
int NEWLINE = 10;

// Store the last 64 values received so we can graph them.
int[] values = new int[64];

void setup()
{
  size(512, 256);

  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()
{
  background(53);
  stroke(255);

  // Graph the stored values by drawing a lines between them.
  for (int i = 0; i < 63; i++)
    line(i * 8, 255 - values[i], (i + 1) * 8, 255 - values[i + 1]);

  while (port.available() > 0)
    serialEvent(port.read());
}

void serialEvent(int serial)
{
  if (serial != NEWLINE) {
    // Store all the characters on the line.
    buff += char(serial);
  } else {
    // The end of each line is marked by two characters, a carriage
    // return and a newline.  We're here because we've gotten a newline,
    // but we still need to strip off the carriage return.
    buff = buff.substring(0, buff.length()-1);

    // Parse the String into an integer.  We divide by 4 because
    // analog inputs go from 0 to 1023 while colors in Processing
    // only go from 0 to 255.
    int val = Integer.parseInt(buff)/4;

    // Clear the value of "buff"
    buff = "";

    // Shift over the existing values to make room for the new one.
    for (int i = 0; i < 63; i++)
      values[i] = values[i + 1];

    // Add the received value to the array.
    values[63] = val;
  }
}
```

# Physical Pixel

An example of using the Arduino board to receive data from the computer. In this case, the Arduino boards turns on an LED when it receives the character 'H', and turns off the LED when it receives the character 'L'.

The data can be sent from the Arduino serial monitor, or another program like Processing (see code below), Flash (via a serial-net proxy), PD, or Max/MSP.

## Circuit

An LED on pin 13.

## Code

```
int outputPin = 13;
int val;

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

void loop()
{
  if (Serial.available()) {
    val = Serial.read();
    if (val == 'H') {
      digitalWrite(outputPin, HIGH);
    }
    if (val == 'L') {
      digitalWrite(outputPin, LOW);
    }
  }
}
```

## Processing Code

```
// mouseover serial
// by BARRAGAN <http://people.interaction-ivrea.it/h.barragan>

// Demonstrates how to send data to the Arduino I/O board, in order to
// turn ON a light if the mouse is over a rectangle and turn it off
// if the mouse is not.

// created 13 May 2004

import processing.serial.*;

Serial port;

void setup()
```

```
{
  size(200, 200);
  noStroke();
  frameRate(10);

  // List all the available serial ports in the output pane.
  // You will need to choose the port that the Arduino board is
  // connected to from this list. The first port in the list is
  // port #0 and the third port in the list is port #2.
  println(Serial.list());

  // Open the port that the Arduino board is connected to (in this case #0)
  // Make sure to open the port at the same speed Arduino is using (9600bps)
  port = new Serial(this, Serial.list()[0], 9600);
}

// function to test if mouse is over square
boolean mouseOverRect()
{
  return ((mouseX >= 50)&&(mouseX <= 150)&&(mouseY >= 50)&(mouseY <= 150));
}

void draw()
{
  background(#222222);
  if(mouseOverRect())      // if mouse is over square
  {
    fill(#BBBBB0);         // change color
    port.write('H');       // send an 'H' to indicate mouse is over square
  } else {
    fill(#666660);         // change color
    port.write('L');       // send an 'L' otherwise
  }
  rect(50, 50, 100, 100);  // draw square
}
```

*Examples > Communication*

# Virtual Color Mixer

Demonstrates one technique for sending multiple values from the Arduino board to the computer. In this case, the readings from three potentiometers are used to set the red, green, and blue components of the background color of a Processing sketch.

## Circuit

Potentiometers connected to analog input pins 0, 1, and 2.

## Code

```
int redPin = 0;
int greenPin = 1;
int bluePin = 2;

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

void loop()
{
  Serial.print("R");
  Serial.println(analogRead(redPin));
  Serial.print("G");
  Serial.println(analogRead(greenPin));
  Serial.print("B");
  Serial.println(analogRead(bluePin));
  delay(100);
}
```

## Processing Code

```
/**
 * Color Mixer
 * by David A. Mellis
 *
 * Created 2 December 2006
 *
 * based on Analog In
 * by <a href="http://itp.jtnimoy.com">Josh Nimoy</a>.
 *
 * Created 8 February 2003
 * Updated 2 April 2005
 */

import processing.serial.*;

String buff = "";
int rval = 0, gval = 0, bval = 0;
int NEWLINE = 10;

Serial port;

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

  // Print a list in case COM1 doesn't work out
  println("Available serial ports:");
  println(Serial.list());

  //port = new Serial(this, "COM1", 9600);
  // Uses the first available port
  port = new Serial(this, Serial.list()[0], 9600);
}

void draw()
{
  while (port.available() > 0) {
    serialEvent(port.read());
  }
  background(rval, gval, bval);
}

void serialEvent(int serial)
{
  // If the variable "serial" is not equal to the value for
  // a new line, add the value to the variable "buff". If the
  // value "serial" is equal to the value for a new line,
  //  save the value of the buffer into the variable "val".
  if(serial != NEWLINE) {
    buff += char(serial);
  } else {
    // The first character tells us which color this value is for
    char c = buff.charAt(0);
    // Remove it from the string
    buff = buff.substring(1);
    // Discard the carriage return at the end of the buffer
    buff = buff.substring(0, buff.length()-1);
    // Parse the String into an integer
    if (c == 'R')
      rval = Integer.parseInt(buff);
    else if (c == 'G')
      gval = Integer.parseInt(buff);
    else if (c == 'B')
      bval = Integer.parseInt(buff);
    // Clear the value of "buff"
    buff = "";
  }
}
```

## Read Two Switches With One I/O Pin

There are handy 20K pullup resistors (resistors connected internally between Arduino I/O pins and VCC - +5 volts in the Arduino's case) built into the Atmega chip upon which Freeduino's are based. They are accessible from software by using the digitalWrite() function, when the pin is set to an input.

This sketch exploits the pullup resistors under software control. The idea is that an external 200K resistor to ground will cause an input pin to report LOW when the internal (20K) pullup resistor is turned off. When the internal pullup resistor is turned on however, it will overwhelm the external 200K resistor and the pin will report HIGH.

One downside of the scheme (there always has to be a downside doesn't there?) is that one can't tell if both buttons are pushed at the same time. In this case the scheme just reports that sw2 is pushed. The job of the 10K series resistor, incidentally, is to prevent a short circuit if a pesky user pushes both buttons at once. It can be omitted on a center-off slide or toggle switch where the states are mutually exclusive.

```
/*
 * Read_Two_Switches_On_One_Pin
 * Read two pushbutton switches or one center-off toggle switch with one Arduino pin
 * Paul Badger 2008
 * From an idea in EDN (Electronic Design News)
 *
 * Exploits the pullup resistors available on each I/O and analog pin
 * The idea is that the 200K resistor to ground will cause the input pin to report LOW when the
 * (20K) pullup resistor is turned off, but when the pullup resistor is turned on,
 * it will overwhelm the 200K resistor and the pin will report HIGH.
 *
 * Schematic Diagram    ( can't belive I drew this funky ascii schematic )
 *
 *
 *                            +5 V
 *                               |
 *                               \
 *                               /
 *                               \    10K
 *                               /
 *                               \
 *                               |
 *                              /    switch 1  or 1/2 of center-off toggle or slide switch
 *                             /
 *                               |
 *            digital pin _______+_____________/\/\/\___________   ground
 *                               |
 *                               |               200K to 1M  (not critical)
 *                              /
 *                             /        switch 2 or 1/2 of center-off toggle or slide switch
 *                               |
 *                               |
 *                             _____
 *                              ___     ground
 *                               _
 *
 */
```

```
#define swPin 2                 // pin for input  - note: no semicolon after #define
int stateA, stateB;             // variables to store pin states
int sw1, sw2;                   // variables to represent switch states

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

void loop()
{
   digitalWrite(swPin, LOW);                   // make sure the puillup resistors are off
   stateA = digitalRead(swPin);
   digitalWrite(swPin, HIGH);                  // turn on the puillup resistors
   stateB = digitalRead(swPin);

   if ( stateA == 1 && stateB == 1 ){          // both states HIGH - switch 1 must be pushed
      sw1 = 1;
      sw2 = 0;
   }
   else if ( stateA == 0 && stateB == 0 ){     // both states LOW - switch 2 must be pushed
      sw1 = 0;
      sw2 = 1;
   }
   else{                                       // stateA HIGH and stateB LOW
      sw1 = 0;                                 // no switches pushed - or center-off toggle in middle
position
      sw2 = 0;
   }

   Serial.print(sw1);
   Serial.print("    ");     // pad some spaces to format print output
   Serial.println(sw2);

   delay(100);
}
```

**Learning** Examples | Foundations | Hacking | Links

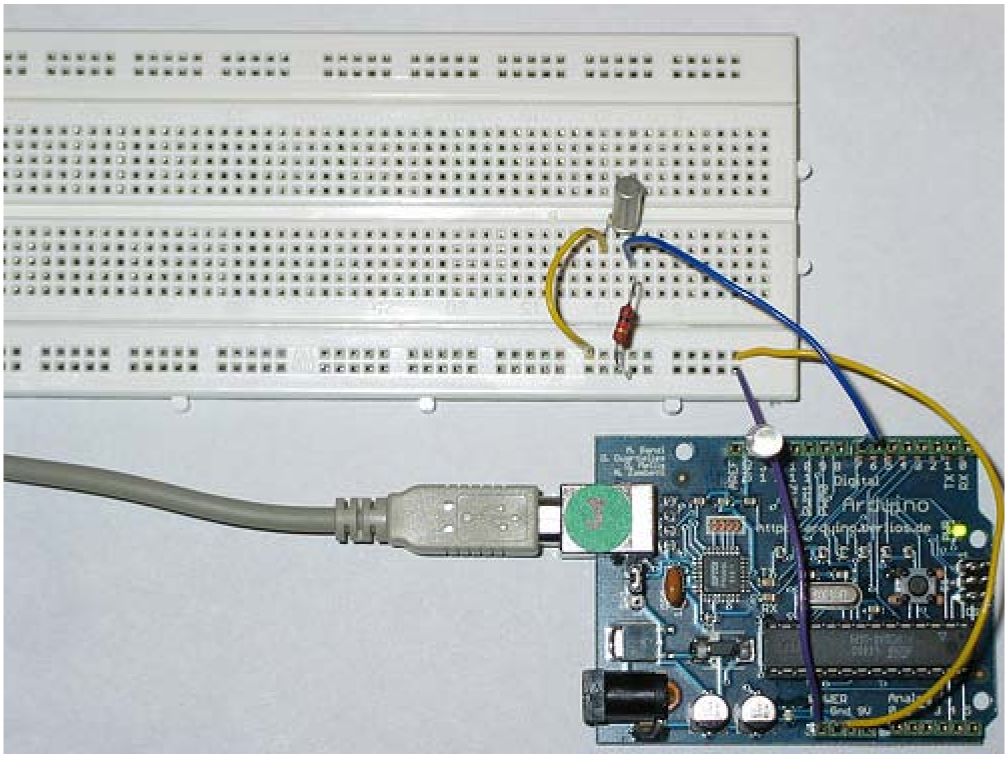
# Tilt Sensor

The tilt sensor is a component that can detect the tilting of an object. However it is only the equivalent to a pushbutton activated through a different physical mechanism. This type of sensor is the environmental-friendly version of a mercury-switch. It contains a metallic ball inside that will commute the two pins of the device from on to off and viceversa if the sensor reaches a certain angle.

The code example is exactly as the one we would use for a pushbutton but substituting this one with the tilt sensor. We use a pull-up resistor (thus use active-low to activate the pins) and connect the sensor to a digital input pin that we will read when needed.

The prototyping board has been populated with a 1K resitor to make the pull-up and the sensor itself. We have chosen the tilt sensor from Assemtech, which datasheet can be found here. The hardware was mounted and photographed by Anders Gran, the software comes from the basic Arduino examples.

## Circuit



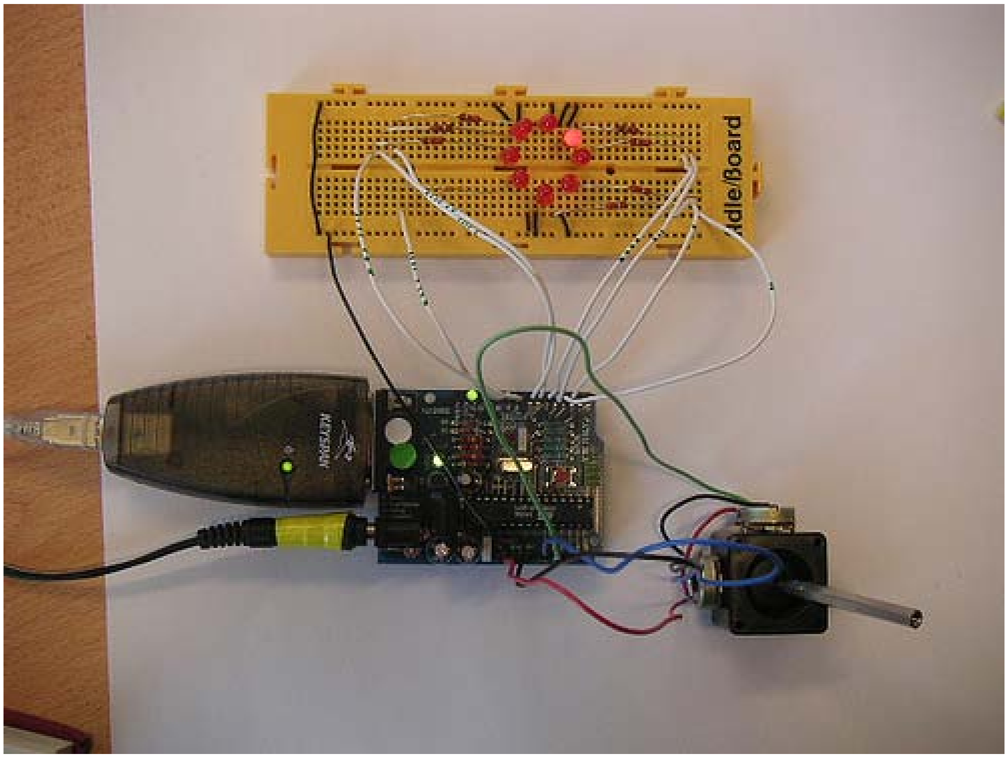
*Picture of a protoboard supporting the tilt sensor, by Anders Gran*

## Code

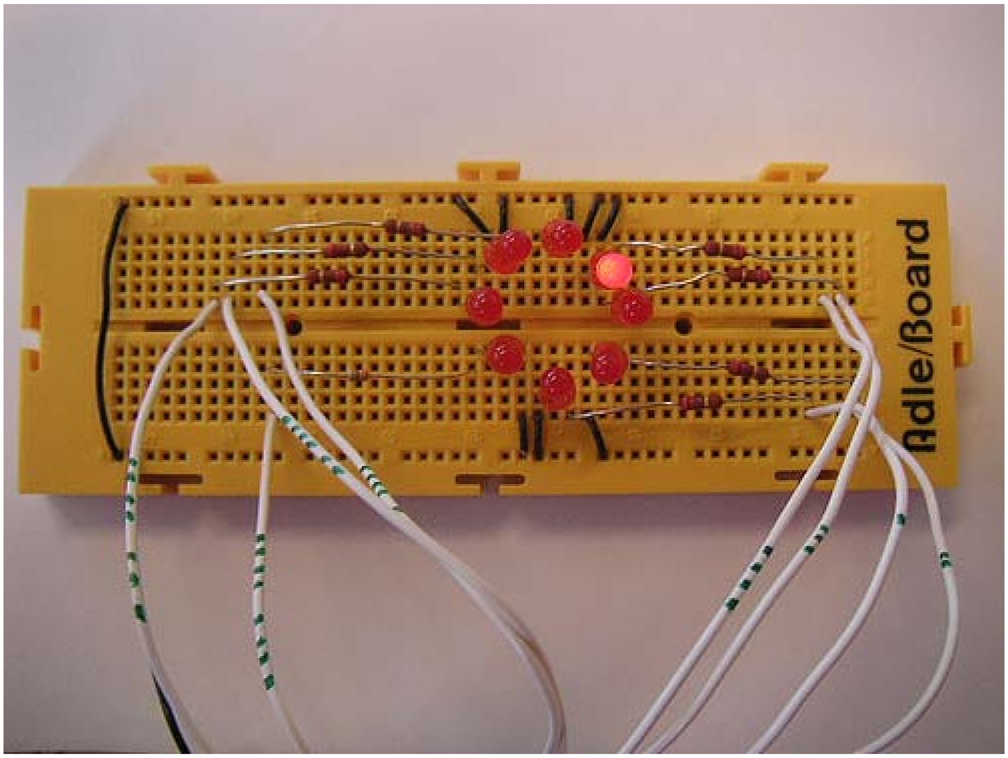
Use the **Digital > Button** example to read the tilt-sensor, but you'll need to make sure that the inputPin variable in the code matches the digital pin you're using on the Arduino board.

## Controlling a circle of LEDs with a Joystick

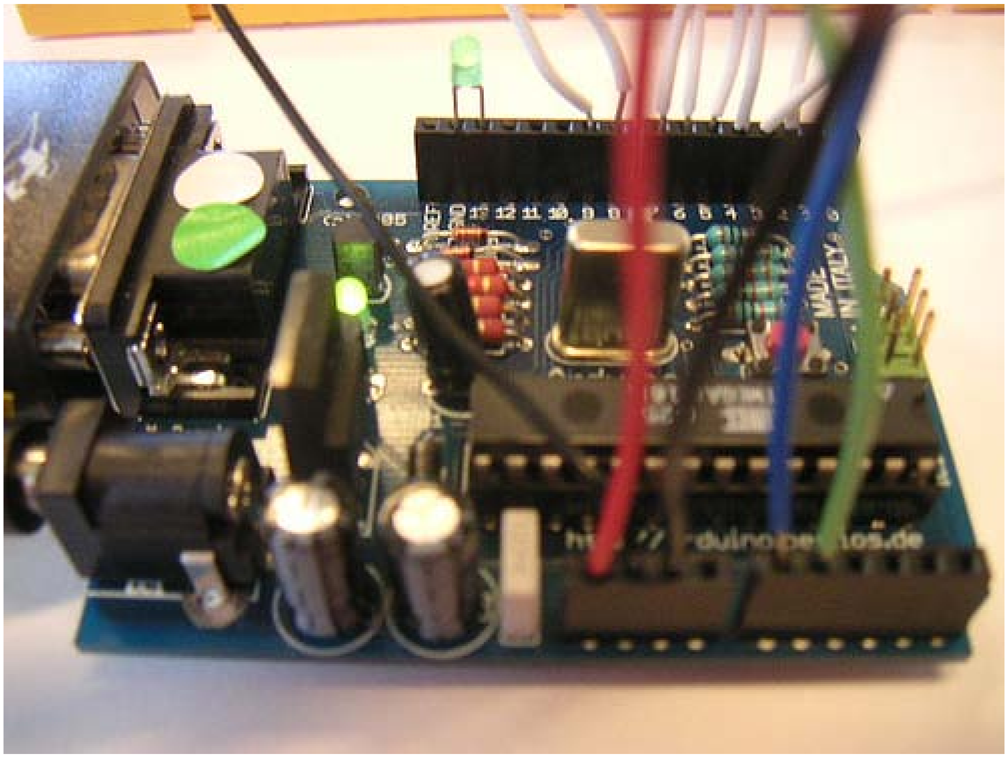
### The whole circuit:



### Detail of the LED wiring



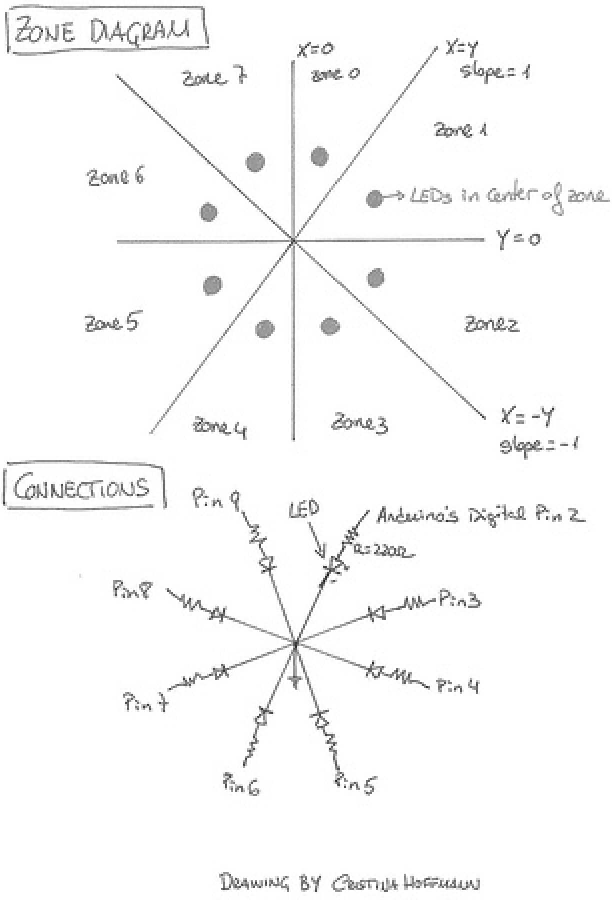
### Detail of the arduino wiring



### How this works

As you know from the **Interfacing a Joystick** tutorial, the joystick gives a coordinate (x,y) back to arduino. As you can see looking to the joystick is that the space in which he moves is a circle. This circle will be from now on our 'Pie' (see bottom right of the first image).

The only thing we need now to understand is that we have divided our Pie in 8 pieces. To each piece will correspond an LED. (See figure below). This way, when the joystick gives us a coordinate, it will necesarilly belong to one of the pies. Then, the program always lights up the LED corresponding to the pie in which the joystick is.



### Code

```
/* Controle_LEDcirle_with_joystik
 * ------------
 * This program controles a cirle of 8 LEDs through a joystick
 *
 * First it reads two analog pins that are connected
 * to a joystick made of two potentiometers
 *
 * This input is interpreted as a coordinate (x,y)
 *
 * The program then calculates to which of the 8
 * possible zones belogns the coordinate (x,y)
 *
 * Finally it ligths up the LED which is placed in the
 * detected zone
 *
 * @authors: Cristina Hoffmann and Gustavo Jose Valera
 * @hardware: Cristina Hofmann and Gustavo Jose Valera
 * @context: Arduino Workshop at medialamadrid
 */

 // Declaration of Variables

 int ledPins [] = { 2,3,4,5,6,7,8,9 };    // Array of 8 leds mounted in a circle
 int ledVerde = 13;
 int espera = 40;                 // Time you should wait for turning on the leds
 int joyPin1 = 0;                 // slider variable connecetd to analog pin 0
 int joyPin2 = 1;                 // slider variable connecetd to analog pin 1
 int coordX = 0;                  // variable to read the value from the analog pin 0
 int coordY = 0;                  // variable to read the value from the analog pin 1
```

```
  int centerX = 500;               // we measured the value for the center of the joystick
  int centerY = 500;
  int actualZone = 0;
  int previousZone = 0;

  // Asignment of the pins
  void setup()
  {
    int i;
    beginSerial(9600);
    pinMode (ledVerde, OUTPUT);
    for (i=0; i< 8; i++)
    {
      pinMode(ledPins[i], OUTPUT);
    }
  }

  // function that calculates the slope of the line that passes through the points
  // x1, y1 and x2, y2
  int calculateSlope(int x1, int y1, int x2, int y2)
  {
    return ((y1-y2) / (x1-x2));
  }

  // function that calculates in which of the 8 possible zones is the coordinate x y, given the center cx,
cy
  int calculateZone (int x, int y, int cx, int cy)
  {
    int alpha = calculateSlope(x,y, cx,cy); // slope of the segment betweent the point and the center

    if (x > cx)
    {
      if (y > cy) // first cuadrant
      {
        if (alpha > 1) // The slope is > 1, thus higher part of the first quadrant
          return 0;
        else
          return 1;    // Otherwise the point is in the lower part of the first quadrant
      }
      else // second cuadrant
      {
        if (alpha > -1)
          return 2;
        else
          return 3;
      }
    }

    else
    {
      if (y < cy) // third cuadrant
      {
        if (alpha > 1)
          return 4;
        else
          return 5;
      }
      else // fourth cuadrant
      {
        if (alpha > -1)
          return 6;
        else
          return 7;
      }
    }
  }
  void loop() {
   digitalWrite(ledVerde, HIGH); // flag to know we entered the loop, you can erase this if you want

   // reads the value of the variable resistors
   coordX = analogRead(joyPin1);
   coordY = analogRead(joyPin2);

   // We calculate in which x
   actualZone = calculateZone(coordX, coordY, centerX, centerY);

   digitalWrite (ledPins[actualZone], HIGH);

   if (actualZone != previousZone)
     digitalWrite (ledPins[previousZone], LOW);

  // we print int the terminal, the cartesian value of the coordinate, and the zone where it belongs.
 //This is not necesary for a standalone version
   serialWrite('C');
   serialWrite(32); // print space
   printInteger(coordX);
   serialWrite(32); // print space
   printInteger(coordY);
   serialWrite(10);
   serialWrite(13);

   serialWrite('Z');
   serialWrite(32); // print space
   printInteger(actualZone);
   serialWrite(10);
   serialWrite(13);

 // But this is necesary so, don't delete it!
   previousZone = actualZone;
  // delay (500);

}
```

@idea: Cristina Hoffmann and Gustavo Jose Valera

@code: Cristina Hoffmann and Gustavo Jose Valera

@pictures and graphics: Cristina Hoffmann

@date: 20051008 - Madrid - Spain

```
/*
* "Coffee-cup" Color Mixer:
* Code for mixing and reporting PWM-mediated color
* Assumes Arduino 0004 or higher, as it uses Serial.begin()-style communication
*
* Control 3 LEDs with 3 potentiometers
* If the LEDs are different colors, and are directed at diffusing surface (stuck in a
*   a Ping-Pong ball, or placed in a paper coffee cup with a cut-out bottom and
*   a white plastic lid), the colors will mix together.
*
* When you mix a color you like, stop adjusting the pots.
* The mix values that create that color will be reported via serial out.
*
* Standard colors for light mixing are Red, Green, and Blue, though you can mix
*   with any three colors; Red + Blue + White would let you mix shades of red,
*   blue, and purple (though no yellow, orange, green, or blue-green.)
*
* Put 220 Ohm resistors in line with pots, to prevent circuit from
*   grounding out when the pots are at zero
*/

// Analog pin settings
int aIn = 0;    // Potentiometers connected to analog pins 0, 1, and 2
int bIn = 1;    //   (Connect power to 5V and ground to analog ground)
int cIn = 2;

// Digital pin settings
int aOut = 9;   // LEDs connected to digital pins 9, 10 and 11
int bOut = 10;  //   (Connect cathodes to digital ground)
int cOut = 11;

// Values
int aVal = 0;   // Variables to store the input from the potentiometers
int bVal = 0;
int cVal = 0;

// Variables for comparing values between loops
int i = 0;            // Loop counter
int wait = (1000);    // Delay between most recent pot adjustment and output

int checkSum     = 0; // Aggregate pot values
int prevCheckSum = 0;
int sens         = 3; // Sensitivity theshold, to prevent small changes in
                     // pot values from triggering false reporting
// FLAGS
int PRINT = 1; // Set to 1 to output values
int DEBUG = 1; // Set to 1 to turn on debugging output

void setup()
{
  pinMode(aOut, OUTPUT);   // sets the digital pins as output
  pinMode(bOut, OUTPUT);
  pinMode(cOut, OUTPUT);
  Serial.begin(9600);     // Open serial communication for reporting
}

void loop()
{
  i += 1; // Count loop

  aVal = analogRead(aIn) / 4;  // read input pins, convert to 0-255 scale
  bVal = analogRead(bIn) / 4;
  cVal = analogRead(cIn) / 4;

  analogWrite(aOut, aVal);    // Send new values to LEDs
  analogWrite(bOut, bVal);
  analogWrite(cOut, cVal);

  if (i % wait == 0)                // If enough time has passed...
  {
    checkSum = aVal+bVal+cVal;      // ...add up the 3 values.
    if ( abs(checkSum - prevCheckSum) > sens )   // If old and new values differ
                                                 // above sensitivity threshold
    {
      if (PRINT)                    // ...and if the PRINT flag is set...
      {
        Serial.print("A: ");        // ...then print the values.
        Serial.print(aVal);
        Serial.print("\t");
        Serial.print("B: ");
        Serial.print(bVal);
        Serial.print("\t");
        Serial.print("C: ");
        Serial.println(cVal);
        PRINT = 0;
      }
    }
    else
    {
      PRINT = 1;  // Re-set the flag
    }
    prevCheckSum = checkSum;  // Update the values

    if (DEBUG)   // If we want debugging output as well...
    {
      Serial.print(checkSum);
      Serial.print("<=>");
      Serial.print(prevCheckSum);
      Serial.print("\tPrint: ");
      Serial.println(PRINT);
    }
  }
}
```

## Stopwatch

A sketch that demonstrates how to do two (or more) things at once by using millis().

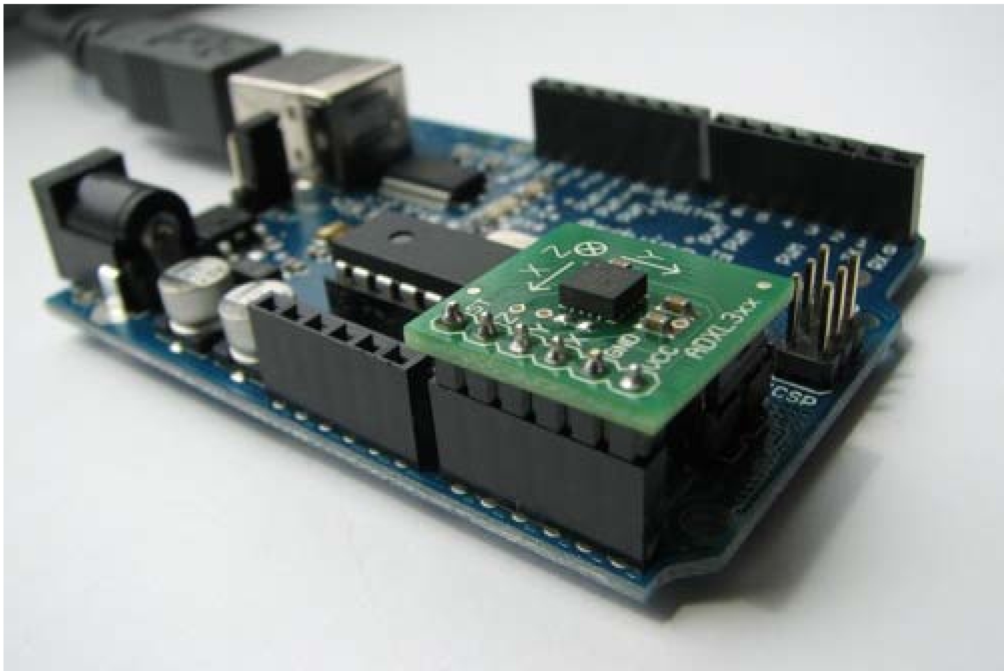
```
/* StopWatch
 * Paul Badger 2008
 * Demonstrates using millis(), pullup resistors,
 * making two things happen at once, printing fractions
 *
 * Physical setup: momentary switch connected to pin 4, other side connected to ground
 * LED with series resistor between pin 13 and ground
 */


#define ledPin  13                  // LED connected to digital pin 13
#define buttonPin 4                 // button on pin 4

int value = LOW;                    // previous value of the LED
int buttonState;                    // variable to store button state
int lastButtonState;                // variable to store last button state
int blinking;                       // condition for blinking - timer is timing
long interval = 100;                // blink interval - change to suit
long previousMillis = 0;            // variable to store last time LED was updated
long startTime ;                    // start time for stop watch
long elapsedTime ;                  // elapsed time for stop watch
int fractional;                     // variable used to store fractional part of time



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

   pinMode(ledPin, OUTPUT);         // sets the digital pin as output

   pinMode(buttonPin, INPUT);       // not really necessary, pins default to INPUT anyway
   digitalWrite(buttonPin, HIGH);   // turn on pullup resistors. Wire button so that press shorts pin to
ground.

}

void loop()
{
    // check for button press
   buttonState = digitalRead(buttonPin);                  // read the button state and store

   if (buttonState == LOW && lastButtonState == HIGH  &&  blinking == false){     // check for a high to
low transition
      // if true then found a new button press while clock is not running - start the clock

      startTime = millis();                                   // store the start time
      blinking = true;                                        // turn on blinking while timing
      delay(5);                                              // short delay to debounce switch
      lastButtonState = buttonState;                          // store buttonState in lastButtonState, to
compare next time

   }

   else if (buttonState == LOW && lastButtonState == HIGH && blinking == true){     // check for a high to
low transition
      // if true then found a new button press while clock is running - stop the clock and report

      elapsedTime =   millis() - startTime;                // store elapsed time
      blinking = false;                                       // turn off blinking, all done timing
      lastButtonState = buttonState;                          // store buttonState in lastButtonState, to
compare next time



         // routine to report elapsed time - this breaks when delays are in single or double digits. Fix
this as a coding exercise.

         Serial.print( (int)(elapsedTime / 1000L) );         // divide by 1000 to convert to seconds - then
cast to an int to print
      Serial.print(".");                                     // print decimal point
      fractional = (int)(elapsedTime % 1000L);               // use modulo operator to get fractional part
of time
      Serial.println(fractional);                            // print fractional part of time

   }

   else{
      lastButtonState = buttonState;                         // store buttonState in lastButtonState, to
compare next time
   }

   // blink routine - blink the LED while timing
   // check to see if it's time to blink the LED; that is, is the difference
   // between the current time and last time we blinked the LED bigger than
   // the interval at which we want to blink the LED.

   if ( (millis() - previousMillis > interval) ) {

      if (blinking == true){
         previousMillis = millis();                         // remember the last time we blinked the LED

         // if the LED is off turn it on and vice-versa.
         if (value == LOW)
            value = HIGH;
         else
            value = LOW;
         digitalWrite(ledPin, value);
      }
      else{
         digitalWrite(ledPin, LOW);                         // turn off LED when not blinking
      }
   }

}
```

*Examples > Analog I/O*

## ADXL3xx Accelerometer

Reads an Analog Devices ADXL3xx series (e.g. ADXL320, ADXL321, ADXL322, ADXL330) accelerometer and communicates the acceleration to the computer. The pins used are designed to be easily compatible with the breakout boards from Sparkfun. The ADXL3xx outputs the acceleration on each axis as an analog voltage between 0 and 5 volts, which is read by an analog input on the Arduino.

### Circuit



*An ADXL322 on a Sparkfun breakout board inserted into the analog input pins of an Arduino.*

Pinout for the above configuration:

| | | | | | | |
|---|---|---|---|---|---|---|
| **Breakout Board Pin** | Self-Test | Z-Axis | Y-Axis | X-Axis | Ground | VDD |
| **Arduino Analog Input Pin** | 0 | 1 | 2 | 3 | 4 | 5 |

Or, if you're using just the accelerometer:

| | | | | | | |
|---|---|---|---|---|---|---|
| **ADXL3xx Pin** | Self-Test | ZOut | YOut | XOut | Ground | VDD |
| **Arduino Pin** | *None* (unconnected) | Analog Input 1 | Analog Input 2 | Analog Input 3 | GND | 5V |

### Code

```
int groundpin = 18;             // analog input pin 4
int powerpin = 19;              // analog input pin 5
int xpin = 3;                   // x-axis of the accelerometer
int ypin = 2;                   // y-axis
int zpin = 1;                   // z-axis (only on 3-axis models)
```

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

  // Provide ground and power by using the analog inputs as normal
  // digital pins.  This makes it possible to directly connect the
  // breakout board to the Arduino.  If you use the normal 5V and
  // GND pins on the Arduino, you can remove these lines.
  pinMode(groundPin, OUTPUT);
  pinMode(powerPin, OUTPUT);
  digitalWrite(groundPin, LOW);
  digitalWrite(powerPin, HIGH);
}

void loop()
{
  Serial.print(analogRead(xpin));
  Serial.print(" ");
  Serial.print(analogRead(ypin));
  Serial.print(" ");
  Serial.print(analogRead(zpin));
  Serial.println();
  delay(1000);
}
```

## Data

Here are some accelerometer readings collected by the positioning the y-axis of an ADXL322 2g accelerometer at various angles from ground. Values should be the same for the other axes, but will vary based on the sensitivity of the device. With the axis horizontal (i.e. parallel to ground or 0°), the accelerometer reading should be around 512, but values at other angles will be different for a different accelerometer (e.g. the ADXL302 5g one).

| **Angle** | -90 | -80 | -70 | -60 | -50 | -40 | -30 | -20 | -10 | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Acceleration** | 662 | 660 | 654 | 642 | 628 | 610 | 589 | 563 | 537 | 510 | 485 | 455 | 433 | 408 | 390 | 374 | 363 | 357 | 355 |

# Arduino

search

Buy | Download | Getting Started | Learning | Reference | Hardware | FAQ Blog » | Forum » | Playground »

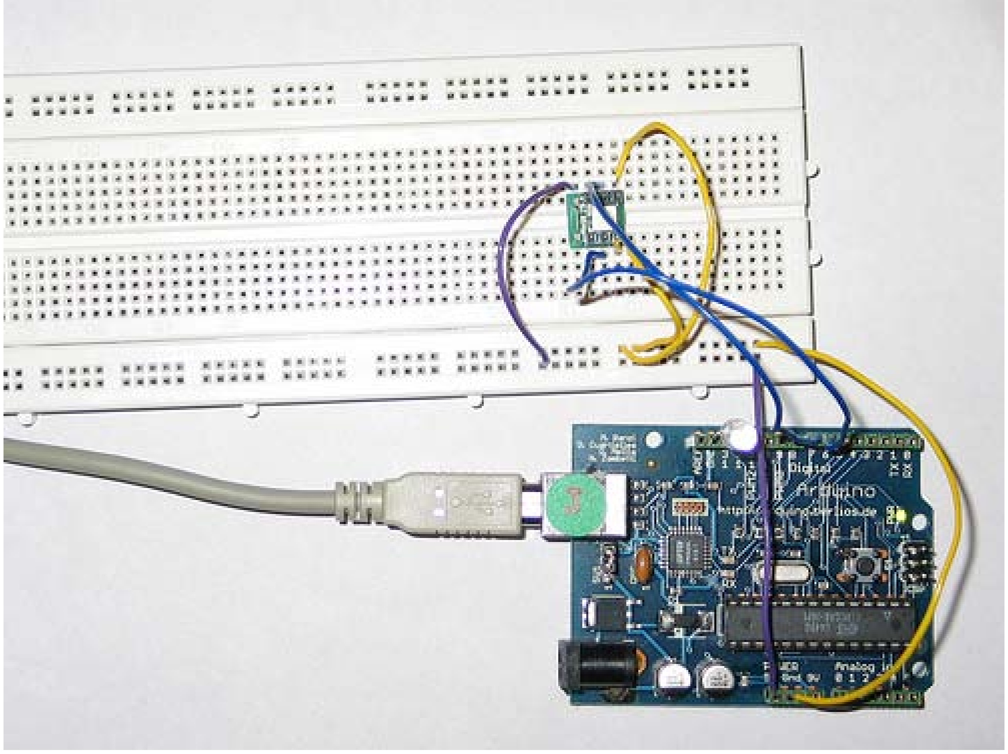
**Learning** Examples | Foundations | Hacking | Links

## Memsic 2125 Accelerometer

The Memsic 2125 is a dual axis accelerometer sensor from Parallax able of measuring up to a 2g acceleration. When making very accurate measurements, the sensor counts with a temperature pin that can be used to compensate possible errors.

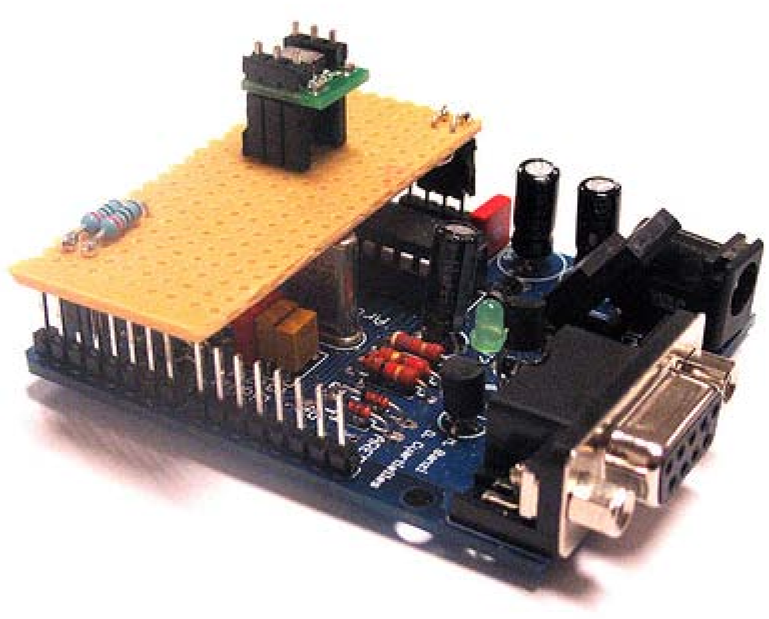
The pins dedicated to measure acceleration can be connected directly to digital inputs to the Arduino board, while the the temperature should be taken as an analog input. The acceleration pins send the signals back to the computer in the form of pulses which width represents the acceleration.

The example shown here was mounted by Anders Gran, while the software was created by Marcos Yarza, who is Arduino's accelerometer technology researcher, at the University of Zaragoza, Spain. The board is connected minimally, only the two axis pins are plugged to the board, leaving the temperature pin open.



*Protoboard with an Accelerometer, picture by Anders Gran*

```
 /* Accelerometer Sensor
* --------------------
*
* Reads an 2-D accelerometer
* attached to a couple of digital inputs and
* sends their values over the serial port; makes
* the monitor LED blink once sent
*
*
* http://www.0j0.org
* copyleft 2005 K3 - Malmo University - Sweden
* @author: Marcos Yarza
* @hardware: Marcos Yarza
* @project: SMEE - Experiential Vehicles
* @sponsor: Experiments in Art and Technology Sweden, 1:1 Scale
*/

int ledPin = 13;
int xaccPin = 7;
int yaccPin = 6;
int value = 0;
int accel = 0;
char sign = ' ';

int timer = 0;
int count = 0;

void setup() {
beginSerial(9600); // Sets the baud rate to 9600
pinMode(ledPin, OUTPUT);
pinMode(xaccPin, INPUT);
pinMode(yaccPin, INPUT);
}

/* (int) Operate Acceleration
* function to calculate acceleration
* returns an integer
*/
int operateAcceleration(int time1) {
return abs(8 * (time1 / 10 - 500));
}

/* (void) readAccelerometer
* procedure to read the sensor, calculate
* acceleration and represent the value
*/
void readAcceleration(int axe){
timer = 0;
count = 0;
value = digitalRead(axe);
while(value == HIGH) { // Loop until pin reads a low
value = digitalRead(axe);
}
while(value == LOW) { // Loop until pin reads a high
value = digitalRead(axe);
}
while(value == HIGH) { // Loop until pin reads a low and count
value = digitalRead(axe);
count = count + 1;
}
timer = count * 18; //calculate the teme in miliseconds

//operate sign
if (timer > 5000){
sign = '+';
}
if (timer < 5000){
sign = '-';
}

//determine the value
accel = operateAcceleration(timer);

//Represent acceleration over serial port
if (axe == 7){
printByte('X');
}
else {
printByte('Y');
}
printByte(sign);
printInteger(accel);
printByte(' ');

}
void loop() {
readAcceleration(xaccPin); //reads and represents acceleration X
readAcceleration(yaccPin); //reads and represents acceleration Y
digitalWrite(ledPin, HIGH);
delay(300);
digitalWrite(ledPin, LOW);
}
```



*Accelerometer mounted on prototyping board, by M. Yarza*

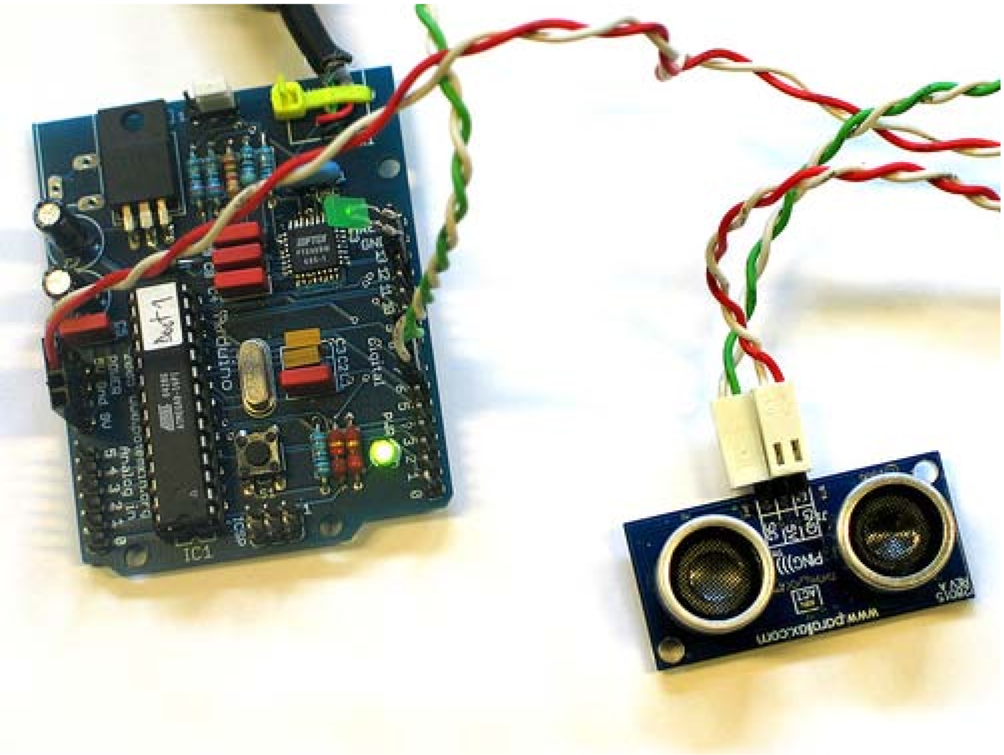
The following example is an adaptation of the previous one. Marcos Yarza added two 220Ohm resistors to the pins coming out of the accelerometer. The board chosen for this small circuit is just a piece of prototyping board. Here the code is exactly the same as before (changing the input pins to be 2 and 3), but the installation on the board allows to embed the whole circutry in a much smaller housing.

## PING range finder

The PING range finder is an ultrasound sensor from Parallax able of detecting objects up to a 3 mts distance. The sensor counts with 3 pins, two are dedicated to power and ground, while the third one is used both as input and output.

The pin dedicated to make the readings has to be shifting configuration from input to output according to the PING specification sheet. First we have to send a pulse that will make the sensor send an ultrasound tone and wait for an echo. Once the tone is received back, the sensor will send a pulse over the same pin as earlier. The width of that pulse will determine the distance to the object.

The example shown here was mounted by Marcus Hannerstig, while the software was created by David Cuartielles. The board is connected as explained using only wires coming from an old computer.



*Ultrasound sensor connected to an Arduino USB v1.0*

```
/* Ultrasound Sensor
 *------------------
 *
 * Reads values (00014-01199) from an ultrasound sensor (3m sensor)
 * and writes the values to the serialport.
 *
 * http://www.xlab.se | http://www.0j0.org
 * copyleft 2005 Mackie for XLAB | DojoDave for DojoCorp
 *
 */

int ultraSoundSignal = 7; // Ultrasound signal pin
int val = 0;
int ultrasoundValue = 0;
int timecount = 0; // Echo counter
int ledPin = 13; // LED connected to digital pin 13

void setup() {
  beginSerial(9600);                 // Sets the baud rate to 9600
  pinMode(ledPin, OUTPUT);           // Sets the digital pin as output
}

void loop() {
 timecount = 0;
 val = 0;
 pinMode(ultraSoundSignal, OUTPUT); // Switch signalpin to output

/* Send low-high-low pulse to activate the trigger pulse of the sensor
 * ---------------------------------------------------------------------------
 */

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

/* Listening for echo pulse
 * ---------------------------------------------------------------------------
 */

pinMode(ultraSoundSignal, INPUT); // Switch signalpin to input
val = digitalRead(ultraSoundSignal); // Append signal value to val
while(val == LOW) { // Loop until pin reads a high value
  val = digitalRead(ultraSoundSignal);
}

while(val == HIGH) { // Loop until pin reads a high value
  val = digitalRead(ultraSoundSignal);
  timecount = timecount +1;            // Count echo pulse time
}

/* Writing out values to the serial port
 * ---------------------------------------------------------------------------
 */

ultrasoundValue = timecount; // Append echo pulse time to ultrasoundValue

serialWrite('A'); // Example identifier for the sensor
printInteger(ultrasoundValue);
serialWrite(10);
serialWrite(13);

/* Lite up LED if any value is passed by the echo pulse
 * ---------------------------------------------------------------------------
 */

if(timecount > 0){
  digitalWrite(ledPin, HIGH);
}

/* Delay of program
 * ---------------------------------------------------------------------------
 */

delay(100);
}
```

## qt401 sensor

full tutorial coming soon

```
/* qt401 demo
 * ------------
 *
 * the qt401 from qprox http://www.qprox.com is a linear capacitive sensor
 * that is able to read the position of a finger touching the sensor
 * the surface of the sensor is divided in 128 positions
 * the pin qt401_prx detects when a hand is near the sensor while
 * qt401_det determines when somebody is actually touching the sensor
 * these can be left unconnected if you are short of pins
 *
 * read the datasheet to understand the parametres passed to initialise the sensor
 *
 * Created January 2006
 * Massimo Banzi http://www.potemkin.org
 *
 * based on C code written by Nicholas Zambetti
 */


// define pin mapping
int qt401_drd = 2; // data ready
int qt401_di  = 3; // data in (from sensor)
int qt401_ss  = 4; // slave select
int qt401_clk = 5; // clock
int qt401_do  = 6; // data out (to sensor)
int qt401_det = 7; // detect
int qt401_prx = 8; // proximity


byte result;

void qt401_init() {
  // define pin directions
  pinMode(qt401_drd, INPUT);
  pinMode(qt401_di,  INPUT);
  pinMode(qt401_ss,  OUTPUT);
  pinMode(qt401_clk, OUTPUT);
  pinMode(qt401_do,  OUTPUT);
  pinMode(qt401_det, INPUT);
  pinMode(qt401_prx, INPUT);


  // initialise pins
  digitalWrite(qt401_clk,HIGH);
  digitalWrite(qt401_ss, HIGH);
}

//
```

```
//  wait for the qt401 to be ready
//
void qt401_waitForReady(void)
{
  while(!digitalRead(qt401_drd)){
    continue;
  }
}



//
//  exchange a byte with the sensor
//

byte qt401_transfer(byte data_out)
{
  byte i = 8;

  byte mask = 0;
  byte data_in = 0;

  digitalWrite(qt401_ss,LOW); // select slave by lowering ss pin
  delayMicroseconds(75); //wait for 75 microseconds

  while(0 < i) {

    mask = 0x01 << --i; // generate bitmask for the appropriate bit MSB first

    // set out byte
    if(data_out & mask){ // choose bit

      digitalWrite(qt401_do,HIGH); // send 1

    }
    else{

      digitalWrite(qt401_do,LOW); // send 0

    }

    // lower clock pin, this tells the sensor to read the bit we just put out
    digitalWrite(qt401_clk,LOW); // tick

    // give the sensor time to read the data

    delayMicroseconds(75);

    // bring clock back up

    digitalWrite(qt401_clk,HIGH); // tock

    // give the sensor some time to think

    delayMicroseconds(20);

    // now read a bit coming from the sensor
    if(digitalRead(qt401_di)){

      data_in |= mask;

    }

    //  give the sensor some time to think
```

```
      delayMicroseconds(20);

  }

  delayMicroseconds(75); //  give the sensor some time to think
  digitalWrite(qt401_ss,HIGH); // do acquisition burst

  return data_in;
}

void qt401_calibrate(void)
{
  // calibrate
  qt401_waitForReady();
  qt401_transfer(0x01);
  delay(600);

  // calibrate ends
  qt401_waitForReady();
  qt401_transfer(0x02);
  delay(600);
}


void qt401_setProxThreshold(byte amount)
{
  qt401_waitForReady();
  qt401_transfer(0x40 & (amount & 0x3F));
}


void qt401_setTouchThreshold(byte amount)
{
  qt401_waitForReady();
  qt401_transfer(0x80 & (amount & 0x3F));
}


byte qt401_driftCompensate(void)
{
  qt401_waitForReady();
  return qt401_transfer(0x03);
}


byte qt401_readSensor(void)
{

  qt401_waitForReady();
  return qt401_transfer(0x00);
}


void setup() {

  //setup the sensor
  qt401_init();
  qt401_calibrate();
  qt401_setProxThreshold(10);
  qt401_setTouchThreshold(10);

  beginSerial(9600);

}
```

```
void loop() {

  if(digitalRead(qt401_det)){
    result = qt401_readSensor();
    if(0x80 & result){
      result = result & 0x7f;

      printInteger(result);
      printNewline();

    }
  }

}
```

# Arduino

search

Buy | Download | Getting Started | Learning | Reference | Hardware | FAQ     Blog » | Forum » | Playground »

**Learning** Examples | Foundations | Hacking | Links

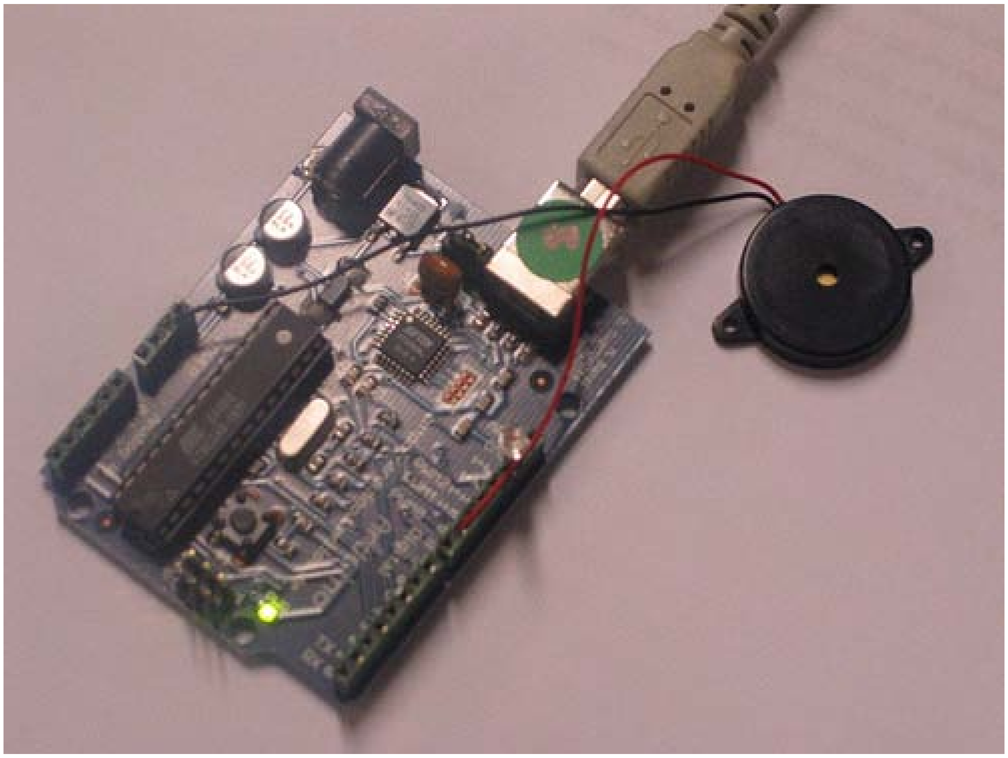
## Play Melody

This example makes use of a Piezo Speaker in order to play melodies. We are taking advantage of the processors capability to produde PWM signals in order to play music. There is more information about how PWM works written by David Cuartielles here and even at K3's old course guide

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the pin number 9, that supports the functionality of writing a PWM signal to it, and not just a plain HIGH or LOW value.

The first example of the code will just send a square wave to the piezo, while the second one will make use of the PWM functionality to control the volume through changing the Pulse Width.

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the output. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc.



*Example of connection of a Piezo to pin 9*

### Example 1: Play Melody

```
/* Play Melody
 * -----------
 *
 * Program to play a simple melody
 *
 * Tones are created by quickly pulsing a speaker on and off
 *   using PWM, to create signature frequencies.
 *
 * Each note has a frequency, created by varying the period of
 *  vibration, measured in microseconds. We'll use pulse-width
 *  modulation (PWM) to create that vibration.

 * We calculate the pulse-width to be half the period; we pulse
 *  the speaker HIGH for 'pulse-width' microseconds, then LOW
 *  for 'pulse-width' microseconds.
 *  This pulsing creates a vibration of the desired frequency.
 *
 * (cleft) 2005 D. Cuartielles for K3
 * Refactoring and comments 2006 clay.shirky@nyu.edu
 * See NOTES in comments at end for possible improvements
 */

// TONES  ==========================================
// Start by defining the relationship between
//       note, period, &  frequency.
#define  c     3830    // 261 Hz
#define  d     3400    // 294 Hz
#define  e     3038    // 329 Hz
#define  f     2864    // 349 Hz
#define  g     2550    // 392 Hz
#define  a     2272    // 440 Hz
#define  b     2028    // 493 Hz
#define  C     1912    // 523 Hz
// Define a special note, 'R', to represent a rest
#define  R     0

// SETUP ============================================
// Set up speaker on a PWM pin (digital 9, 10 or 11)
int speakerOut = 9;
// Do we want debugging on serial out? 1 for yes, 0 for no
int DEBUG = 1;

void setup() {
  pinMode(speakerOut, OUTPUT);
  if (DEBUG) {
    Serial.begin(9600); // Set serial out if we want debugging
  }
}

// MELODY and TIMING  =======================================
//  melody[] is an array of notes, accompanied by beats[],
//  which sets each note's relative length (higher #, longer note)
int melody[] = {  C,  b,  g,  C,  b,   e,  R,  C,  c,  g, a, C };
int beats[]  = { 16, 16, 16,  8,  8,  16, 32, 16, 16, 16, 8, 8 };
int MAX_COUNT = sizeof(melody) / 2; // Melody length, for looping.

// Set overall tempo
long tempo = 10000;
// Set length of pause between notes
int pause = 1000;
// Loop variable to increase Rest length
int rest_count = 100; //<-BLETCHEROUS HACK; See NOTES

// Initialize core variables
int tone = 0;
int beat = 0;
long duration  = 0;

// PLAY TONE  ==============================================
// Pulse the speaker to play a tone for a particular duration
void playTone() {
  long elapsed_time = 0;
  if (tone > 0) { // if this isn't a Rest beat, while the tone has
    //  played less long than 'duration', pulse speaker HIGH and LOW
    while (elapsed_time < duration) {

      digitalWrite(speakerOut,HIGH);
      delayMicroseconds(tone / 2);

      // DOWN
      digitalWrite(speakerOut, LOW);
      delayMicroseconds(tone / 2);

      // Keep track of how long we pulsed
      elapsed_time += (tone);
    }
  }
  else { // Rest beat; loop times delay
    for (int j = 0; j < rest_count; j++) { // See NOTE on rest_count
      delayMicroseconds(duration);
    }
  }
}

// LET THE WILD RUMPUS BEGIN =============================
void loop() {
  // Set up a counter to pull from melody[] and beats[]
  for (int i=0; i<MAX_COUNT; i++) {
    tone = melody[i];
    beat = beats[i];

    duration = beat * tempo; // Set up timing

    playTone();
    // A pause between notes...
    delayMicroseconds(pause);

    if (DEBUG) { // If debugging, report loop, tone, beat, and duration
      Serial.print(i);
      Serial.print(":");
      Serial.print(beat);
      Serial.print(" ");
      Serial.print(tone);
      Serial.print(" ");
      Serial.println(duration);
    }
  }
}

/*
 * NOTES
 * The program purports to hold a tone for 'duration' microseconds.
 *  Lies lies lies! It holds for at least 'duration' microseconds, _plus_
 *  any overhead created by incremeting elapsed_time (could be in excess of
 *  3K microseconds) _plus_ overhead of looping and two digitalWrites()
 *
 * As a result, a tone of 'duration' plays much more slowly than a rest
 *  of 'duration.' rest_count creates a loop variable to bring 'rest' beats
 *  in line with 'tone' beats of the same length.
 *
 * rest_count will be affected by chip architecture and speed, as well as
 *  overhead from any program mods. Past behavior is no guarantee of future
 *  performance. Your mileage may vary. Light fuse and get away.
 *
 * This could use a number of enhancements:
 * ADD code to let the programmer specify how many times the melody should
 *     loop before stopping
 * ADD another octave
 * MOVE tempo, pause, and rest_count to #define statements
 * RE-WRITE to include volume, using analogWrite, as with the second program at
 *          http://www.arduino.cc/en/Tutorial/PlayMelody
 * ADD code to make the tempo settable by pot or other input device
 * ADD code to take tempo or volume settable by serial communication
 *          (Requires 0005 or higher.)
 * ADD code to create a tone offset (higer or lower) through pot etc
 * REPLACE random melody with opening bars to 'Smoke on the Water'
 */
```

Second version, with volume control set using analogWrite()

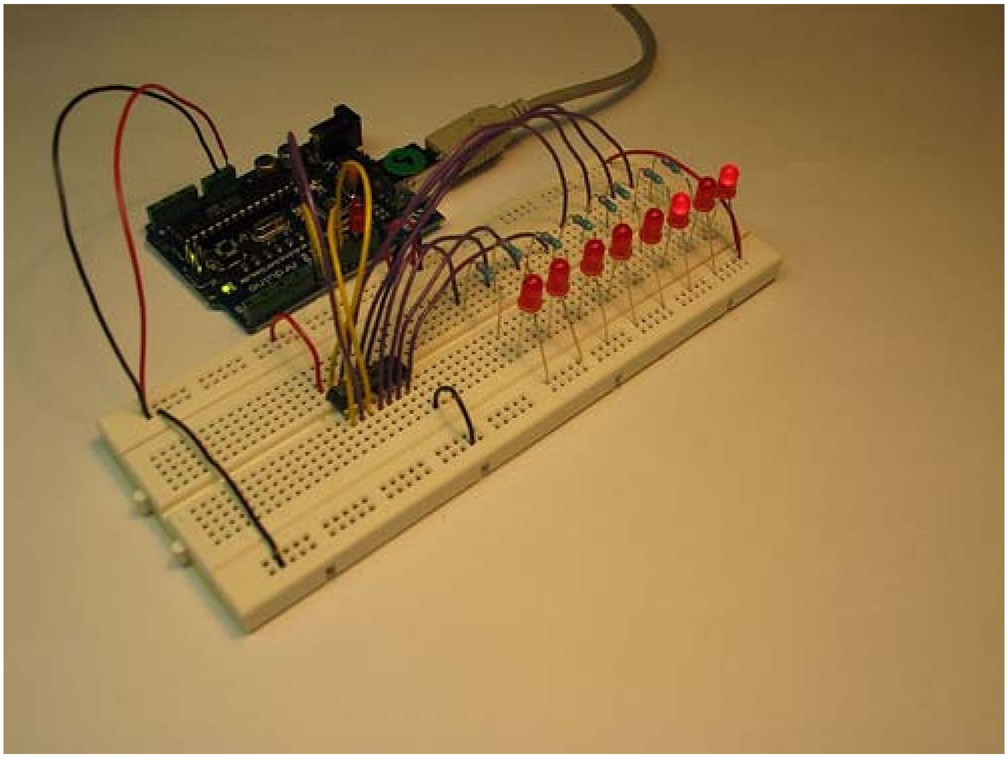
```
/* Play Melody
 * -----------
 *
 * Program to play melodies stored in an array, it requires to know
 * about timing issues and about how to play tones.
 *
 * The calculation of the tones is made following the mathematical
 * operation:
 *
 *       timeHigh = 1/(2 * toneFrequency) = period / 2
 *
 * where the different tones are described as in the table:
 *
 * note         frequency       period  PW (timeHigh)
 * c            261 Hz          3830    1915
 * d            294 Hz          3400    1700
 * e            329 Hz          3038    1519
 * f            349 Hz          2864    1432
 * g            392 Hz          2550    1275
 * a            440 Hz          2272    1136
 * b            493 Hz          2028    1014
 * C            523 Hz          1912    956
 *
 * (cleft) 2005 D. Cuartielles for K3
 */

int ledPin = 13;
int speakerOut = 9;
byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};
byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p";
// count length:1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//                              10                  20                  30
int count = 0;
int count2 = 0;
int count3 = 0;
int MAX_COUNT = 24;
int statePin = LOW;

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

void loop() {
  analogWrite(speakerOut, 0);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          analogWrite(speakerOut,500);
          delayMicroseconds(tones[count2]);
          analogWrite(speakerOut, 0);
          delayMicroseconds(tones[count2]);
        }
        if (melody[count*2 + 1] == 'p') {
          // make a pause of a certain size
          analogWrite(speakerOut, 0);
          delayMicroseconds(500);
        }
      }
    }
  }
}
```

## LED Driver

This example makes use of an LED Driver in order to control an almost endless amount of LEDs with only 4 pins. We use the 4794 from Philips. There is more information about this microchip that you will find in its datasheet.

An LED Driver has a shift register embedded that will take data in serial format and transfer it to parallel. It is possible to daisy chain this chip increasing the total amount of LEDs by 8 each time.

The code example you will see here is taking a value stored in the variable **dato** and showing it as a decoded binary number. E.g. if dato is 1, only the first LED will light up; if dato is 255 all the LEDs will light up.



*Example of connection of a 4794*

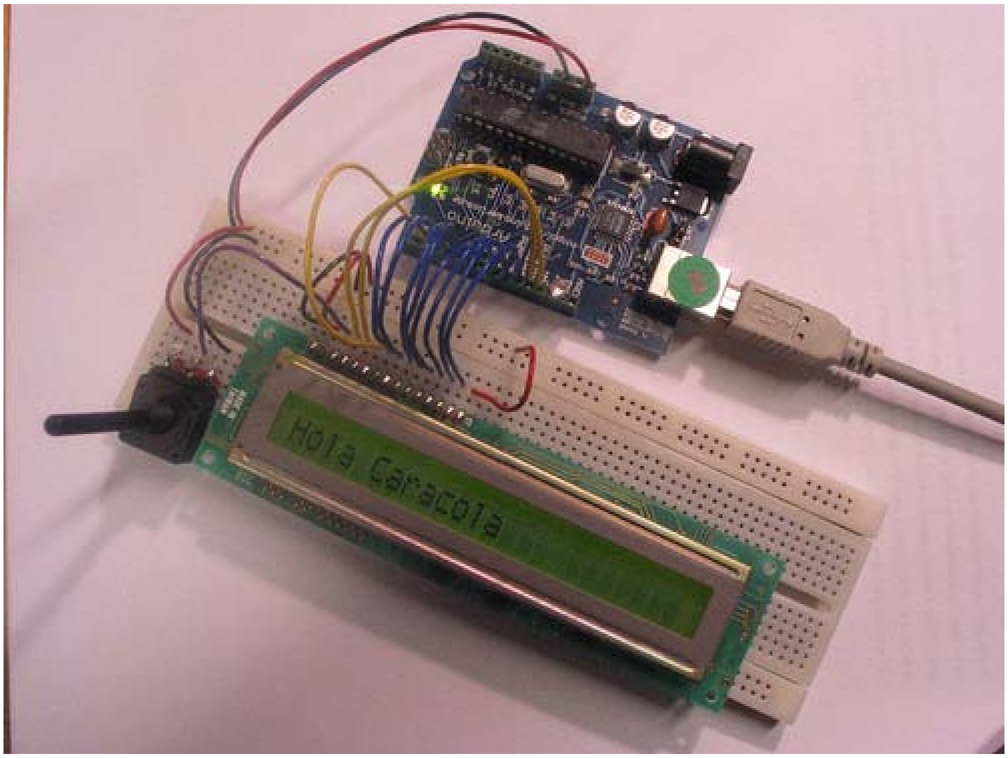
```
/* Shift Out Data
 * --------------
 *
 * Shows a byte, stored in "dato" on a set of 8 LEDs
 *
 * (copyleft) 2005 K3, Malmo University
 * @author: David Cuartielles, Marcus Hannerstig
 * @hardware: David Cuartielles, Marcos Yarza
 * @project: made for SMEE - Experiential Vehicles
 */


int data = 9;
int strob = 8;
int clock = 10;
int oe = 11;
int count = 0;
int dato = 0;

void setup()
{
  beginSerial(9600);
  pinMode(data, OUTPUT);
  pinMode(clock, OUTPUT);
  pinMode(strob, OUTPUT);
  pinMode(oe, OUTPUT);
}

void PulseClock(void) {
    digitalWrite(clock, LOW);
    delayMicroseconds(20);
    digitalWrite(clock, HIGH);
    delayMicroseconds(50);
    digitalWrite(clock, LOW);
}

void loop()
{
   dato = 5;
   for (count = 0; count < 8; count++) {
    digitalWrite(data, dato & 01);
    //serialWrite((dato & 01) + 48);
    dato>>=1;
    if (count == 7){
    digitalWrite(oe, LOW);
    digitalWrite(strob, HIGH);

    }
    PulseClock();
     digitalWrite(oe, HIGH);
 }

  delayMicroseconds(20);
  digitalWrite(strob, LOW);
  delay(100);


  serialWrite(10);
  serialWrite(13);
 delay(100);                  // waits for a moment
}
```

## LCD Display - 8 bits

This example shows the most basic action to be done with a LCD display: to show a welcome message. In our case we have an LCD display with backlight and contrast control. Therefore we will use a potentiometer to regulate the contrast.

LCD displays are most of the times driven using an industrial standard established by Hitachi. According to it there is a group of pins dedicated to sending data and locations of that data on the screen, the user can choose to use 4 or 8 pins to send data. On top of that three more pins are needed to synchronize the communication towards the display.

The backdrop of this example is that we are using almost all the available pins on Arduino board in order to drive the display, but we have decided to show it this way for simplicity.



*Picture of a protoboard supporting the display and a potentiometer*

```
/* LCD Hola
 * --------
 *
 * This is the first example in how to use an LCD screen
 * configured with data transfers over 8 bits. The example
 * uses all the digital pins on the Arduino board, but can
 * easily display data on the display
 *
 * There are the following pins to be considered:
 *
 * - DI, RW, DB0..DB7, Enable (11 in total)
 *
 * the pinout for LCD displays is standard and there is plenty
 * of documentation to be found on the internet.
 *
 * (cleft) 2005 DojoDave for K3
 *
 */

int DI = 12;
int RW = 11;
int DB[] = {3, 4, 5, 6, 7, 8, 9, 10};
int Enable = 2;

void LcdCommandWrite(int value) {
 // poll all the pins
 int i = 0;
 for (i=DB[0]; i <= DI; i++) {
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);
 // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
}

void LcdDataWrite(int value) {
 // poll all the pins
 int i = 0;
 digitalWrite(DI, HIGH);
 digitalWrite(RW, LOW);
 for (i=DB[0]; i <= DB[7]; i++) {
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);
 // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
}

void setup (void) {
 int i = 0;
 for (i=Enable; i <= DI; i++) {
   pinMode(i,OUTPUT);
 }
 delay(100);
 // initiatize lcd after a short pause
 // needed by the LCDs controller
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(64);
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(50);
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(20);
 LcdCommandWrite(0x06);  // entry mode set:
                         // increment automatically, no display shift
 delay(20);
 LcdCommandWrite(0x0E);  // display control:
                         // turn display on, cursor on, no blinking
 delay(20);
 LcdCommandWrite(0x01);  // clear display, set cursor position to zero
 delay(100);
 LcdCommandWrite(0x80);  // display control:
                         // turn display on, cursor on, no blinking
 delay(20);
}

void loop (void) {
  LcdCommandWrite(0x02);  // set cursor position to zero
  delay(10);
  // Write the welcome message
  LcdDataWrite('H');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  LcdDataWrite(' ');
  LcdDataWrite('C');
  LcdDataWrite('a');
  LcdDataWrite('r');
  LcdDataWrite('a');
  LcdDataWrite('c');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  delay(500);
}
```

# Arduino Liquid Crystal Library LCD Interface

In this tutorial you will control a Liquid Crystal Display (LCD) using the Arduino LiquidCrystal library. The library provides functions for accessing any LCD using the common HD44780 parallel interface chipset, such as those available from Sparkfun. It currently implements 8-bit control and one line display of 5x7 characters. Functions are provided to initialize the screen, to print characters and strings, to clear the screen, and to send commands directly to the HD44780 chip. This tutorial will walk you through the steps of wiring an LCD to an Arduino microcontroller board and implementing each of these functions.

Materials needed:

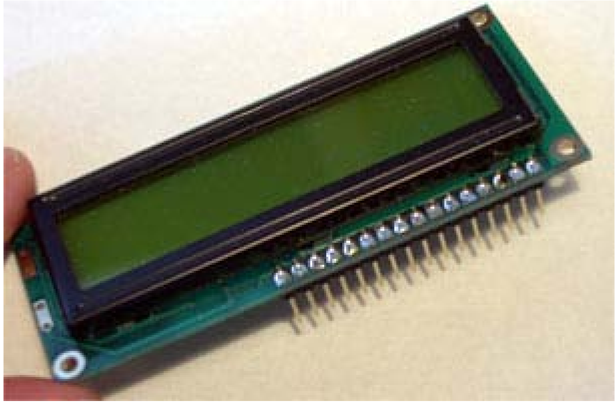
- Solderless breadboard
- Hookup wire
- Arduino Microcontoller Module
- Potentiometer
- Liquid Crystal Display (LCD) with HD44780 chip interface
- Light emitting Diode (LED) - optional, for debugging

### Install the Library

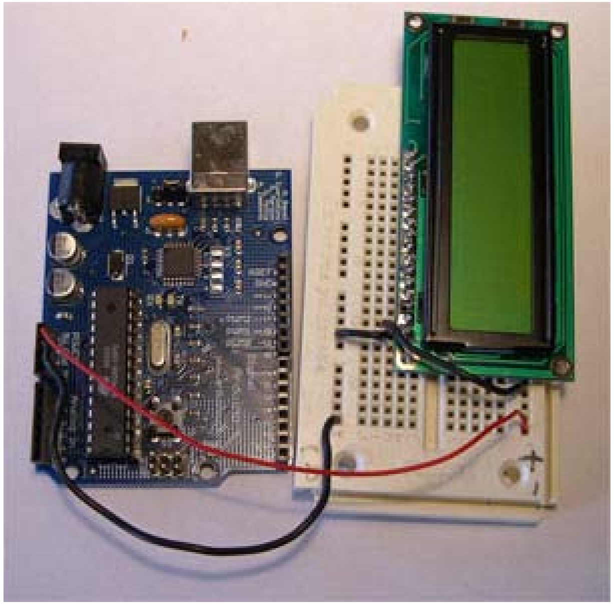
For a basic explanation of how libraries work in Arduino read the library page. Download the LiquidCrystal library here. Unzip the files and place the whole LiquidCrystal folder inside your arduino-0004\lib\targets\libraries folder. Start the Arduino program and check to make sure LiquidCrystal is now available as an option in the Sketch menu under "Import Library".

### Prepare the breadboard

Solder a header to the LCD board if one is not present already.



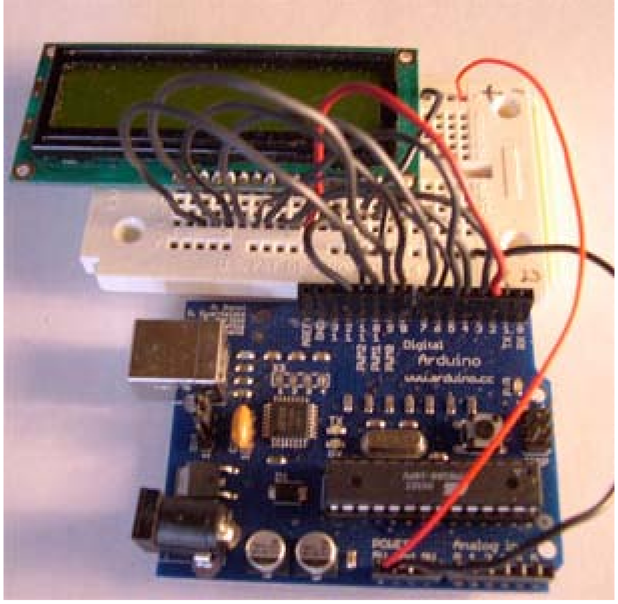
Insert the LCD header into the breadboard and connect power and ground on the breadboard to power and ground from the microcontroller. On the Arduino module, use the 5V and any of the ground connections.



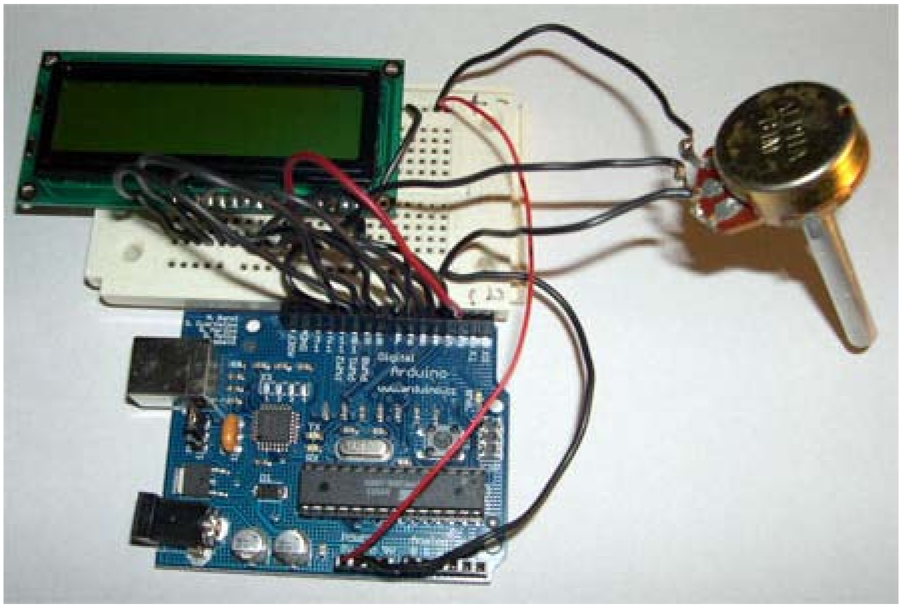
Connect wires from the breadboard to the arduino input sockets. It is a lot of wires, so keep them as short and tidy as possible. Look at the datasheet for your LCD board to figure out which pins are where. Make sure to take note of whether the pin view is from the front or back side of the LCD board, you don't want to get your pins reversed!

The pinout is as follows:

| Arduino | LCD |
|---|---|
| 2 | Enable |
| 3 | Data Bit 0 (DB0) |
| 4 | (DB1) |
| 5 | (DB2) |
| 6 | (DB3) |
| 7 | (DB4) |
| 8 | (DB5) |
| 9 | (DB6) |
| 10 | (DB7) |
| 11 | Read/Write (RW) |
| 12 | Register Select (RS) |



Connect a potentiometer a a voltage divider between 5V, Ground, and the contrast adjustment pin on your LCD.



Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground.

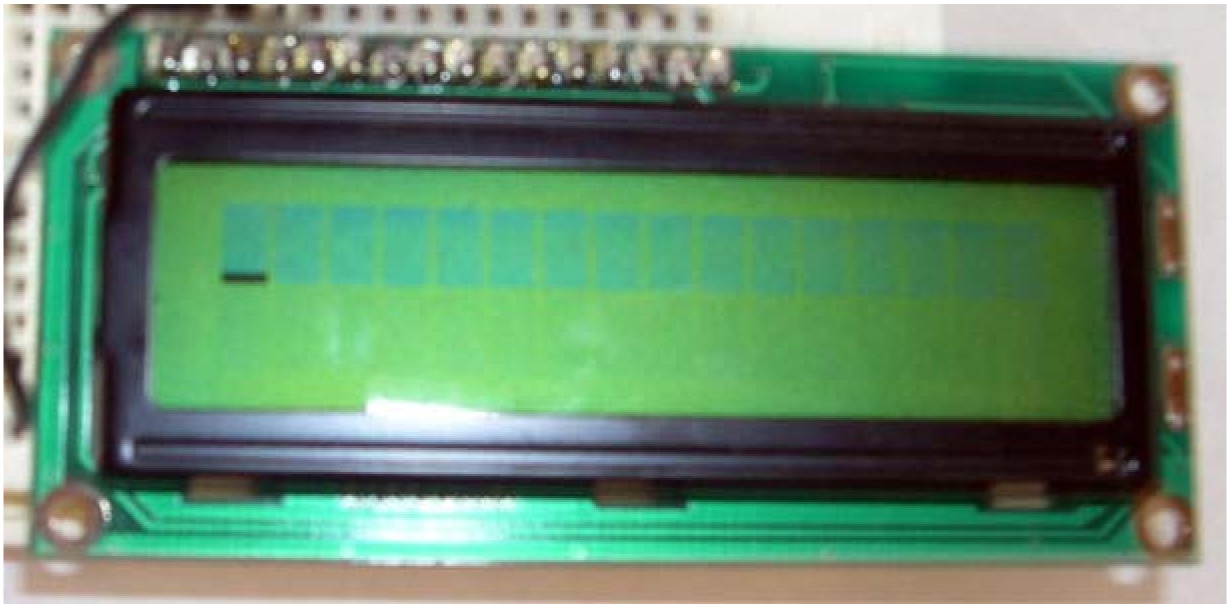
### Program the Arduino

First start by opening a new sketch in Arduino and saving it. Now go to the Sketch menu, scroll down to "import library", and choose "LiquidCrystal". The phrase #include <LiquidCrystal.h> should pop up at the top of your sketch.

The first program we are going to try is simply for calibration and debugging. Copy the following code into your sketch, compile and upload to the Arduino.

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

void setup(void){
  lcd.init(); //initialize the LCD
  digitalWrite(13,HIGH); //turn on an LED for debugging
}

void loop(void){
  delay(1000); //repeat forever
}
```

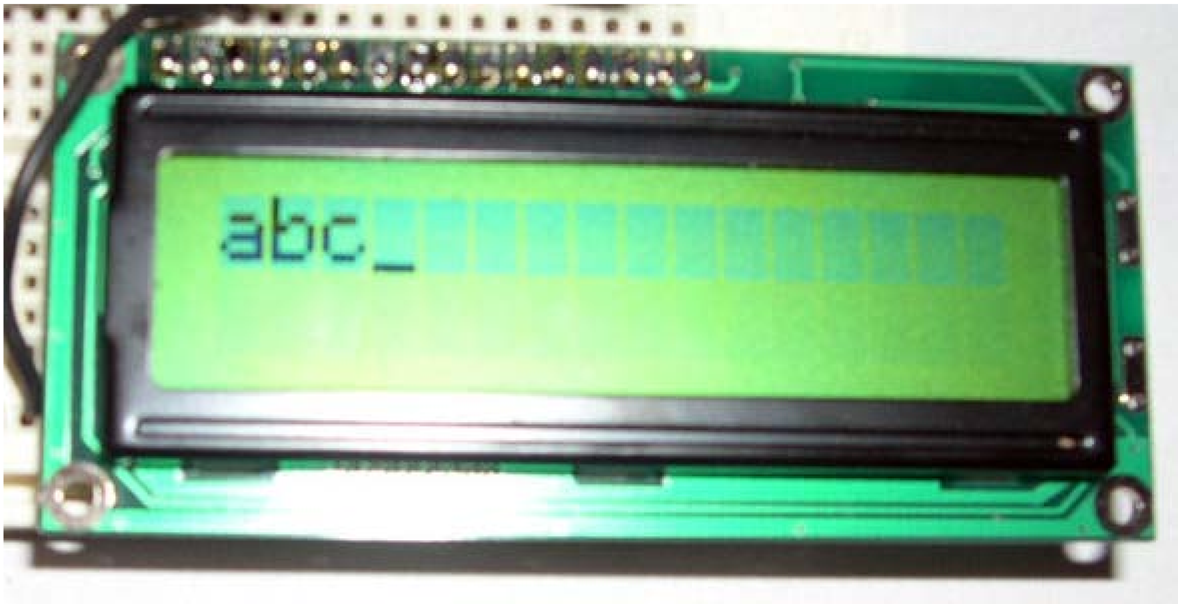
If all went as planned both the LCD and the LED should turn on. Now you can use the potentiometer to adjust the contrast on the LCD until you can clearly see a cursor at the beginning of the first line. If you turn the potentiometer too far in one direction black blocks will appear. Too far in the other direction everything will fade from the display. There should be a small spot in the middle where the cursor appears crisp and dark.



Now let's try something a little more interesting. Compile and upload the following code to the Arduino.

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

void setup(void){
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
}

void loop(void){
   lcd.clear(); //clear the display
   delay(1000); //delay 1000 ms to view change
   lcd.print('a'); //send individual letters to the LCD
   lcd.print('b');
   lcd.print('c');
   delay(1000);//delay 1000 ms to view change

} //repeat forever
```

This time you should see the letters a b and c appear and clear from the display in an endless loop.



This is all great fun, but who really wants to type out each letter of a message indivually? Enter the printIn() function. Simply initialize a string, pass it to printIn(), and now we have ourselves a proper hello world program.

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD
char string1[] = "Hello!"; //variable to store the string "Hello!"

void setup(void){
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
}
void loop(void){
   lcd.clear(); //clear the display
   delay(1000); //delay 1000 ms to view change
   lcd.printIn(string1); //send the string to the LCD
   delay(1000); //delay 1000 ms to view change
} //repeat forever
```

Finally, you should know there is a lot of functionality in the HD44780 chip interface that is not drawn out into Arduino functions. If you are feeling ambitious glance over the datasheet and try out some of the direct commands using the commandWrite() function. For example, commandWrite(2) tells the board to move the cursor back to starting position. Here is an example:

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD
char string1[] = "Hello!"; //variable to store the string "Hello!"

void setup(void){
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
}
void loop(void){
   lcd.commandWrite(2); //bring the cursor to the starting position
   delay(1000); //delay 1000 ms to view change
   lcd.printIn(string1); //send the string to the LCD
   delay(1000); //delay 1000 ms to view change
} //repeat forever
```

This code makes the cursor jump back and forth between the end of the message an the home position.

To interface an LCD directly in Arduino code see this example.

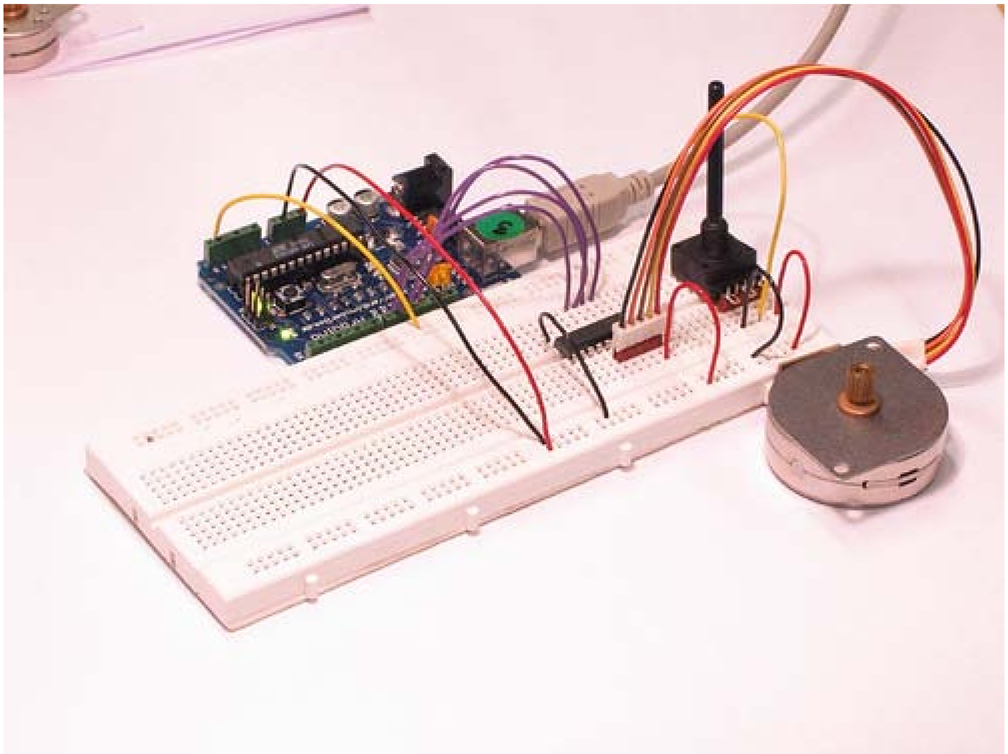
*LCD interface library and tutorial by Heather Dewey-Hagborg*

## Unipolar Stepper Motor

This page shows two examples on how to drive a unipolar stepper motor. These motors can be found in old floppy drives and are easy to control. The one we use has 6 connectors of which one is power (VCC) and the other four are used to drive the motor sending synchronous signals.

The first example is the basic code to make the motor spin in one direction. It is aiming those that have no knowledge in how to control stepper motors. The second example is coded in a more complex way, but allows to make the motor spin at different speeds, in both directions, and controlling both from a potentiometer.

The prototyping board has been populated with a 10K potentiomenter that we connect to an analog input, and a ULN2003A driver. This chip has a bunch of transistors embedded in a single housing. It allows the connection of devices and components that need much higher current than the ones that the ATMEGA8 from our Arduino board can offer.



*Picture of a protoboard supporting the ULN2003A and a potentiometer*

### Example 1: Simple example

```
/* Stepper Copal
 * -------------
 *
 * Program to drive a stepper motor coming from a 5'25 disk drive
 * according to the documentation I found, this stepper: "[...] motor
 * made by Copal Electronics, with 1.8 degrees per step and 96 ohms
 * per winding, with center taps brought out to separate leads [...]"
 * [http://www.cs.uiowa.edu/~jones/step/example.html]
 *
 * It is a unipolar stepper motor with 5 wires:
 *
 * - red: power connector, I have it at 5V and works fine
 * - orange and black: coil 1
 * - brown and yellow: coil 2
 *
 * (cleft) 2005 DojoDave for K3
 * http://www.0j0.org | http://arduino.berlios.de
 *
 * @author: David Cuartielles
 * @date: 20 Oct. 2005
 */

int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;
int delayTime = 500;

void setup() {
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
}

void loop() {
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  delay(delayTime);
}
```

### Example 2: Stepper Unipolar Advanced

```
/* Stepper Unipolar Advanced
 * -------------------------
 *
 * Program to drive a stepper motor coming from a 5'25 disk drive
 * according to the documentation I found, this stepper: "[...] motor
 * made by Copal Electronics, with 1.8 degrees per step and 96 ohms
 * per winding, with center taps brought out to separate leads [...]"
 * [http://www.cs.uiowa.edu/~jones/step/example.html]
 *
 * It is a unipolar stepper motor with 5 wires:
 *
 * - red: power connector, I have it at 5V and works fine
 * - orange and black: coil 1
 * - brown and yellow: coil 2
 *
 * (cleft) 2005 DojoDave for K3
 * http://www.0j0.org | http://arduino.berlios.de
 *
 * @author: David Cuartielles
 * @date: 20 Oct. 2005
 */

int motorPins[] = {8, 9, 10, 11};
int count = 0;
int count2 = 0;
int delayTime = 500;
int val = 0;

void setup() {
  for (count = 0; count < 4; count++) {
    pinMode(motorPins[count], OUTPUT);
  }
}

void moveForward() {
  if ((count2 == 0) || (count2 == 1)) {
    count2 = 16;
  }
  count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[count], count2>>count&0x01);
  }
  delay(delayTime);
}

void moveBackward() {
  if ((count2 == 0) || (count2 == 1)) {
    count2 = 16;
  }
  count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[3 - count], count2>>count&0x01);
  }
  delay(delayTime);
}

void loop() {
  val = analogRead(0);
  if (val > 540) {
    // move faster the higher the value from the potentiometer
    delayTime = 2048 - 1024 * val / 512 + 1;
    moveForward();
  } else if (val < 480) {
    // move faster the lower the value from the potentiometer
    delayTime = 1024 * val / 512 + 1;
    moveBackward();
  } else {
    delayTime = 1024;
  }
}
```

### References

In order to work out this example, we have been looking into quite a lot of documentation. The following links may be useful for you to visit in order to understand the theory underlying behind stepper motors:

- information about the motor we are using - here

- basic explanation about steppers - here

- good PDF with basic information - here

## DMX Master Device

Please see this updated tutorial on the playground.

# Arduino Software Serial Interface

**Note:** *If you just want to use a software serial interface, see the SoftwareSerial library included with Arduino 0007 and later. Read on if you'd like to know how that library works.*

In this tutorial you will learn how to implement Asynchronous serial communication on the Arduino in software to communicate with other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page. For a good explanation of serial communication see Wikipedia. The software serial connection can run at 4800 baud or 9600 baud reliably.

Functions Available:

**SWread();** Returns a byte long integer value from the software serial connection

Example:

```
byte RXval;
RXval = SWread();
```

**SWprint();** Sends a byte long integer value out the software serial connection

Example:

```
byte TXval = 'h';
byte TXval2 = 126;
SWprint(TXval);
SWprint(TXval2);
```

Definitions Needed:

```
#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94
```

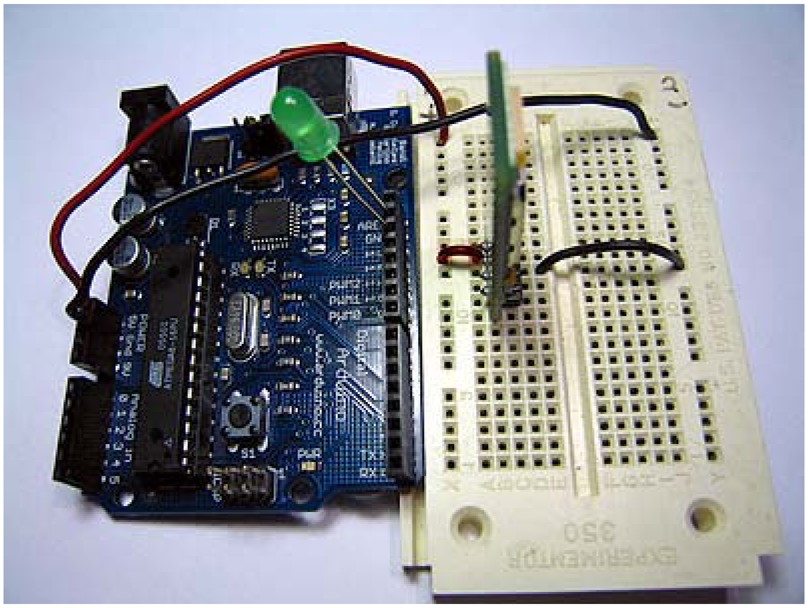
These definitions set the delays necessary for 9600 baud and 4800 baud software serial operation.

Materials needed:

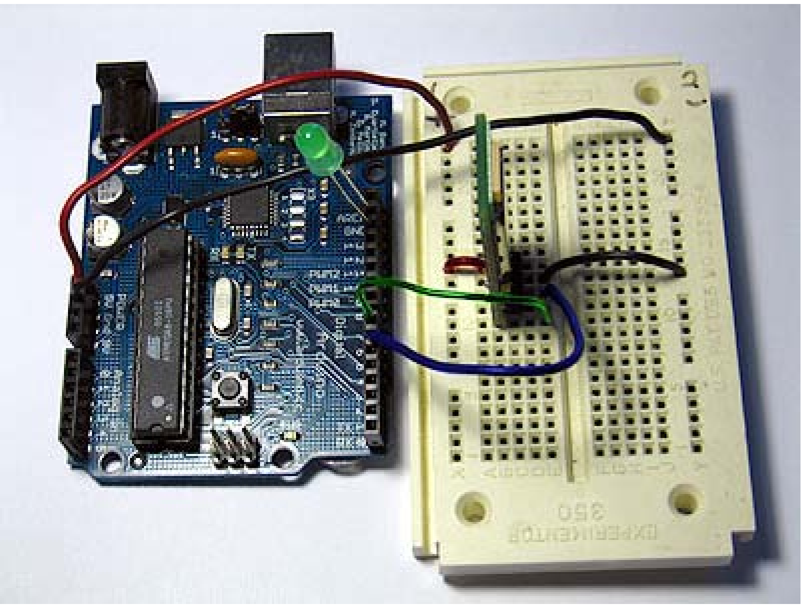
- Device to communicate with
- Solderless breadboard
- Hookup wire
- Arduino Microcontroller Module
- Light emitting Diode (LED) - optional, for debugging

### Prepare the breadboard

Insert the device you want to communicate with in the breadboard. Connect ground on the breadboard to ground from the microcontroller. If your device uses 5v power connect 5v from the microcontoller to 5v on the breadboard. Otherwise connect power and ground from an alternate power source to the breadboard in the same fashion. Make any other connections necessary for your device. Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground.



Decide which pins you want to use for transmitting and receiving. In this example we will use pin 7 for transmitting and pin 6 for receiving, but any of the digital pins should work.



### Program the Arduino

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. We will walk through the code in small sections.

```
#include <ctype.h>

#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94
```

Here we set up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

First we include the file ctype.h in our application. This gives us access to the `toupper()` function from the Character Operations C library which we will use later in our main loop. This library is part of the Arduino install, so you don't need to do anything other than type the `#include` line in order to use it. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates. The `#define bit9600Delay 84` line causes the compiler to substitute the number 84 where ever it encounters the label "bit9600Delay". Pre-processor definitions are often used for constants because they don't take up any program memory space on the chip.

```
byte rx = 6;
byte tx = 7;
byte SWval;
```

Here we set our transmit (tx) and recieve (rx) pins. Change the pin numbers to suit your application. We also allocate a variable to store our recieved data in, `SWval`.

```
void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  digitalWrite(13,HIGH); //turn on debugging LED
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}
```

Here we initialize the lines, turn on our debugging LED and print a debugging message to confirm all is working as planned. We can pass inidvidual characters or numbers to the SWprint function.

```
void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}
```

This is the SWprint function. First the transmit line is pulled low to signal a start bit. Then we itterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted. Finally we pull the line high again to signal a stop bit. For each bit we transmit we hold the line high or low for the specified delay. In this example we are using a 9600 baudrate. To use 4800 simply replace the variable `bit9600Delay` with `bit4800Delay`.

```
int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfBit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}
```

This is the SWread function. This will wait for a byte to arrive on the recieve pin and then return it to the allocated variable.

First we wait for the recieve line to be pulled low. We check after a half bit delay to make sure the line is still low and we didn't just recieve line noise. Then we iterate through a bit mask and shift 1s or 0s into our output byte based on what we recieve. Finally we allow a pause for the stop bit and then return the value.

```
void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```

Finally we implement our main program loop. In this program we simply wait for characters to arrive, change them to uppercase and send them back. This is always a good program to run when you want to make sure a serial connection is working properly.

For lots of fun serial devices check out the Sparkfun online catalog. They have lots of easy to use serial modules for GPS, bluetooth, wi-fi, LCDs, etc.

For easy copy and pasting the full program text of this tutorial is below:

```
//Created August 15 2006
//Heather Dewey-Hagborg
//http://www.arduino.cc

#include <ctype.h>

#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94

byte rx = 6;
byte tx = 7;
byte SWval;

void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  digitalWrite(13,HIGH); //turn on debugging LED
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}

void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}

int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfBit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}

void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```

*code and tutorial by Heather Dewey-Hagborg*

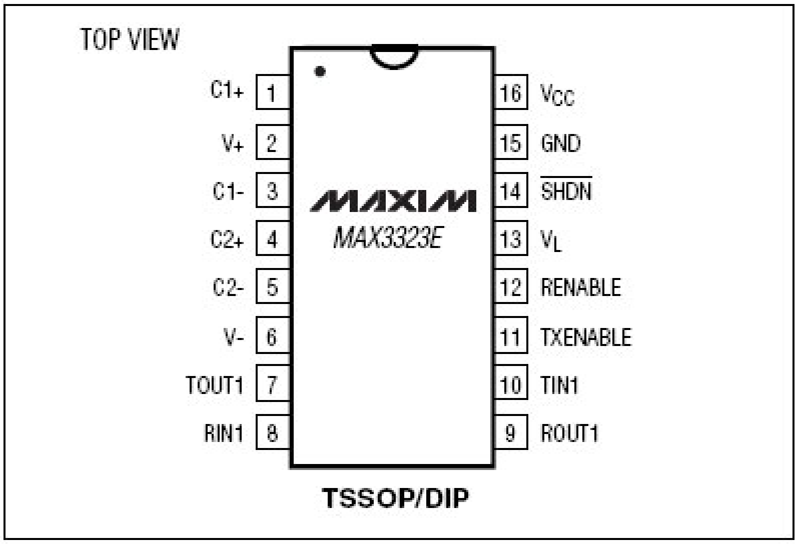
# RS-232

In this tutorial you will learn how to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino. A general purpose software serial tutorial can be found here.

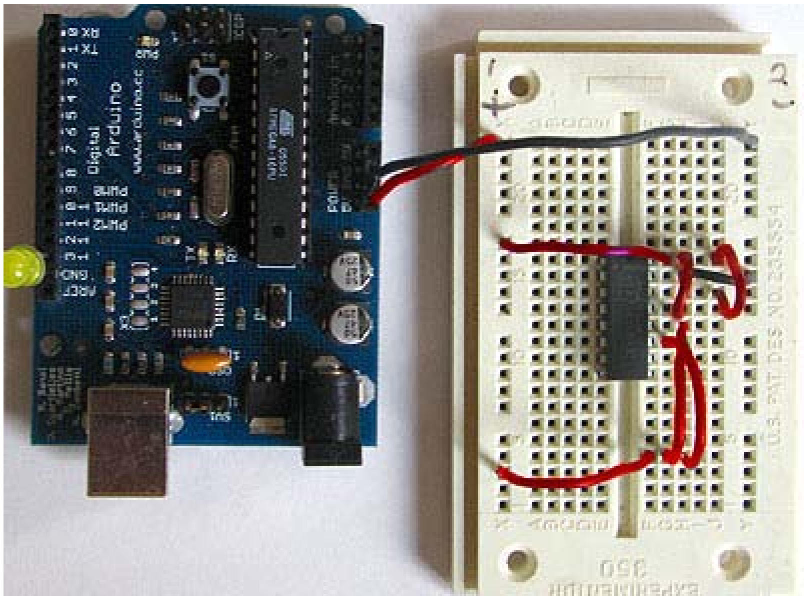
Materials needed:

- Computer with a terminal program installed (ie. HyperTerminal or RealTerm on the PC, Zterm on Mac)
- Serial-Breadboard cable
- MAX3323 chip (or similar)
- 4 1uf capacitors
- Solderless breadboard
- Hookup wire
- Arduino Microcontroller Module
- Light emitting Diode (LED) - optional, for debugging

## Prepare the breadboard

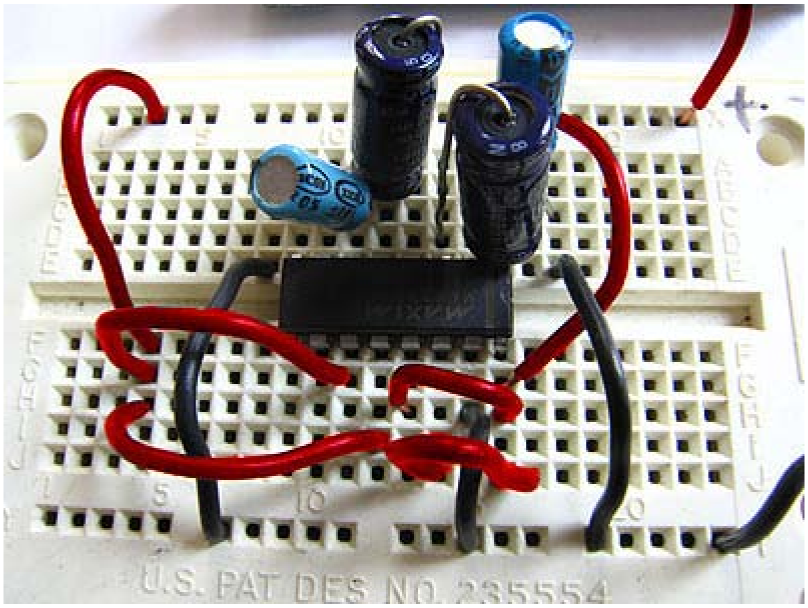


Insert the MAX3323 chip in the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect pin 15 on the MAX233 chip to ground and pins 16 and 14 - 11 to 5V. If you are using an LED connect it between pin 13 and ground.



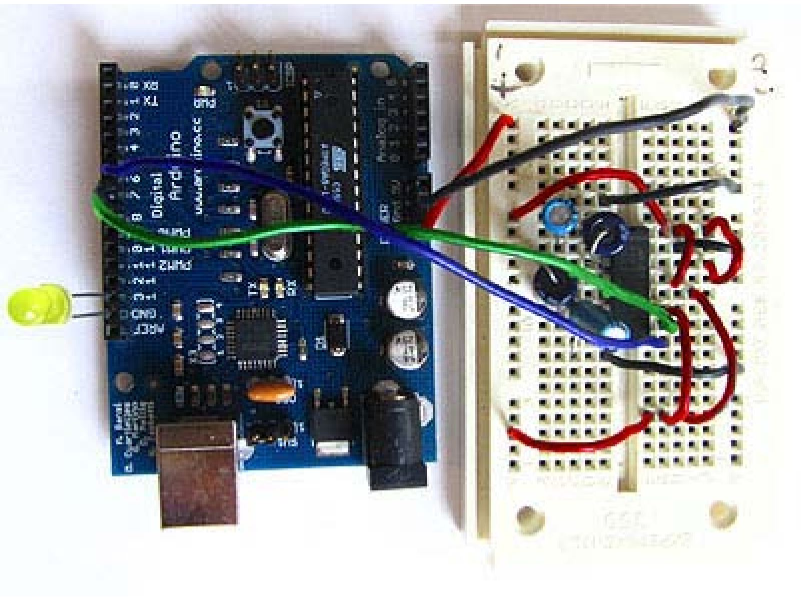
*+5v wires are red, GND wires are black*

Connect a 1uF capacitor across pins 1 and 3, another across pins 4 and 5, another between pin 1 and ground, and the last between pin 6 and ground. If you are using polarized capacitors make sure the negative pins connect to the negative sides (pins 3 and 5 and ground).



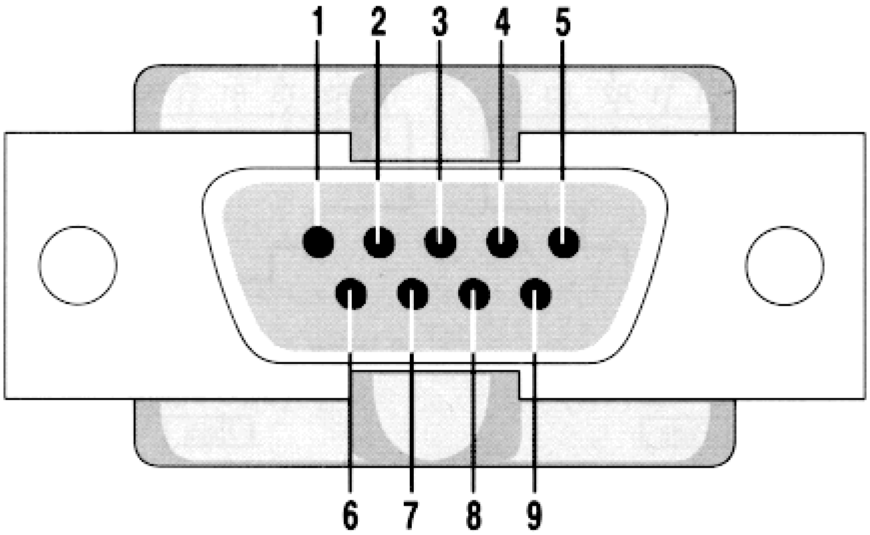
*+5v wires are red, GND wires are black*

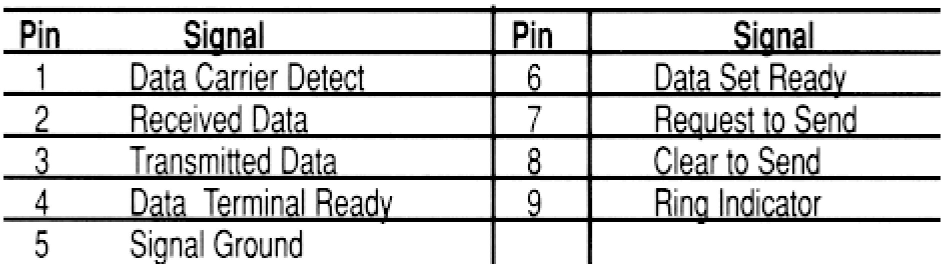
Determine which Arduino pins you want to use for your transmit (TX) and recieve (RX) lines. In this tutorial we will be using Arduino pin 6 for receiving and pin 7 for transmitting. Connect your TX pin (7) to MAX3323 pin 10 (T1IN). Connect your RX pin (6) to MAX3323 pin 9 (R1OUT).



*TX wire Green, RX wire Blue, +5v wires are red, GND wires are black*

## Cables

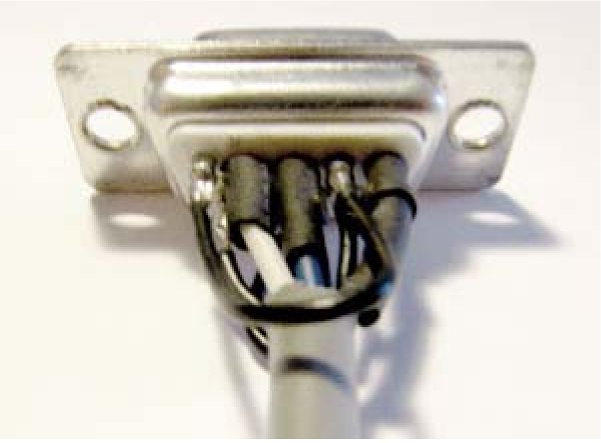




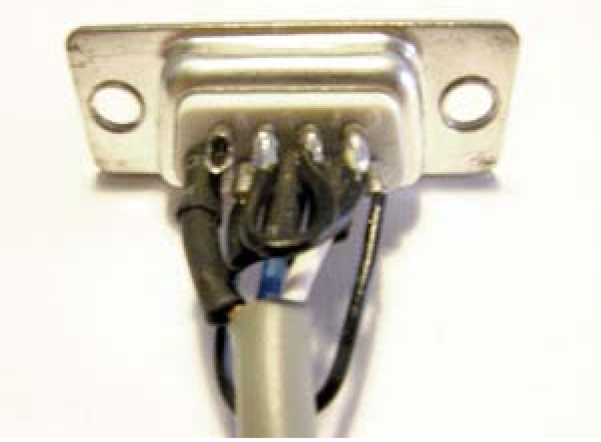
| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data  Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

*(DB9 Serial Connector Pin Diagram)*

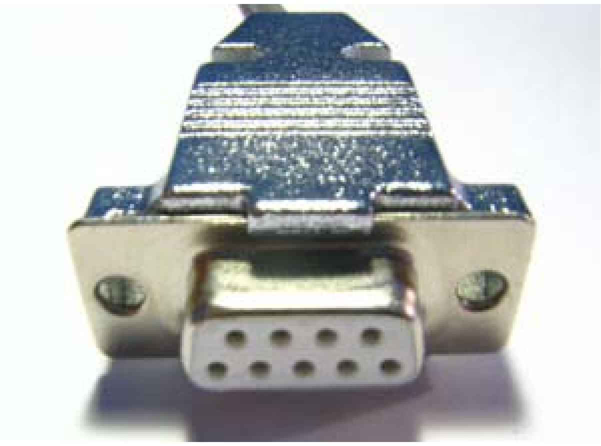
If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. To do this, pick up a female DB9 connector from radioshack. Pick three different colors of wire, one for TX, one for RX, and one for ground. Solder your TX wire to pin 2 of the DB9 connector, RX wire to pin 3 and Ground to pin 5.



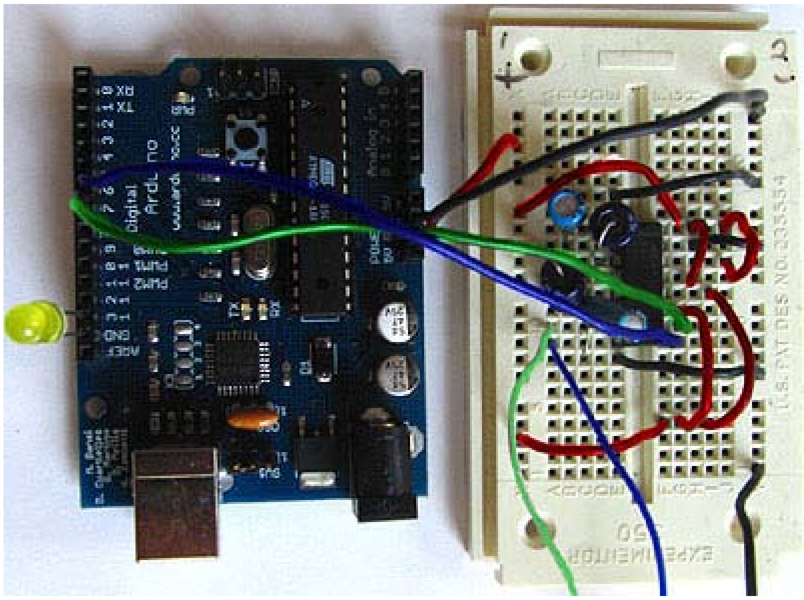
Connect pins 1 and 6 to pin 4 and pin 7 to pin 8. Heatshrink the wire connections to avoid accidental shorts.



Enclose the connector in a backshell to further protect the signal and enable easy unplugging from your serial port.



Connect the TX line from your computer to pin 8 (R1IN) on the MAX233 and the RX line to pin 7 (T1OUT). Connect the ground line from your computer to ground on the breadboard.



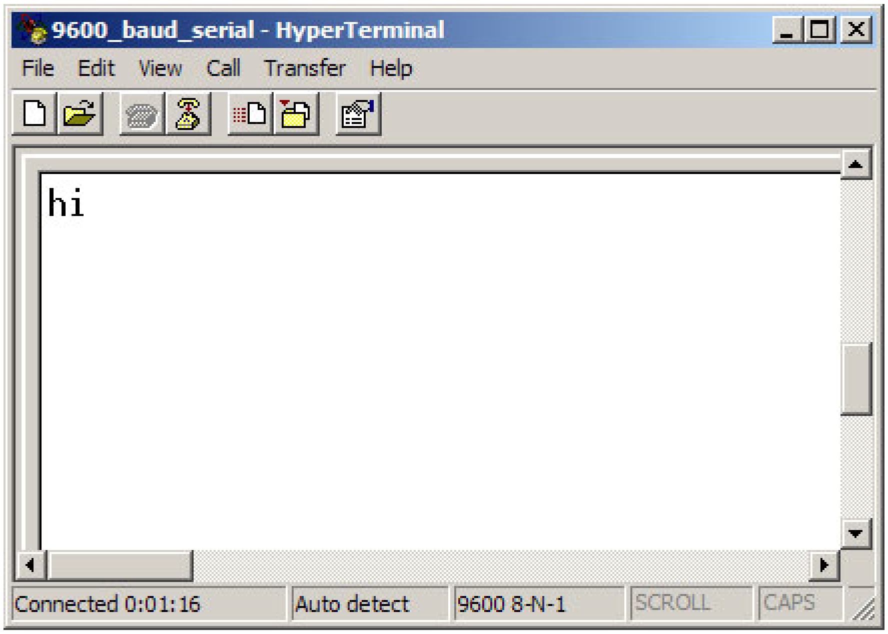
*TX wires Green, RX wires Blue, +5v wires are red, GND wires are black*

## Program the Arduino

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. Upload the following code into the Arduino microcontroller module:

```
//Created August 23 2006
//Heather Dewey-Hagborg
//http://www.arduino.cc

#include <ctype.h>

#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94

byte rx = 6;
byte tx = 7;
byte SWval;

void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  digitalWrite(13,HIGH); //turn on debugging LED
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}

void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}

int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfBit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}

void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```

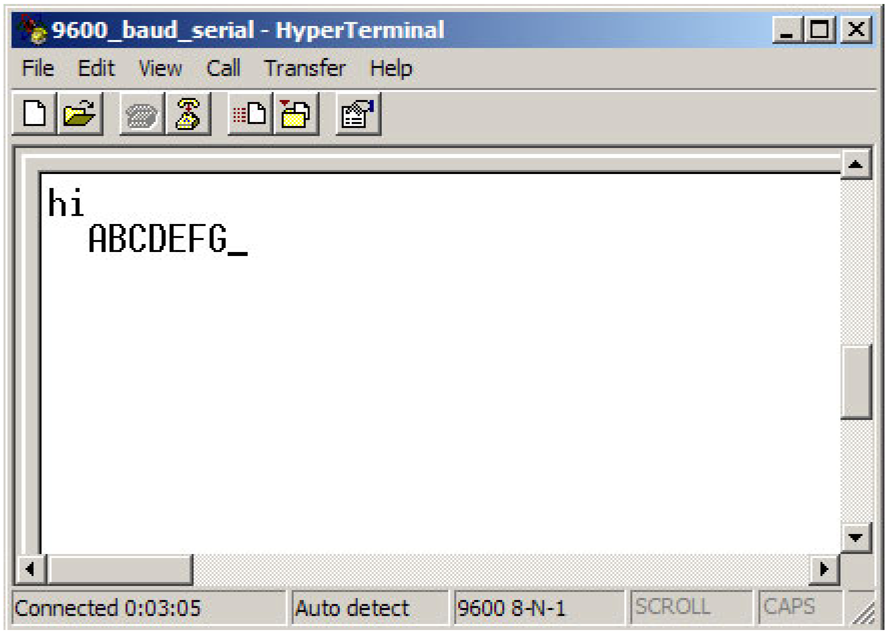
Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by an advancement to the next line. Here is a shot of what it should look like in Hyperterminal, the free pre-installed windows terminal application.


9600_baud_serial - HyperTerminal
File Edit View Call Transfer Help

```
hi
```

Connected 0:01:16
Auto detect
9600 8-N-1
SCROLL
CAPS

Now, try typing a lowercase character into the terminal window. You should see the letter you typed return to you in uppercase.



```
hi
  ABCDEFG_
```

Connected 0:03:05 | Auto detect | 9600 8-N-1 | SCROLL | CAPS

If this works, congratulations! Your serial connection is working as planned. You can now use your new serial/computer connection to print debugging statements from your code, and to send commands to your microcontroller.

*code and tutorial by Heather Dewey-Hagborg, photos by Thomas Dexter*

# Interfacing a Serial EEPROM Using SPI

In this tutorial you will learn how to interface with an AT25HP512 Atmel serial EEPROM using the Serial Peripheral Interface (SPI) protocol. EEPROM chips such as this are very useful for data storage, and the steps we will cover for implementing SPI communication can be modified for use with most other SPI devices. Note that the chip on the Arduino board contains an internal EEPROM, so follow this tutorial only if you need more space than it provides.

Materials Needed:

- AT25HP512 Serial EEPROM chip (or similar)
- Hookup wire
- Arduino Microcontroller Module

## Introduction to the Serial Peripheral Interface

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by Microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices,

- Master In Slave Out (MISO) - The Slave line for sending data to the master,
- Master Out Slave In (MOSI) - The Master line for sending data to the peripherals,
- Serial Clock (SCK) - The clock pulses which synchronize data transmission generated by the master, and
- Slave Select pin - allocated on each device which the master can use to enable and disable specific devices and avoid false transmissions due to line noise.

The difficult part about SPI is that the standard is loose and each device implements it a little differently. This means you have to pay special attention to the datasheet when writing your interface code. Generally speaking there are three modes of transmission numbered 0 - 3. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal, and whether the clock is idle when high or low.

All SPI settings are determined by the Arduino SPI Control Register (SPCR). A register is just a byte of microcontroller memory that can be read from or written to. Registers generally serve three purposes, control, data and status.

Control registers code control settings for various microcontroller functionalities. Usually each bit in a control register effects a particular setting, such as speed or polarity.

Data registers simply hold bytes. For example, the SPI data register (SPDR) holds the byte which is about to be shifted out the MOSI line, and the data which has just been shifted in the MISO line.

Status registers change their state based on various microcontroller conditions. For example, the seventh bit of the SPI status register (SPSR) gets set to 1 when a value is shifted in or out of the SPI.

The SPI control register (SPCR) has 8 bits, each of which control a particular SPI setting.

```
SPCR
| 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| SPIE | SPE  | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 |

SPIE - Enables the SPI interrupt when 1
SPE - Enables the SPI when 1
DORD - Sends data least Significant Bit First when 1, most Significant Bit first when 0
MSTR - Sets the Arduino in master mode when 1, slave mode when 0
CPOL - Sets the data clock to be idle when high if set to 1, idle when low if set to 0
CPHA - Samples data on the falling edge of the data clock when 1, rising edge when 0
```

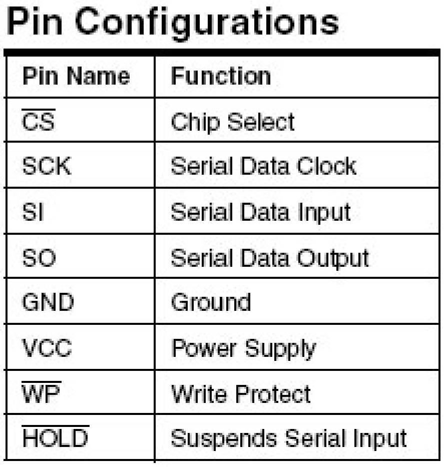
```
SPR1 and SPR0 - Sets the SPI speed, 00 is fastest (4MHz) 11 is slowest (250KHz)
```

This means that to write code for a new SPI device you need to note several things and set the SPCR accordingly:

- Is data shifted in MSB or LSB first?
- Is the data clock idle when high or low?
- Are samples on the rising or falling edge of clock pulses?
- What speed is the SPI running at?

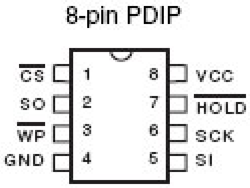
Once you have your SPI Control Register set correctly you just need to figure out how long you need to pause between instructions and you are ready to go. Now that you have a feel for how SPI works, let's take a look at the details of the EEPROM chip.

## Introduction to Serial EEPROM



Pin Configurations

| Pin Name | Function |
|---|---|
| $\overline{CS}$ | Chip Select |
| SCK | Serial Data Clock |
| SI | Serial Data Input |
| SO | Serial Data Output |
| GND | Ground |
| VCC | Power Supply |
| $\overline{WP}$ | Write Protect |
| $\overline{HOLD}$ | Suspends Serial Input |

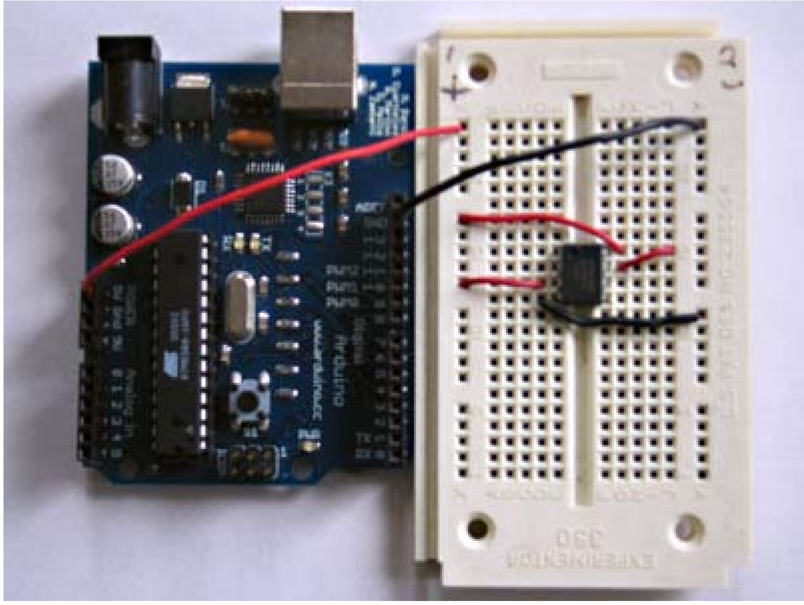


The AT25HP512 is a 65,536 byte serial EEPROM. It supports SPI modes 0 and 3, runs at up to 10MHz at 5v and can run at slower speeds down to 1.8v. It's memory is organized as 512 pages of 128 bytes each. It can only be written 128 bytes at a time, but it can be read 1-128 bytes at a time. The device also offers various degerees of write protection and a hold pin, but we won't be covering those in this tutorial.

The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 8 bit operational codes (opcodes) and are shifted in on the rising edge of the data clock. It takes the EEPROM about 10 milliseconds to write a page (128 bytes) of data, so a 10ms pause should follow each EEPROM write routine.

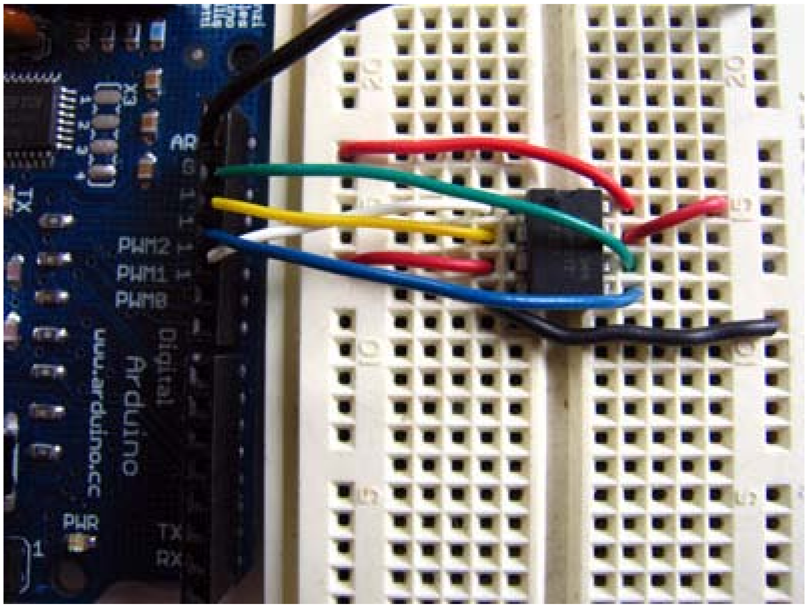
## Prepare the Breadboard

Insert the AT25HP512 chip into the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect EEPROM pins 3, 7 and 8 to 5v and pin 4 to ground.



*+5v wires are red, GND wires are black*

Connect EEPROM pin 1 to Arduino pin 10 (Slave Select - SS), EEPROM pin 2 to Arduino pin 12 (Master In Slave Out - MISO), EEPROM pin 5 to Arduino pin 11 (Master Out Slave In - MOSI), and EEPROM pin 6 to Arduino pin 13 (Serial Clock - SCK).



*SS wire is white, MISO wire is yellow, MOSI wire is blue, SCK wire is green*

## Program the Arduino

Now we will write the code to enable SPI communication between the EEPROM and the Arduino. In the setup routine this program fills 128 bytes, or one page of the EEPROM with data. In the main loop it reads that data back out, one byte at a time and prints that byte out the built in serial port. We will walk through the code in small sections.

The first step is setting up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

We define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Then we define our opcodes for the EEPROM. Opcodes are control commands:

```
#define DATAOUT 11//MOSI
#define DATAIN  12//MISO
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss

//opcodes
```

```
#define WREN  6
#define WRDI  4
#define RDSR  5
#define WRSR  1
#define READ  3
#define WRITE 2
```

Here we allocate the global variables we will be using later in the program. Note char buffer [128];. this is a 128 byte array we will be using to store the data for the EEPROM write:

```
byte eeprom_output_data;
byte eeprom_input_data=0;
byte clr;
int address=0;
//data buffer
char buffer [128];
```

First we initialize our serial connection, set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise:

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

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
```

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variable to clear out any spurious data from past runs:

```

  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
```

Here we fill our data array with numbers and send a write enable instruction to the EEPROM. The EEPROM MUST be write enabled before every write instruction. To send the instruction we pull the SLAVESELECT line low, enabling the device, and then send the instruction using the spi_transfer function. Note that we use the WREN opcode we defined at the beginning of the program. Finally we pull the SLAVESELECT line high again to release it:

```
  //fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
```

Now we pull the SLAVESELECT line low to select the device again after a brief delay. We send a WRITE instruction to tell the EEPROM we will be sending data to record into memory. We send the 16 bit address to begin writing at in two bytes, Most Significant Bit first. Next we send our 128 bytes of data from our buffer array, one byte after another without pause. Finally we set the SLAVESELECT pin high to release the device and pause to allow the EEPROM to write the data:

```
  delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
```

We end the setup function by sending the word "hi" plus a line feed out the built in serial port for debugging purposes. This way if our data comes out looking funny later on we can tell it isn't just the serial port acting up:

```
  Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
}
```

In our main loop we just read one byte at a time from the EEPROM and print it out the serial port. We add a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 because we have only filled 128 addresses in the EEPROM with data:

```
void loop()
{
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  delay(500); //pause for readability
}
```

The fill_buffer function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application:

```
void fill_buffer()
{
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
}
```

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait for the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

The read_eeprom function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data:

```
byte read_eeprom(int EEPROM_address)
{
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
}
```

For easy copy and pasting the full program text of this tutorial is below:

```
#define DATAOUT 11//MOSI
#define DATAIN  12//MISO
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss

//opcodes
#define WREN  6
#define WRDI  4
#define RDSR  5
#define WRSR  1
#define READ  3
#define WRITE 2

byte eeprom_output_data;
byte eeprom_input_data=0;
byte clr;
int address=0;
//data buffer
char buffer [128];

void fill_buffer()
{
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
}

char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}

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

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  //fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
  delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
  Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
}

byte read_eeprom(int EEPROM_address)
{
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
}

void loop()
{
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  if (address == 128)
    address = 0;
  delay(500); //pause for readability
}
```

*code and tutorial by Heather Dewey-Hagborg, photos by Thomas Dexter*

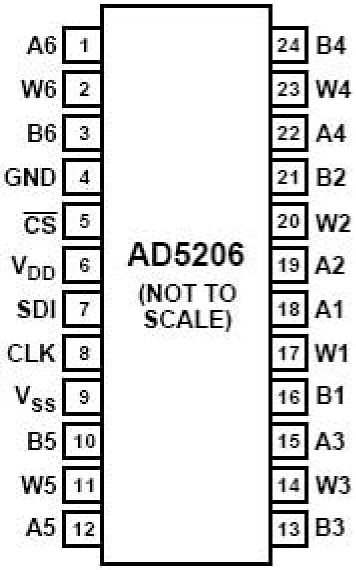
# Controlling a Digital Potentiometer Using SPI

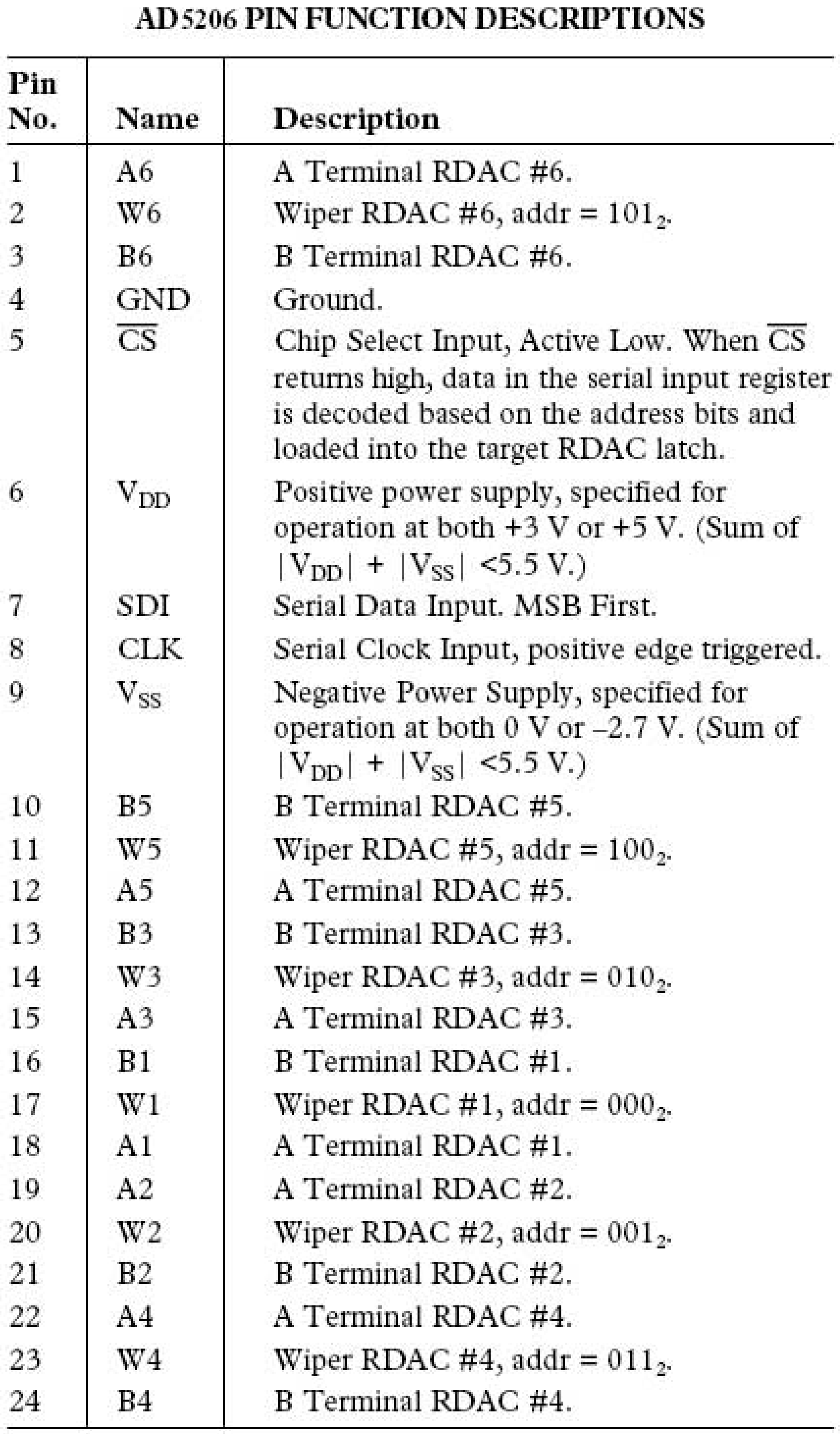
In this tutorial you will learn how to control the AD5206 digital potentiometer using Serial Peripheral Interface (SPI). For an explanation of SPI see the SPI EEPROM tutorial. Digital potentiometers are useful when you need to vary the resistance in a ciruit electronically rather than by hand. Example applications include LED dimming, audio signal conditioning and tone generation. In this example we will use a six channel digital potentiometer to control the brightness of six LEDs. The steps we will cover for implementing SPI communication can be modified for use with most other SPI devices.

Materials Needed:

- AD5206 Digital Potentiometer
- Arduino Microcontroller Module
- 6 Light Emitting Diodes (LEDs)
- Hookup Wire

## Introduction to the AD5206 Digital Potentiometer





## AD5206 PIN FUNCTION DESCRIPTIONS

| Pin No. | Name | Description |
|---|---|---|
| 1 | A6 | A Terminal RDAC #6. |
| 2 | W6 | Wiper RDAC #6, addr = $101_2$. |
| 3 | B6 | B Terminal RDAC #6. |
| 4 | GND | Ground. |
| 5 | $\overline{CS}$ | Chip Select Input, Active Low. When $\overline{CS}$ returns high, data in the serial input register is decoded based on the address bits and loaded into the target RDAC latch. |
| 6 | $V_{DD}$ | Positive power supply, specified for operation at both +3 V or +5 V. (Sum of $\lvert V_{DD}\rvert + \lvert V_{SS}\rvert < 5.5$ V.) |
| 7 | SDI | Serial Data Input. MSB First. |
| 8 | CLK | Serial Clock Input, positive edge triggered. |
| 9 | $V_{SS}$ | Negative Power Supply, specified for operation at both 0 V or –2.7 V. (Sum of $\lvert V_{DD}\rvert + \lvert V_{SS}\rvert < 5.5$ V.) |
| 10 | B5 | B Terminal RDAC #5. |
| 11 | W5 | Wiper RDAC #5, addr = $100_2$. |
| 12 | A5 | A Terminal RDAC #5. |
| 13 | B3 | B Terminal RDAC #3. |
| 14 | W3 | Wiper RDAC #3, addr = $010_2$. |
| 15 | A3 | A Terminal RDAC #3. |
| 16 | B1 | B Terminal RDAC #1. |
| 17 | W1 | Wiper RDAC #1, addr = $000_2$. |
| 18 | A1 | A Terminal RDAC #1. |
| 19 | A2 | A Terminal RDAC #2. |
| 20 | W2 | Wiper RDAC #2, addr = $001_2$. |
| 21 | B2 | B Terminal RDAC #2. |
| 22 | A4 | A Terminal RDAC #4. |
| 23 | W4 | Wiper RDAC #4, addr = $011_2$. |
| 24 | B4 | B Terminal RDAC #4. |

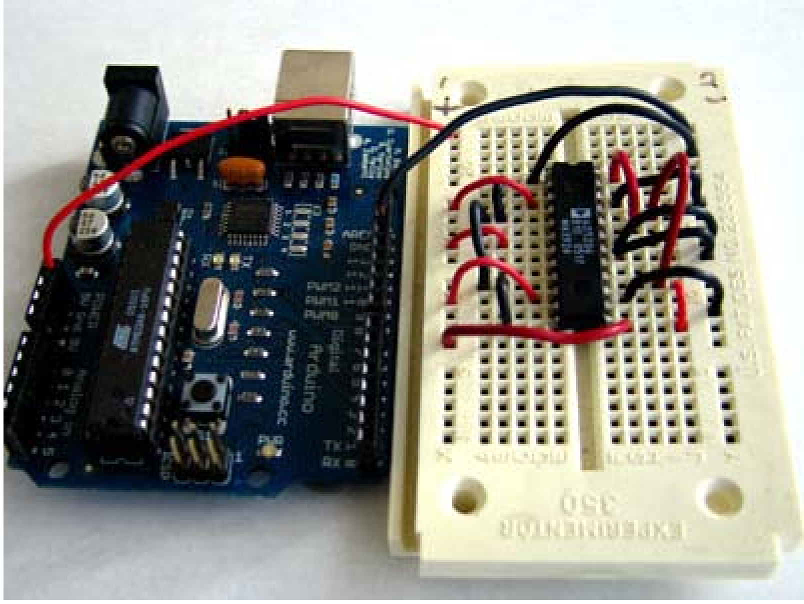
The AD5206 is a 6 channel digital potentiometer. This means it has six variable resistors (potentiometers) built in for individual electronic control. There are three pins on the chip for each of the six internal variable resistors, and they can be interfaced with just as you would use a mechanical potentiometer. The individual variable resistor pins are labeled Ax, Bx and Wx, ie. A1, B1 and W1.

For example, in this tutorial we will be using each variable resistor as a voltage divider by pulling one side pin (pin B) high, pulling another side pin (pin A) low and taking the variable voltage output of the center pin (Wiper).

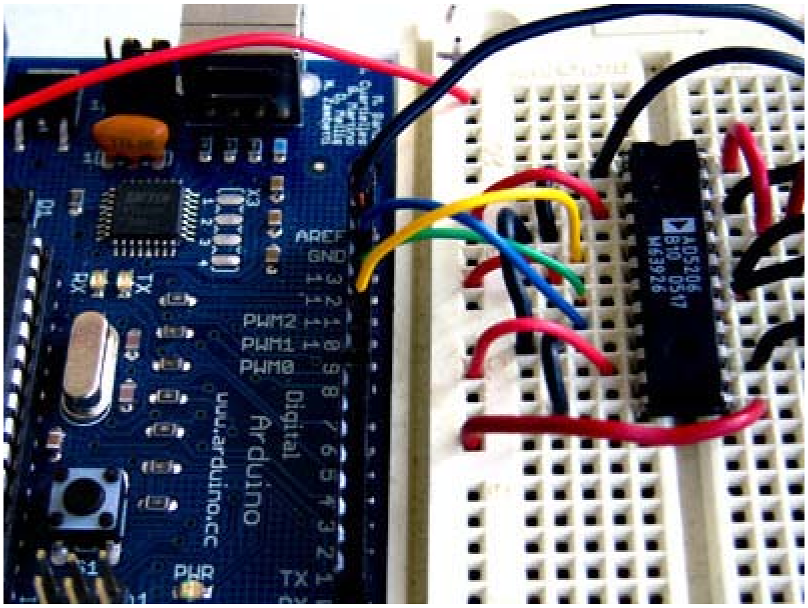
The AD5206 is digitally controlled using SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) with the three most significant bits (11-9) defining the address of which potentiometer to adjust and the eight least significant bits (8-1) defining what value to set that potentiometer to from 0-255. Data is shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs much faster than the Arduino, so we don't need to worry about pre-scaling to slow down the transmission.

## Prepare the Breadboard

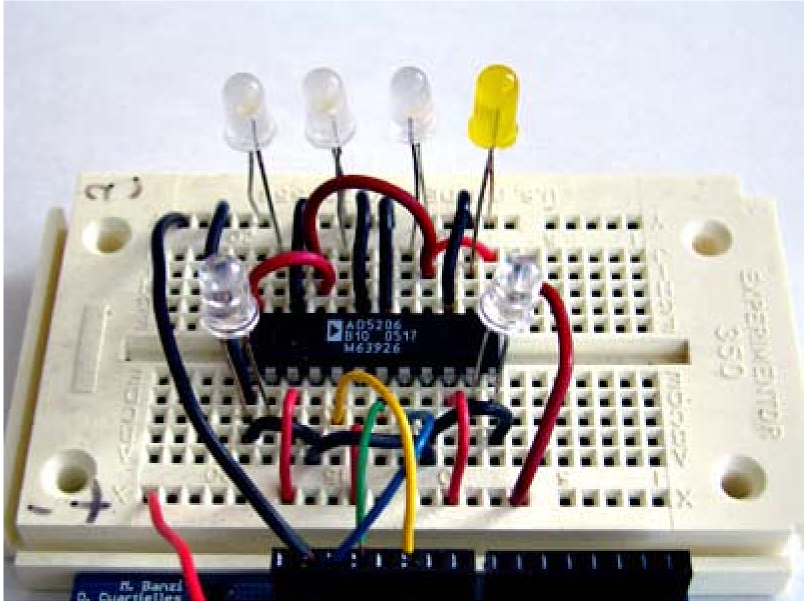
Insert the AD5206 chip into the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect AD5206 pins 3, 6, 10, 13, 16, 21 and 24 to 5v and pins 1, 4, 9, 12, 15, 18, 19, and 22 to ground. We are connecting all the A pins to ground and all of the B pins to 5v to create 6 voltage dividers.



Connect AD5206 pin 5 to Arduino pin 10 (Slave Select - SS), AD5206 pin 7 to Arduino pin 11 (Master Out Slave In - MOSI), and AD5206 pin 8 to Arduino pin 13 (Serial Clock - SCK).



Finally, connect an LED between each Wiper pin (AD5206 pins 2, 11, 14, 17, 20 and 23) and ground so that the long pin of the LED connects to the wiper and the short pin, or flat side of the LED connects to ground.



## Program the Arduino

Now we will write the code to enable SPI control of the AD5206. This program will sequentially pulse each LED on and then fade it out gradually. This is accomplished in the main loop of the program by individually changing the resistance of each potentiometer from full off to full on over its full range of 255 steps.

We will walk through the code in small sections.

We define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Although we are not reading any data back out of the AD5206 in this program, pin 12 is attached to the builtin SPI so it is best not to use it for other programming functions to avoid any possible errors:

```
#define DATAOUT 11//MOSI
#define DATAIN 12//MISO - not used, but part of builtin SPI
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss
```

Next we allocate variables to store resistance values and address values for the potentiometers:

```
byte pot=0;
byte resistance=0;
```

First we set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise:

```
void setup()
{
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
```

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variable to clear out any spurious data from past runs:

```
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
```

We conclude the setup function by setting all the potentiometers to full on resistance states thereby turning the LEDs off:

```
  for (i=0;i<6;i++)
  {
    write_pot(i,255);
  }
}
```

In our main loop we iterate through each resistance value (0-255) for each potentiometer address (0-5). We pause for 10 milliseconds each iteration to make the steps visible. This causes the LEDs to sequentially flash on brightly and then fade out slowly:

```
void loop()
{
     write_pot(pot,resistance);
     delay(10);
     resistance++;
     if (resistance==255)
     {
       pot++;
     }
     if (pot==6)
     {
       pot=0;
     }
}
```

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

The write_pot function allows us to control the individual potentiometers. We set the SLAVESELECT line low to enable the device. Then we transfer the address byte followed by the resistance value byte. Finally, we set the SLAVSELECT line high again to release the chip and signal the end of our data transfer.

```
byte write_pot(int address, int value)
{
  digitalWrite(SLAVESELECT,LOW);
  //2 byte opcode
  spi_transfer(address);
  spi_transfer(value);
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
}
```

LED video

For easy copy and pasting the full program text of this tutorial is below:

```
#define DATAOUT 11//MOSI
#define DATAIN 12//MISO - not used, but part of builtin SPI
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss
```

```
byte pot=0;
byte resistance=0;

char spi_transfer(volatile char data)
{
  SPDR = data;                      // Start the transmission
  while (!(SPSR & (1<<SPIF)))       // Wait the end of the transmission
  {
  };
  return SPDR;                      // return the received byte
}

void setup()
{
  byte i;
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  for (i=0;i<6;i++)
  {
    write_pot(i,255);
  }
}

byte write_pot(int address, int value)
{
  digitalWrite(SLAVESELECT,LOW);
  //2 byte opcode
  spi_transfer(address);
  spi_transfer(value);
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
}

void loop()
{
     write_pot(pot,resistance);
     delay(10);
     resistance++;
     if (resistance==255)
     {
       pot++;
     }
     if (pot==6)
     {
       pot=0;
     }
}
```

*code, tutorial and photos by Heather Dewey-Hagborg*

# Serial to Parallel Shifting-Out with a 74HC595

Started by Carlyn Maw and Tom Igoe Nov, 06

## Shifting Out & the 595 chip

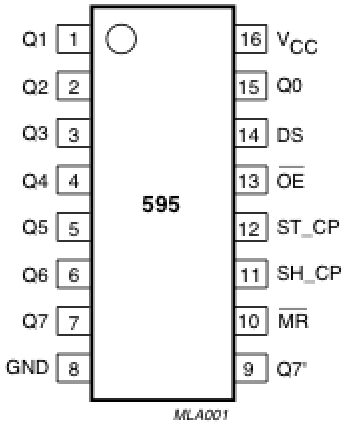
At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more. (Users may also wish to search for other driver chips with "595" or "596" in their part numbers, there are many. The STP16C596 for example will drive 16 LED's and eliminates the series resistors with built-in constant current sources.)

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. It's by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast to using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be set independently to an agreed upon specified data rate. Once the whole byte is transmitted to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

The “serial output” part of this component comes from its extra pin which can pass the serial information received from the microcontroller out again unchanged. This means you can transmit 16 bits in a row (2 bytes) and the first 8 will flow through the first register into the second register and be expressed there. You can learn to do that from the second example.

“3 states” refers to the fact that you can set the output pins as either high, low or “high impedance.” Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Neither example takes advantage of this feature and you won’t usually need to worry about getting a chip that has it.

**Here is a table explaining the pin-outs adapted from the Phillip's datasheet.**

| | | | |
|---|---|---|---|
|  | PINS 1-7, 15 | Q0 – Q7 | Output Pins |
| | PIN 8 | GND | Ground, Vss |
| | PIN 9 | Q7’ | Serial Out |
| | PIN 10 | MR | Master Reclear, active low |
| | PIN 11 | SH_CP | Shift register clock pin |
| | PIN 12 | ST_CP | Storage register clock pin (latch pin) |
| | PIN 13 | OE | Output enable, active low |
| | PIN 14 | DS | Serial data input |
| | PIN 16 | Vcc | Positive supply voltage |

## Example 1: One Shift Register

The first step is to extend your Arduino with one shift register.

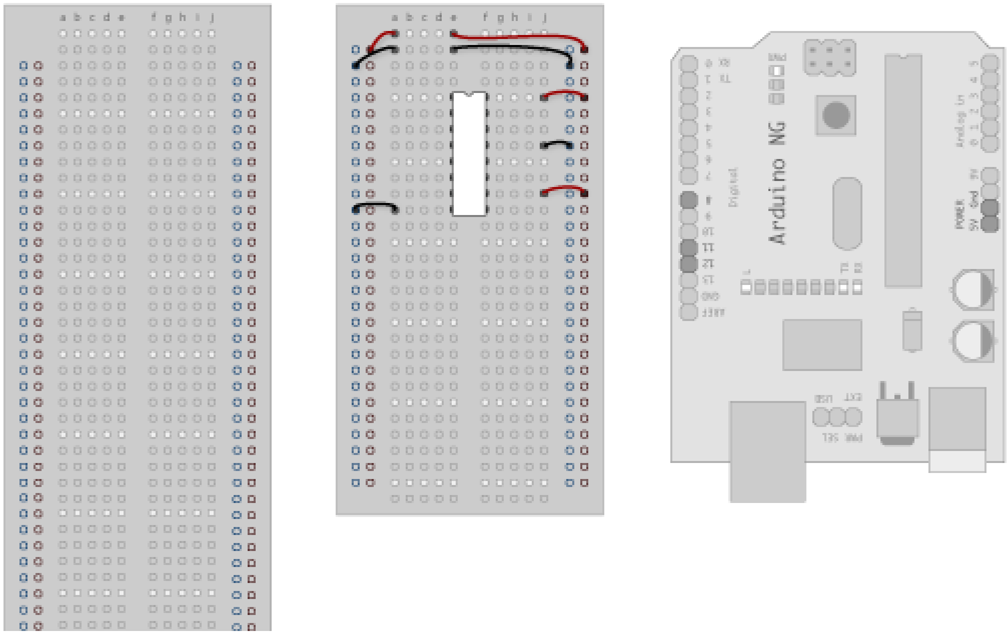
## The Circuit

### 1. Turning it on

Make the following connections:

- GND (pin 8) to ground,
- Vcc (pin 16) to 5V
- OE (pin 13) to ground
- MR (pin 10) to 5V

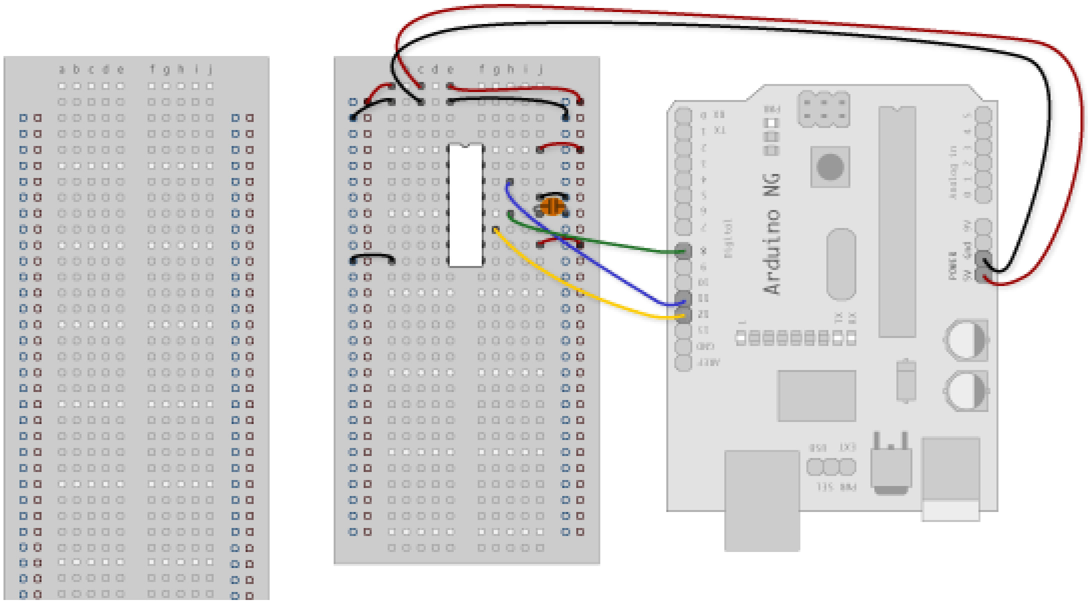
This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. You can get around this by controlling the MR and OE pins from your Arduino board too, but this way will work and leave you with more open pins.



### 2. Connect to Arduino

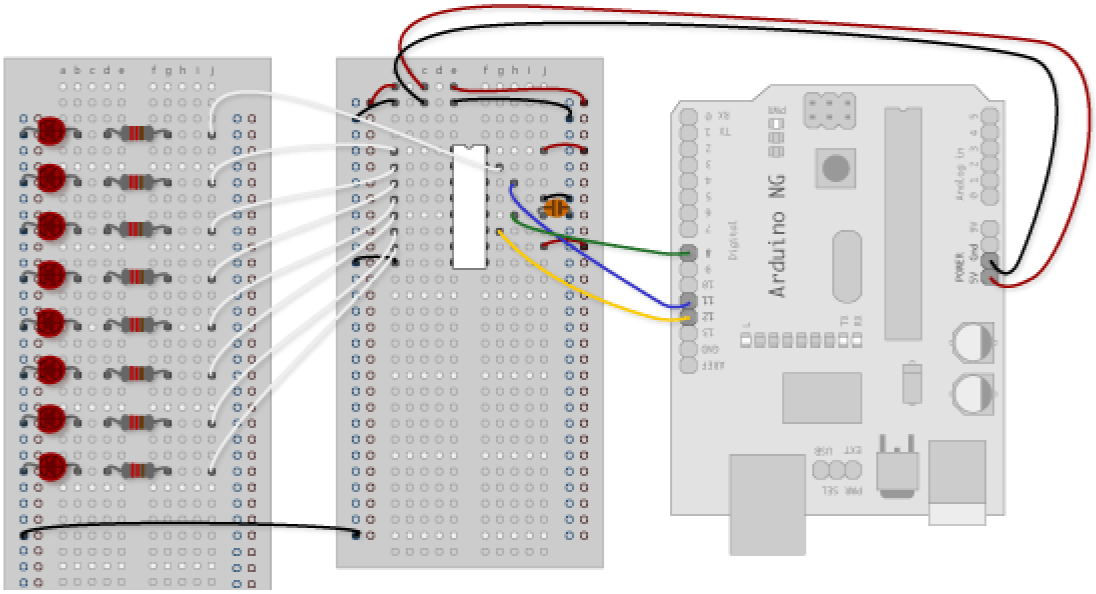
- DS (pin 14) to Ardunio DigitalPin 11 (blue wire)
- SH_CP (pin 11) to to Ardunio DigitalPin 12 (yellow wire)
- ST_CP (pin 12) to Ardunio DigitalPin 8 (green wire)

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin, if you have some flicker when the latch pin pulses you can use a capacitor to even it out.

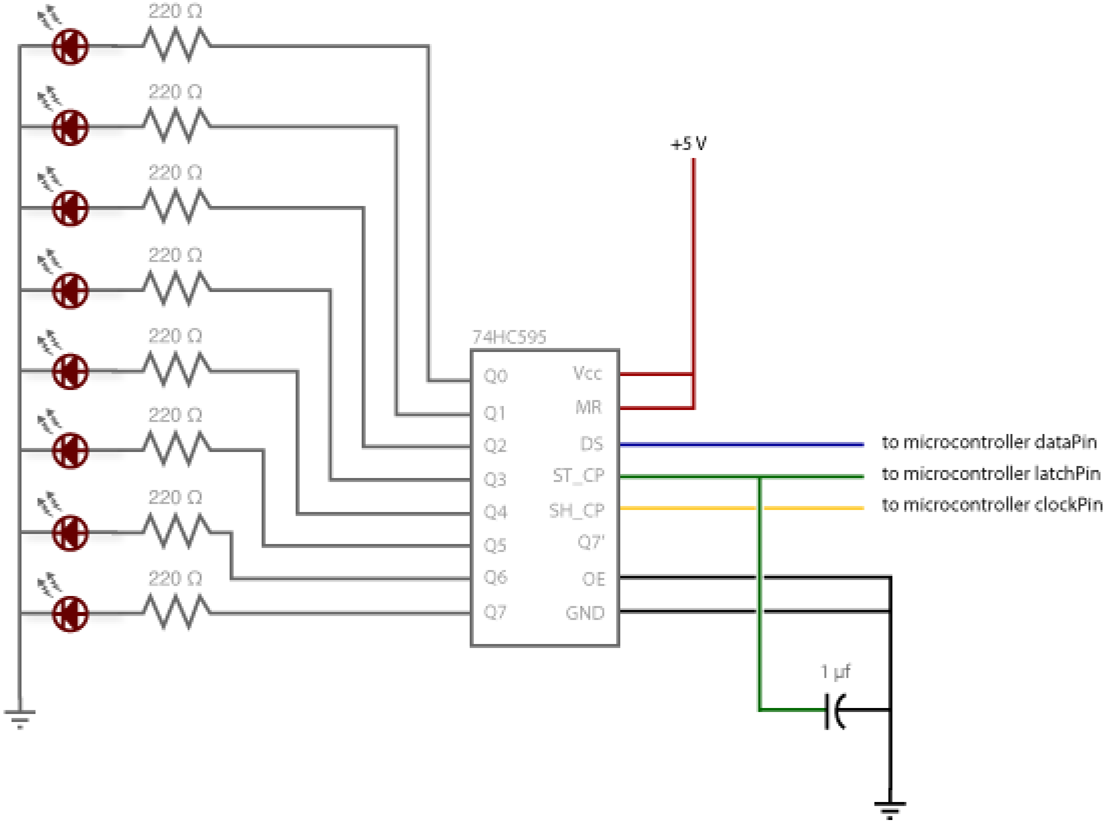


### 3. Add 8 LEDs.

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Using the shift register to supply power like this is called *sourcing current.* Some shift registers can't source current, they can only do what is called *sinking current.* If you have one of those it means you will have to flip the direction of the LEDs, putting the anodes directly to power and the cathodes (ground pins) to the shift register outputs. You should check the your specific datasheet if you aren’t using a 595 series chip. Don’t forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

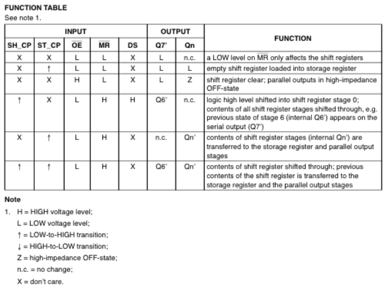


**Circuit Diagram**



### The Code

Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.



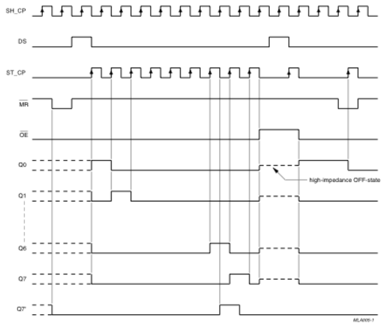
**FUNCTION TABLE**
See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{OE}$ | $\overline{MR}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{MR}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



595 Timing Diagram

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.

Code Sample 1.1 – Hello World
Code Sample 1.2 – One by One

Code Sample 1.3 – from Defined Array

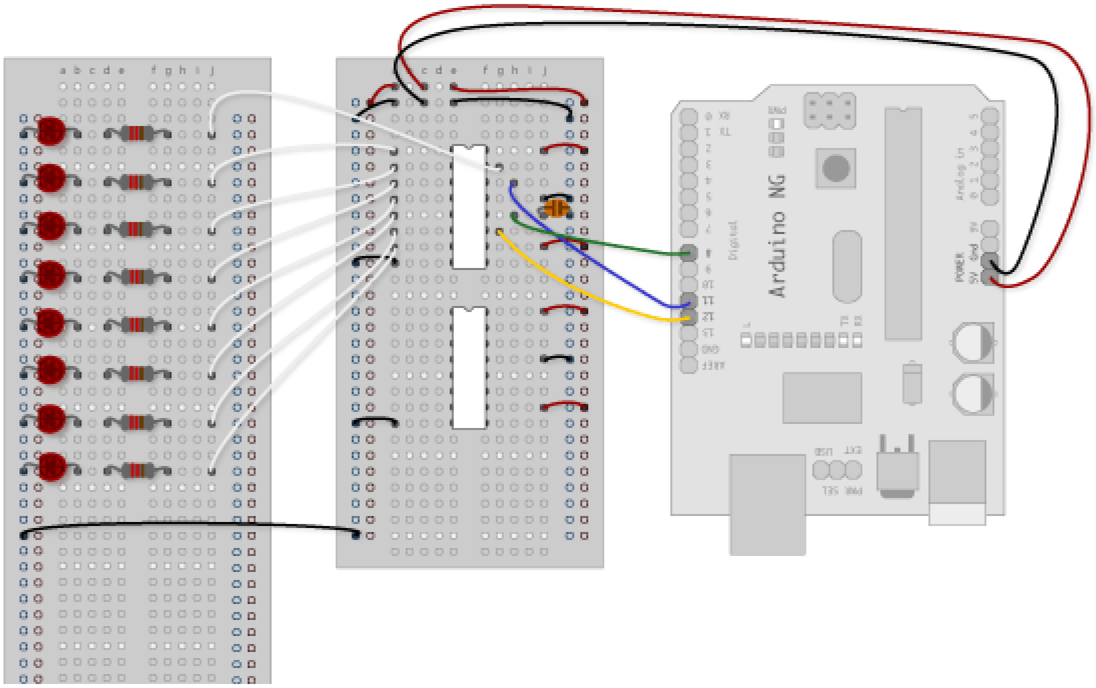
## Example 2

In this example you'll add a second shift register, doubling the number of output pins you have while still using the same number of pins from the Arduino.

### The Circuit

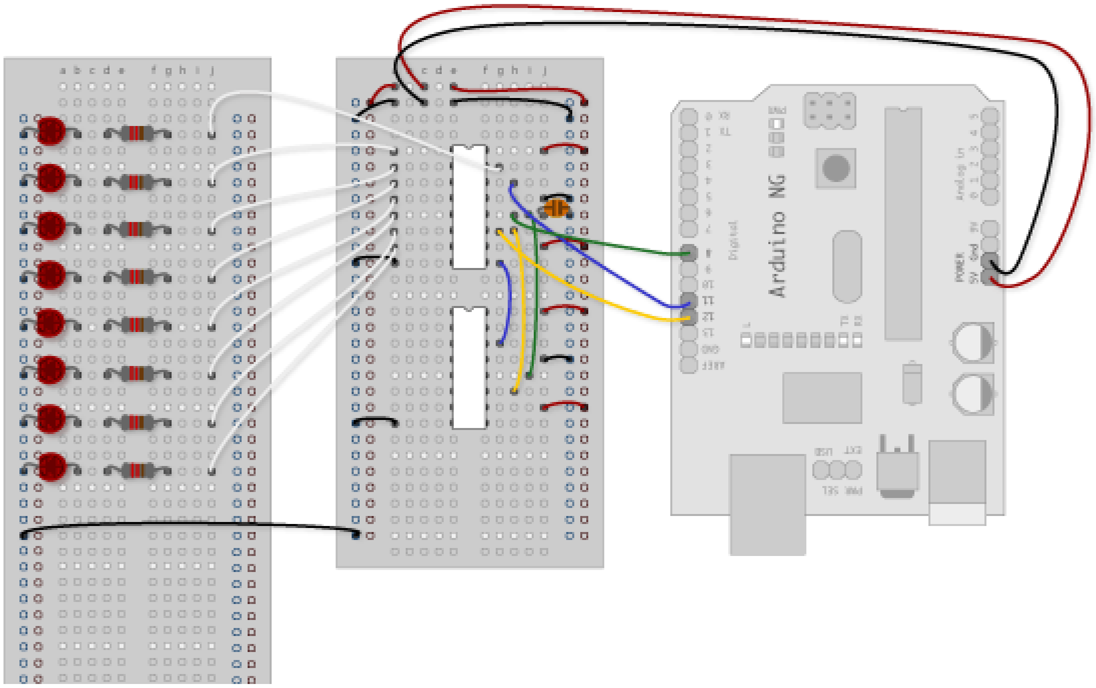
**1. Add a second shift register.**

Starting from the previous example, you should put a second shift register on the board. It should have the same leads to power and ground.



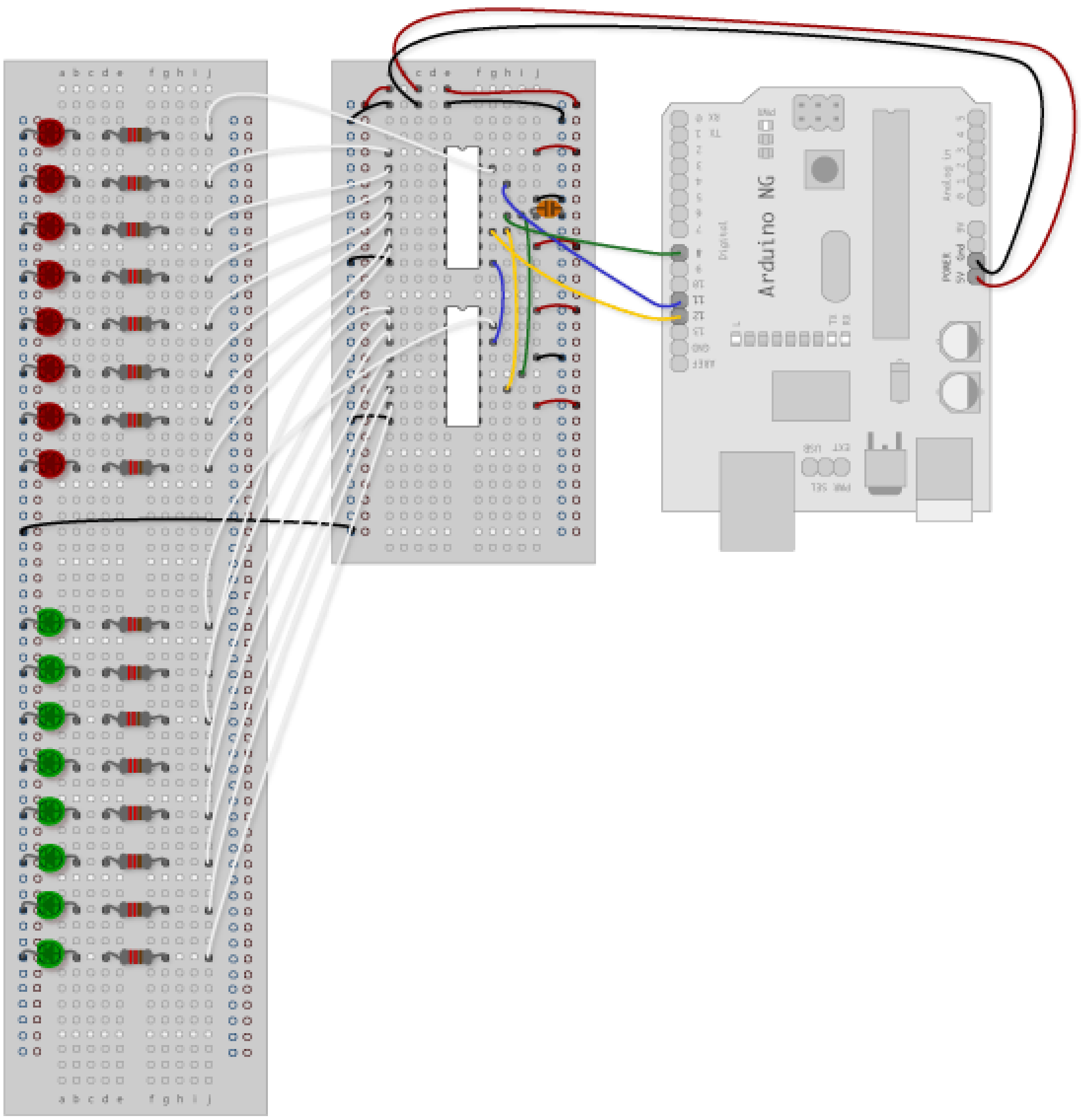
**2. Connect the 2 registers.**

Two of these connections simply extend the same clock and latch signal from the Arduino to the second shift register (yellow and green wires). The blue wire is going from the serial out pin (pin 9) of the first shift register to the serial data input (pin 14) of the second register.

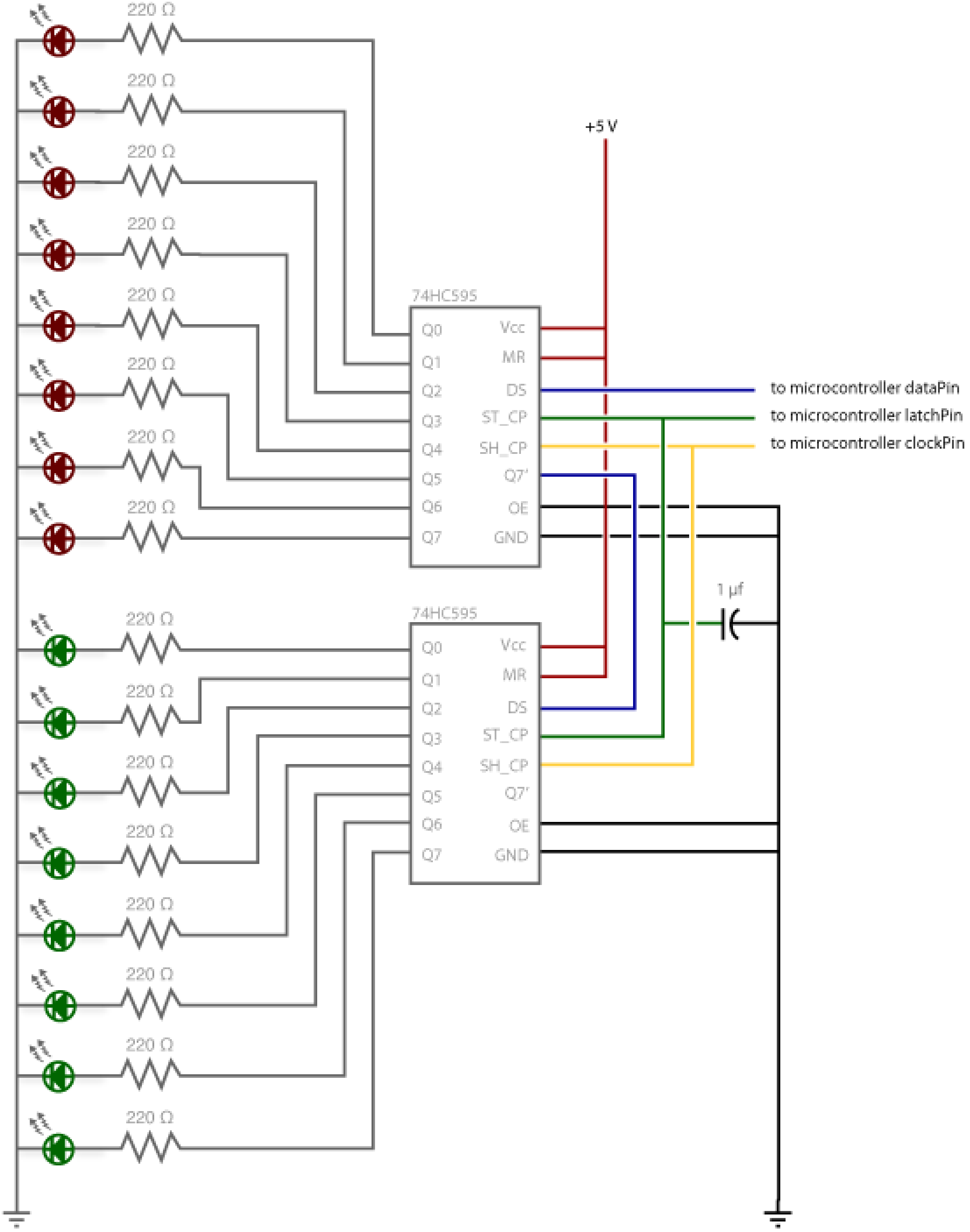


**3. Add a second set of LEDs.**

In this case I added green ones so when reading the code it is clear which byte is going to which set of LEDs



**Circuit Diagram**


220 Ω
+5 V
74HC595
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Vcc
MR
DS
ST_CP
SH_CP
Q7'
OE
GND
to microcontroller dataPin
to microcontroller latchPin
to microcontroller clockPin
1 µf

### The Code

Here again are three code samples. If you are curious, you might want to try the samples from the first example with this circuit set up just to see what happens.

Code Sample 2.1 – Dual Binary Counters
There is only one extra line of code compared to the first code sample from Example 1. It sends out a second byte. This forces the first shift register, the one directly attached to the Arduino, to pass the first byte sent through to the second register, lighting the green LEDs. The second byte will then show up on the red LEDs.

Code Sample 2.2 – 2 Byte One By One
Comparing this code to the similar code from Example 1 you see that a little bit more has had to change. The blinkAll() function has been changed to the blinkAll_2Bytes() function to reflect the fact that now there are 16 LEDs to control. Also, in version 1 the pulsings of the latchPin were situated inside the subfunctions lightShiftPinA and lightShiftPinB(). Here they need to be moved back into the main loop to accommodate needing to run each subfunction twice in a row, once for the green LEDs and once for the red ones.

Code Sample 2.3 - Dual Defined Arrays

Like sample 2.2, sample 2.3 also takes advantage of the new blinkAll_2bytes() function. 2.3's big difference from sample 1.3 is only that instead of just a single variable called “data” and a single array called “dataArray” you have to have a dataRED, a dataGREEN, dataArrayRED, dataArrayGREEN defined up front. This means that line

```
data = dataArray[j];
```

becomes

```
dataRED = dataArrayRED[j];
dataGREEN = dataArrayGREEN[j];
```

and

```
shiftOut(dataPin, clockPin, data);
```

becomes

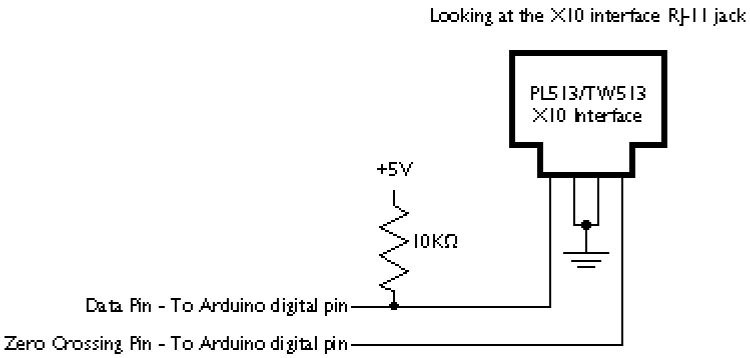
```
shiftOut(dataPin, clockPin, dataGREEN);
shiftOut(dataPin, clockPin, dataRED);
```

## X10 Library

This library enables you to send and receive X10 commands from an Arduino module. X10 is a synchronous serial protocol that travels over AC power lines, sending a bit every time the AC power crosses zero volts. It's used in home automation. You can find X10 controllers and devices at http://www.x10.com, http://www.smarthome.com, and more.

This library has been tested using the PL513 one-way X10 controller, and the TW523 two-way X10 controller. Both of these are essentially X10 modems, converting the 5V output of the Arduino into AC signals on the zero crossing.

To connect an Arduino to one of these modules, get a phone cable with an RJ-11 connector, and cut one end off. Then wire the pins as follows:


Looking at the X10 interface RJ-11 jack
PL513/TW513
X10 Interface
+5V
10KΩ
Data Pin - To Arduino digital pin
Zero Crossing Pin - To Arduino digital pin

Download: X10.zip

To use, unzip it and copy the resulting folder, called TextString, into the lib/targets/libraries directory of your arduino application folder. Then re-start the Arduino application.

When you restart, you'll see a few warning messages in the debugger pane at the bottom of the program. You can ignore them.

As of version 0.2, here's what you can do:

**x10(int strLength)** - initialize an instance of the X10 library on two digital pins. e.g.

```
x10 myHouse = x10(9, 10); // initializes X10 on pins 9 (zero crossing pin) and 10 (data pin)
```

**void write(byte houseCode, byte numberCode, int numRepeats)** - Send an X10 message, e.g.

```
myHouse.write(A, ALL_LIGHTS_ON, 1);     // Turns on all lights in house code A
```

**version(void)** - get the library version. Since there will be more functions added, printing the version is a useful debugging tool when you get an error from a given function. Perhaps you're using an earlier version that doesn't feature the version you need! e.g.

```
Serial.println(myHouse.version());   // prints the version of the library
```

There are a number of constants added to make X10 easier. They are as follows:

- A through F: house code values.
- UNIT_1 through UNIT_16: unit code values

- ALL_UNITS_OFF
- ALL_LIGHTS_ON
- ON
- OFF
- DIM
- BRIGHT
- ALL_LIGHTS_OFF
- EXTENDED_CODE
- HAIL_REQUEST
- HAIL_ACKNOWLEDGE
- PRE_SET_DIM
- EXTENDED_DATA
- STATUS_ON
- STATUS_OFF
- STATUS_REQUEST

For a full explanation of X10 and these codes, see this technote

---

If anyone's interested in helping to develop this library further, please contact me at tom.igoe at gmail.com

Edit Page | Page History | Printable View | All Recent Site Changes

*Examples > EEPROM Library*

# EEPROM Clear

Sets all of the bytes of the EEPROM to 0.

## Code

```
#include <EEPROM.h>

void setup()
{
  // write a 0 to all 512 bytes of the EEPROM
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, 0);

  // turn the LED on when we're done
  digitalWrite(13, HIGH);
}

void loop()
{
}
```

## See also

- EEPROM Read example
- EEPROM Write example
- EEPROM library reference

*Examples > EEPROM Library*

# EEPROM Read

Reads the value of each byte of the EEPROM and prints it to the computer.

## Code

```
#include <EEPROM.h>

// start reading from the first byte (address 0) of the EEPROM
int address = 0;
byte value;

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

void loop()
{
  // read a byte from the current address of the EEPROM
  value = EEPROM.read(address);

  Serial.print(address);
  Serial.print("\t");
  Serial.print(value, DEC);
  Serial.println();

  // advance to the next address of the EEPROM
  address = address + 1;

  // there are only 512 bytes of EEPROM, from 0 to 511, so if we're
  // on address 512, wrap around to address 0
  if (address == 512)
    address = 0;

  delay(500);
}
```

## See also

- EEPROM Clear example
- EEPROM Write example
- EEPROM library reference

*Examples > EEPROM Library*

# EEPROM Write

Stores values read from analog input 0 into the EEPROM. These values will stay in the EEPROM when the board is turned off and may be retrieved later by another sketch.

## Code

```
#include <EEPROM.h>

// the current address in the EEPROM (i.e. which byte
// we're going to write to next)
int addr = 0;

void setup()
{
}

void loop()
{
  // need to divide by 4 because analog inputs range from
  // 0 to 1023 and each byte of the EEPROM can only hold a
  // value from 0 to 255.
  int val = analogRead(0) / 4;

  // write the value to the appropriate byte of the EEPROM.
  // these values will remain there when the board is
  // turned off.
  EEPROM.write(addr, val);

  // advance to the next address.  there are 512 bytes in
  // the EEPROM, so go back to 0 when we hit 512.
  addr = addr + 1;
  if (addr == 512)
    addr = 0;

  delay(100);
}
```

## See also

- EEPROM Clear example
- EEPROM Read example
- EEPROM library reference

*Examples > Stepper Library*

# Motor Knob

### Description

A stepper motor follows the turns of a potentiometer (or other sensor) on analog input 0. The unipolar or bipolar stepper is controlled with pins 8, 9, 10, and 11, using one of the circuits on the linked pages.

### Code

```
#include <Stepper.h>

// change this to the number of steps on your motor
#define STEPS 100

// create an instance of the stepper class, specifying
// the number of steps of the motor and the pins it's
// attached to
Stepper stepper(STEPS, 8, 9, 10, 11);

// the previous reading from the analog input
int previous = 0;

void setup()
{
  // set the speed of the motor to 30 RPMs
  stepper.setSpeed(30);
}

void loop()
{
  // get the sensor value
  int val = analogRead(0);

  // move a number of steps equal to the change in the
  // sensor reading
  stepper.step(val - previous);

  // remember the previous value of the sensor
  previous = val;
}
```

### See also

- Stepper library reference

## Tutorial.HomePage History

Hide minor edits - Show changes to markup

July 02, 2008, at 03:11 PM by David A. Mellis -
Changed lines 2-3 from:

## Arduino Examples

to:

## Examples

Restore
July 02, 2008, at 03:11 PM by David A. Mellis -
Changed lines 4-5 from:

*See the **foundations page** for in-depth description of core concepts of the Arduino hardware and software, and the **links page** for other documentation.*

to:

*See the **foundations page** for in-depth description of core concepts of the Arduino hardware and software; the **hacking page** for information on extending and modifying the Arduino hardware and software; and the **links page** for other documentation.*

Restore
July 02, 2008, at 02:07 PM by David A. Mellis -
Added line 63:

- Read an ADXL3xx accelerometer

Restore
May 21, 2008, at 09:44 PM by David A. Mellis -
Deleted lines 42-45:

**Matrix Library**

- Hello Matrix?: blinks a smiley face on the LED matrix.

Restore
May 21, 2008, at 09:43 PM by David A. Mellis -
Added lines 43-46:

**Matrix Library**

- Hello Matrix?: blinks a smiley face on the LED matrix.

Restore
May 21, 2008, at 09:36 PM by David A. Mellis -
Added lines 43-46:

**Stepper Library**

- Motor Knob: control a stepper motor with a potentiometer.

Restore
May 21, 2008, at 09:25 PM by David A. Mellis - adding EEPROM examples.
Added lines 37-42:

**EEPROM Library**

- EEPROM Clear: clear the bytes in the EEPROM.
- EEPROM Read: read the EEPROM and send its values to the computer.
- EEPROM Write: stores values from an analog input to the EEPROM.

Restore
May 21, 2008, at 09:22 PM by David A. Mellis -
Changed line 15 from:

- BlinkWithoutDelay: blinking an LED without using the delay() function.

to:

- Blink Without Delay: blinking an LED without using the delay() function.

Restore
April 29, 2008, at 06:55 PM by David A. Mellis - moving the resources to the links page.
Changed lines 2-5 from:

## Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino Getting Started.

to:

## Arduino Examples

*See the **foundations page** for in-depth description of core concepts of the Arduino hardware and software, and the **links page** for other documentation.*

Added line 15:

- BlinkWithoutDelay: blinking an LED without using the delay() function.

Changed lines 37-42 from:

### Timing & Millis

- Blinking an LED without using the delay() function
- Stopwatch

(:if false:)

- TimeSinceStart:

(:ifend:)

to:

(:cell width=50%:)

Changed lines 41-42 from:

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

to:

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included on the page.

Deleted lines 43-44:
Added lines 49-51:

### Timing & Millis

- Stopwatch

Deleted lines 75-125:

(:cell width=50%:)

### Foundations

See the foundations page for explanations of the concepts involved in the Arduino hardware and software.

### Tutorials

Tutorials created by the Arduino community. Hosted on the publicly-editable playground wiki.

Board Setup and Configuration: Information about the components and usage of Arduino hardware.

Interfacing With Hardware: Code, circuits, and instructions for using various electronic components with an Arduino board.

- Output
- Input
- Interaction
- Storage
- Communication

Interfacing with Software: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

Code Library and Tutorials: Arduino functions for performing specific tasks and other programming tutorials.

Electronics Techniques: tutorials on soldering and other electronics resources.

### Manuals, Curricula, and Other Resources

Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Examples from Tom Igoe

Examples from Jeff Gray

Restore
April 23, 2008, at 10:29 PM by David A. Mellis -
Changed line 6 from:

(:table width=90% border=0 cellpadding=5 cellspacing=0:)

to:

(:table width=100% border=0 cellpadding=5 cellspacing=0:)

Restore
April 22, 2008, at 05:59 PM by Paul Badger -
Changed line 39 from:
to:

(:if false:)

Changed line 41 from:
to:

(:ifend:)

Restore
April 22, 2008, at 05:56 PM by Paul Badger -
Added lines 40-41:

- TimeSinceStart:

Restore
April 18, 2008, at 07:22 AM by Paul Badger -
Added lines 36-39:

### Timing & Millis

- Blinking an LED without using the delay() function
- Stopwatch

Changed line 46 from:

- Blinking an LED without using the delay() function

to:
Restore
April 08, 2008, at 08:23 PM by David A. Mellis -
Changed line 43 from:

- * TwoSwitchesOnePin: Read two switches with one I/O pin

to:

- TwoSwitchesOnePin: Read two switches with one I/O pin

Restore
April 08, 2008, at 08:22 PM by David A. Mellis - moving TwoSwitchesOnePin to "other examples" since it's not (yet) in the distribution.
Changed lines 18-19 from:

- TwoSwitchesOnePin: Read two switches with one I/O pin

to:
Added line 43:

- * TwoSwitchesOnePin: Read two switches with one I/O pin

Restore
April 08, 2008, at 07:41 PM by Paul Badger -
Changed lines 18-19 from:
to:

- TwoSwitchesOnePin: Read two switches with one I/O pin

Restore
March 09, 2008, at 07:20 PM by David A. Mellis -
Changed lines 73-78 from:

- Foundations has moved here

- Bootloader: A small program pre-loaded on the Arduino board to allow uploading sketches.

to:

See the foundations page for explanations of the concepts involved in the Arduino hardware and software.

Restore
March 07, 2008, at 09:26 PM by Paul Badger -
Changed lines 73-75 from:

to:

- Foundations has moved here

Restore
March 07, 2008, at 09:24 PM by Paul Badger -
Changed lines 74-107 from:

- Memory: The various types of memory available on the Arduino board.
- Digital Pins: How the pins work and what it means for them to be configured as inputs or outputs.
- Analog Input Pins: Details about the analog-to-digital conversion and other uses of the pins.
- Foundations

(:if false:)

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.
- Communication?: An overview of the various ways in which an Arduino board can communicate with other devices (serial, I2C, SPI, Midi, etc.)
- Serial Communication?: How to send serial data from an Arduino board to a computer or other device (including via the USB connection).
- Interrupts?: Code that interrupts other code under certain conditions.
- Numbers?: The various types of numbers available and how to use them.
- Variables: How to define and use variables.
- Arrays?: How to store multiple values of the same type.
- Pointers?:
- Functions?: How to write and call functions.
- Optimization?: What to do when your program runs too slowly.
- Debugging?: Figuring out what's wrong with your hardware or software and how to fix it.

(:ifend:)

to:
Restore
March 07, 2008, at 09:09 PM by Paul Badger -
Added lines 80-81:

- Foundations

Restore
February 15, 2008, at 06:00 PM by David A. Mellis -
Changed lines 72-73 from:

### Tutorials

to:

### Foundations

Changed lines 108-109 from:

### More Tutorials

to:

### Tutorials

Restore
February 13, 2008, at 10:42 PM by Paul Badger -
Changed lines 4-5 from:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide.

to:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino Getting Started.

Restore
February 13, 2008, at 10:06 PM by David A. Mellis -
Restore
February 13, 2008, at 09:58 PM by David A. Mellis -
Added lines 100-103:

- Optimization?: What to do when your program runs too slowly.

- Debugging?: Figuring out what's wrong with your hardware or software and how to fix it.

Restore
February 13, 2008, at 09:41 PM by David A. Mellis -
Added lines 90-99:

- Numbers?: The various types of numbers available and how to use them.

- Variables: How to define and use variables.

- Arrays?: How to store multiple values of the same type.

- Pointers?:

- Functions?: How to write and call functions.

Restore
February 13, 2008, at 09:38 PM by David A. Mellis -
Changed lines 86-87 from:

- Serial Communication?: How to send serial data from an Arduino board to a computer or other device.

to:

- Serial Communication?: How to send serial data from an Arduino board to a computer or other device (including via the USB connection).

- Interrupts?: Code that interrupts other code under certain conditions.

Restore
February 13, 2008, at 09:36 PM by David A. Mellis -
Added lines 80-81:

(:if false:)

Added lines 84-89:

- Communication?: An overview of the various ways in which an Arduino board can communicate with other devices (serial, I2C, SPI, Midi, etc.)

- Serial Communication?: How to send serial data from an Arduino board to a computer or other device.

(:ifend:)

Restore
February 13, 2008, at 09:31 PM by David A. Mellis -
Changed lines 80-81 from:

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.

to:

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.

Restore
February 13, 2008, at 09:30 PM by David A. Mellis -
Added lines 80-81:

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.

Restore
February 13, 2008, at 09:22 PM by David A. Mellis -
Added lines 80-81:

- Bootloader: A small program pre-loaded on the Arduino board to allow uploading sketches.

Restore
February 13, 2008, at 09:12 PM by David A. Mellis -
Added lines 74-81:

- Memory: The various types of memory available on the Arduino board.

- Digital Pins: How the pins work and what it means for them to be configured as inputs or outputs.

- Analog Input Pins: Details about the analog-to-digital conversion and other uses of the pins.

## More Tutorials

Restore
January 11, 2008, at 11:31 AM by David A. Mellis - linking to board setup and configuration on the playground.
Added lines 76-77:

Board Setup and Configuration: Information about the components and usage of Arduino hardware.

Restore
December 19, 2007, at 11:54 PM by David A. Mellis - adding links to other pages: the tutorial parts of the playground, ladyada's tutorials, todbot, etc.
Changed lines 36-42 from:

(:cell width=50%:)

## Tutorials

These are more complex tutorials for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

to:

## Other Examples

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

Changed lines 71-78 from:

### Other Arduino Tutorials

- Tutorials from the Arduino playground
- Example labs from ITP
- Spooky Arduino and more from Todbot
- Examples from Tom Igoe
- Examples from Jeff Gray

to:

(:cell width=50%:)

## Tutorials

Tutorials created by the Arduino community. Hosted on the publicly-editable playground wiki.

Interfacing With Hardware: Code, circuits, and instructions for using various electronic components with an Arduino board.

- Output
- Input
- Interaction
- Storage
- Communication

Interfacing with Software: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

Code Library and Tutorials: Arduino functions for performing specific tasks and other programming tutorials.

Electronics Techniques: tutorials on soldering and other electronics resources.

### Manuals, Curricula, and Other Resources

Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Examples from Tom Igoe

Examples from Jeff Gray

Restore
December 13, 2007, at 11:08 PM by David A. Mellis - adding debounce example.
Added line 16:

- Debounce: read a pushbutton, filtering noise.

Restore
August 28, 2007, at 11:15 PM by Tom Igoe -
Changed lines 71-72 from:
to:

- X10 output control devices over AC powerlines using X10

Restore
June 15, 2007, at 05:04 PM by David A. Mellis - adding link to Processing (for the communication examples)
Added lines 27-28:

*These examples include code that allows the Arduino to talk to Processing sketches running on the computer. For more information or to download Processing, see processing.org.*

Restore
June 12, 2007, at 08:57 AM by David A. Mellis - removing link to obsolete joystick example.
Deleted line 43:

- Interfacing a Joystick

Restore
June 11, 2007, at 11:14 PM by David A. Mellis -
Changed lines 10-11 from:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.)

Restore
June 11, 2007, at 11:13 PM by David A. Mellis -
Changed lines 10-11 from:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder.

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.

Restore
June 11, 2007, at 11:10 PM by David A. Mellis - updating to 0008 examples
Changed lines 10-11 from:

**Digital Output**

- Blinking LED

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder.

**Digital I/O**

- Blink: turn an LED on and off.
- Button: use a pushbutton to control an LED.
- Loop: controlling multiple LEDs with a loop and an array.

**Analog I/O**

- Analog Input: use a potentiometer to control the blinking of an LED.
- Fading: uses an analog output (PWM pin) to fade an LED.
- Knock: detect knocks with a piezo element.
- Smoothing: smooth multiple readings of an analog input.

**Communication**

- ASCII Table: demonstrates Arduino's advanced serial output functions.
- Dimmer: move the mouse to change the brightness of an LED.
- Graph: sending data to the computer and graphing it in Processing.
- Physical Pixel: turning on and off an LED by sending data from Processing.
- Virtual Color Mixer: sending multiple variables from Arduino to the computer and reading them in Processing.

(:cell width=50%:)

## Tutorials

These are more complex tutorials for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

**Miscellaneous**

Deleted lines 42-51:

- Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- More complex dimming/color crossfader
- Knight Rider example
- Shooting star
- PWM all of the digital pins in a sinewave pattern

**Digital Input**

- Digital Input and Output (from ITP physcomp labs)

- Read a Pushbutton
- Using a pushbutton as a switch

Deleted lines 43-45:

#### Analog Input

- Read a Potentiometer

Deleted lines 45-46:

- Read a Piezo Sensor
- 3 LED cross-fades with a potentiometer

Changed lines 52-53 from:

- Use two Arduino pins as a capacitive sensor

to:
Deleted line 54:

- More sound ideas

Added line 64:

- Build your own DMX Master device

Changed lines 70-72 from:

- Multiple digital inputs with a CD4021 Shift Register

### Other Arduino Examples

to:

#### Other Arduino Tutorials

- Tutorials from the Arduino playground

Added line 75:

- Spooky Arduino and more from Todbot

Deleted lines 78-105:

(:cell width=50%:)

### Interfacing with Other Software

- Introduction to Serial Communication (from ITP physcomp labs)
- Arduino + Flash
- Arduino + Processing
- Arduino + PD
- Arduino + MaxMSP
- Arduino + VVVV
- Arduino + Director
- Arduino + Ruby
- Arduino + C

### Tech Notes (from the forums or playground)

- Software serial (serial on pins besides 0 and 1)
- L297 motor driver
- Hex inverter
- Analog multiplexer
- Power supplies
- The components on the Arduino board
- Arduino build process
- AVRISP mkII on the Mac
- Non-volatile memory (EEPROM)
- Bluetooth
- Zigbee

- LED as light sensor (en Francais)
- Arduino and the Asuro robot
- Using Arduino from the command line

Restore
May 11, 2007, at 06:06 AM by Paul Badger -
Changed lines 17-18 from:
to:

- PWM all of the digital pins in a sinewave pattern

Restore
May 10, 2007, at 07:07 PM by Paul Badger -
Changed lines 36-37 from:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 |Use a couple of Arduino pins as a capacitive sensor]]

to:

- Use two Arduino pins as a capacitive sensor

Restore
May 10, 2007, at 07:05 PM by Paul Badger -
Changed lines 36-37 from:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 Use a couple of Arduino pins as a capacitive sensor

to:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 |Use a couple of Arduino pins as a capacitive sensor]]

Restore
May 10, 2007, at 07:04 PM by Paul Badger -
Changed lines 36-37 from:
to:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 Use a couple of Arduino pins as a capacitive sensor

Restore
May 10, 2007, at 06:59 PM by Paul Badger -
Added line 39:

- More sound ideas

Restore
April 24, 2007, at 03:40 PM by Clay Shirky -
Changed lines 13-14 from:

- Dimming 3 LEDs with Pulse-Width Modulation (PWM)

to:

- Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- More complex dimming/color crossfader

Restore
February 08, 2007, at 12:02 PM by Carlyn Maw -
Changed lines 52-53 from:
to:

- Multiple digital inputs with a CD4021 Shift Register

Restore
February 06, 2007, at 02:52 PM by Carlyn Maw -
Changed lines 52-54 from:

- Multiple digital ins with a CD4021 Shift Register

to:
Restore

February 06, 2007, at 02:51 PM by Carlyn Maw -
Changed lines 52-53 from:
to:

- Multiple digital ins with a CD4021 Shift Register

Restore
January 30, 2007, at 03:37 PM by David A. Mellis -
Deleted line 46:

- Build your own DMX Master device

Restore
December 25, 2006, at 11:57 PM by David A. Mellis -
Added line 20:

- Using a pushbutton as a switch

Restore
December 07, 2006, at 06:04 AM by David A. Mellis - adding link to todbot's C serial port code.
Changed lines 69-70 from:
to:

- Arduino + C

Restore
December 02, 2006, at 10:43 AM by David A. Mellis -
Added line 1:

(:title Tutorials:)

Restore
November 21, 2006, at 10:13 AM by David A. Mellis -
Added line 64:

- Arduino + MaxMSP

Changed lines 67-68 from:
to:

- Arduino + Ruby

Restore
November 18, 2006, at 02:42 AM by David A. Mellis -
Changed lines 20-21 from:

- Controlling an LED circle with a joystick

to:
Added line 24:

- Controlling an LED circle with a joystick

Restore
November 09, 2006, at 03:10 PM by Carlyn Maw -
Changed lines 50-51 from:
to:

- Multiple digital outs with a 595 Shift Register

Restore
November 06, 2006, at 10:49 AM by David A. Mellis -
Changed lines 37-38 from:

- MIDI Output (from ITP physcomp labs)

to:

- MIDI Output (from ITP physcomp labs) and from Spooky Arduino

Restore
November 04, 2006, at 12:25 PM by David A. Mellis -
Deleted line 53:

Deleted line 54:
Restore
November 04, 2006, at 12:24 PM by David A. Mellis -
Added lines 51-58:

### Other Arduino Examples

- Example labs from ITP
- Examples from Tom Igoe
- Examples from Jeff Gray

Deleted lines 83-89:

### Other Arduino Examples

- Example labs from ITP
- Examples from Tom Igoe.
- Examples from Jeff Gray.

Restore
November 04, 2006, at 12:24 PM by David A. Mellis -
Changed lines 50-51 from:

**Example labs from ITP**

to:
Changed lines 77-78 from:

Also, see the examples from Tom Igoe and those from Jeff Gray.

to:

- Example labs from ITP
- Examples from Tom Igoe.
- Examples from Jeff Gray.

Restore
November 04, 2006, at 12:23 PM by David A. Mellis -
Changed line 77 from:

### Other Arduino Sites

to:

### Other Arduino Examples

Deleted lines 79-81:

### Do you need extra help?

Is there a sensor you would like to see characterized for Arduino, or is there something you would like to see published in this site? Refer to the forum for further help.

Restore
November 04, 2006, at 10:38 AM by David A. Mellis -
Changed lines 3-4 from:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide?.

to:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide.

Restore
November 04, 2006, at 10:37 AM by David A. Mellis - lots of content moved to the new guide.
Deleted lines 52-67:

### The Arduino board

This guide to the Arduino board explains the functions of the various parts of the board.

### The Arduino environment

This guide to the Arduino IDE (integrated development environment) explains the functions of the various buttons and menus.

The libraries page explains how to use libraries in your sketches and how to make your own.

### Video Lectures by Tom Igoe

Watch Tom introduce Arduino. Thanks to Pollie Barden for the great videos.

### Course Guides

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm), class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Deleted lines 82-87:

### External Resources

Instant Soup is an introduction to electronics through a series of beautifully-documented fun projects.

Make magazine has some great links in its electronics archive.

hack a day has links to interesting hacks and how-to articles on various topics.

Restore
November 04, 2006, at 10:17 AM by David A. Mellis -
Changed lines 3-4 from:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Howto.

to:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide?.

Restore
November 01, 2006, at 06:54 PM by Carlyn Maw -
Deleted line 49:

- Extend your digital outs with 74HC595 shift registers

Restore
November 01, 2006, at 06:06 PM by Carlyn Maw -
Added line 50:

- Extend your digital outs with 74HC595 shift registers

Restore
October 31, 2006, at 10:47 AM by Tod E. Kurt -
Changed lines 67-68 from:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm).

to:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm), class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Restore
October 22, 2006, at 12:52 PM by David A. Mellis -
Changed lines 1-4 from:

## Learning to use Arduino

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto.

to:

## Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Howto.

Restore
October 22, 2006, at 12:51 PM by David A. Mellis -
Changed lines 67-68 from:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors).

to:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm).

Restore
October 21, 2006, at 04:25 PM by David A. Mellis - adding links to todbot's class notes.
Added lines 66-68:

### Course Guides

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors).

Restore
October 08, 2006, at 05:46 PM by David A. Mellis -
Changed lines 59-62 from:

This guide to the Arduino IDE? (integrated development environment) explains the functions of the various buttons and menus.

The libraries? page explains how to use libraries in your sketches and how to make your own.

to:

This guide to the Arduino IDE (integrated development environment) explains the functions of the various buttons and menus.

The libraries page explains how to use libraries in your sketches and how to make your own.

Restore
October 08, 2006, at 05:45 PM by David A. Mellis -
Changed lines 3-4 from:

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto?.

to:

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto.

Restore
October 08, 2006, at 05:38 PM by David A. Mellis -
Added lines 1-102:

## Learning to use Arduino

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto?.

(:table width=90% border=0 cellpadding=5 cellspacing=0:) (:cell width=50%:)

## Examples

### Digital Output

- Blinking LED
- Blinking an LED without using the delay() function
- Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- Knight Rider example
- Shooting star

### Digital Input

- Digital Input and Output (from ITP physcomp labs)
- Read a Pushbutton
- Read a Tilt Sensor
- Controlling an LED circle with a joystick

### Analog Input

- Read a Potentiometer
- Interfacing a Joystick
- Read a Piezo Sensor
- 3 LED cross-fades with a potentiometer
- 3 LED color mixer with 3 potentiometers

### Complex Sensors

- Read an Accelerometer
- Read an Ultrasonic Range Finder (ultrasound sensor)
- Reading the qprox qt401 linear touch sensor

### Sound

- Play Melodies with a Piezo Speaker
- Play Tones from the Serial Connection
- MIDI Output (from ITP physcomp labs)

### Interfacing w/ Hardware

- Multiply the Amount of Outputs with an LED Driver
- Interfacing an LCD display with 8 bits
    - LCD interface library
- Driving a DC Motor with an L293 (from ITP physcomp labs).
- Driving a Unipolar Stepper Motor
- Build your own DMX Master device
- Implement a software serial connection
    - RS-232 computer interface
- Interface with a serial EEPROM using SPI
- Control a digital potentiometer using SPI

### Example labs from ITP

(:cell width=50%:)

## The Arduino board

This guide to the Arduino board explains the functions of the various parts of the board.

## The Arduino environment

This guide to the Arduino IDE? (integrated development environment) explains the functions of the various buttons and menus.

The libraries? page explains how to use libraries in your sketches and how to make your own.

## Video Lectures by Tom Igoe

Watch Tom introduce Arduino. Thanks to Pollie Barden for the great videos.

## Interfacing with Other Software

- Introduction to Serial Communication (from ITP physcomp labs)
- Arduino + Flash
- Arduino + Processing
- Arduino + PD
- Arduino + VVVV
- Arduino + Director

## Tech Notes (from the forums or playground)

- Software serial (serial on pins besides 0 and 1)
- L297 motor driver
- Hex inverter
- Analog multiplexer
- Power supplies
- The components on the Arduino board
- Arduino build process
- AVRISP mkII on the Mac
- Non-volatile memory (EEPROM)
- Bluetooth
- Zigbee
- LED as light sensor (en Francais)
- Arduino and the Asuro robot
- Using Arduino from the command line

## Other Arduino Sites

Also, see the examples from Tom Igoe and those from Jeff Gray.

## Do you need extra help?

Is there a sensor you would like to see characterized for Arduino, or is there something you would like to see published in this site? Refer to the forum for further help.

## External Resources

Instant Soup is an introduction to electronics through a series of beautifully-documented fun projects.

Make magazine has some great links in its electronics archive.

hack a day has links to interesting hacks and how-to articles on various topics. (:tableend:)

Restore

**Learning** Examples | Foundations | Hacking | Links

# Examples

*See the foundations page for in-depth description of core concepts of the Arduino hardware and software; the hacking page for information on extending and modifying the Arduino hardware and software; and the links page for other documentation.*

## Examples

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.)

### Digital I/O

- Blink: turn an LED on and off.
- Blink Without Delay: blinking an LED without using the delay() function.
- Button: use a pushbutton to control an LED.
- Debounce: read a pushbutton, filtering noise.
- Loop: controlling multiple LEDs with a loop and an array.

### Analog I/O

- Analog Input: use a potentiometer to control the blinking of an LED.
- Fading: uses an analog output (PWM pin) to fade an LED.
- Knock: detect knocks with a piezo element.
- Smoothing: smooth multiple readings of an analog input.

### Communication

*These examples include code that allows the Arduino to talk to Processing sketches running on the computer. For more information or to download Processing, see processing.org.*

- ASCII Table: demonstrates Arduino's advanced serial output functions.
- Dimmer: move the mouse to change the brightness of an LED.
- Graph: sending data to the computer and graphing it in Processing.
- Physical Pixel: turning on and off an LED by sending data from Processing.
- Virtual Color Mixer: sending multiple variables from Arduino to the computer and reading them in Processing.

## Other Examples

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included on the page.

### Miscellaneous

- TwoSwitchesOnePin: Read two switches with one I/O pin
- Read a Tilt Sensor
- Controlling an LED circle with a joystick
- 3 LED color mixer with 3 potentiometers

## Timing & Millis

- Stopwatch

### Complex Sensors

- Read an ADXL3xx accelerometer
- Read an Accelerometer
- Read an Ultrasonic Range Finder (ultrasound sensor)
- Reading the qprox qt401 linear touch sensor

### Sound

- Play Melodies with a Piezo Speaker
- Play Tones from the Serial Connection
- MIDI Output (from ITP physcomp labs) and from Spooky Arduino

### Interfacing w/ Hardware

- Multiply the Amount of Outputs with an LED Driver
- Interfacing an LCD display with 8 bits
  - LCD interface library
- Driving a DC Motor with an L293 (from ITP physcomp labs).
- Driving a Unipolar Stepper Motor
- Build your own DMX Master device
- Implement a software serial connection
  - RS-232 computer interface
- Interface with a serial EEPROM using SPI

## EEPROM Library

- EEPROM Clear: clear the bytes in the EEPROM.
- EEPROM Read: read the EEPROM and send its values to the computer.
- EEPROM Write: stores values from an analog input to the EEPROM.

## Stepper Library

- Motor Knob: control a stepper motor with a potentiometer.

- Control a digital potentiometer using SPI
- Multiple digital outs with a 595 Shift Register
- X10 output control devices over AC powerlines using X10

## Foundations Page Discussion

The Foundations page is intended to supplement the material in the examples and reference, providing more in-depth explanations of the underlying functionality and principles involved.

These pages are cross-linked with the applicable language reference, example, and other pages, providing a single source for people looking for a longer discussion of a particular topic.

This section is a work in progress, and there are many topics yet to be covered. Here's a rough list of ideas:

- PROGRAMMING
  - conditionals
  - loops
  - functions
  - numbers and arithmetic
  - bits and bytes
  - characters and encodings
  - arrays
  - strings
- ELECTRONICS
  - voltage, current, and resistance
  - resistive sensors
  - capacitors
  - transistors
  - power
  - noise
- COMMUNICATION
  - serial communication
  - i2c (aka twi)
  - bluetooth
- MICROCONTROLLER
  - reset
  - pins and ports
  - interrupts

If you see anything in the list that interests you, feel free to take a shot at writing it up. Don't worry if it's not finished or polished, we can always edit and improve it. You can post works-in-progress to the playground and mention them on the forum. Also, be sure to let us know if you think there's anything that we've forgotten, or if you have other suggestions.

Foundations Page

# First Sketch

In the getting started guide (Windows, Mac OS X, Linux), you uploaded a sketch that blinks an LED. In this tutorial, you'll learn how each part of that sketch works.

## Sketch

A *sketch* is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board.

## Comments

The first few lines of the Blink sketch are a *comment*:

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

Everything between the `/*` and `*/` is ignored by the Arduino when it runs the sketch (the `*` at the start of each line is only there to make the comment look pretty, and isn't required). It's there for people reading the code: to explain what the program does, how it works, or why it's written the way it is. It's a good practice to comment your sketches, and to keep the comments up-to-date when you modify the code. This helps other people to learn from or modify your code.

There's another style for short, single-line comments. These start with `//` and continue to the end of the line. For example, in the line:

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

the message "LED connected to digital pin 13" is a comment.

## Variables

A *variable* is a place for storing a piece of data. It has a name, a type, and a value. For example, the line from the Blink sketch above declares a variable with the name `ledPin`, the type `int`, and an initial value of 13. It's being used to indicate which Arduino pin the LED is connected to. Every time the name `ledPin` appears in the code, its value will be retrieved. In this case, the person writing the program could have chosen not to bother creating the `ledPin` variable and instead have simply written 13 everywhere they needed to specify a pin number. The advantage of using a variable is that it's easier to move the LED to a different pin: you only need to edit the one line that assigns the initial value to the variable.

Often, however, the value of a variable will change while the sketch runs. For example, you could store the value read from an input into a variable. There's more information in the Variables tutorial.

## Functions

A *function* (otherwise known as a *procedure* or *sub-routine*) is a named piece of code that can be used from elsewhere in a sketch. For example, here's the definition of the `setup()` function from the Blink example:

```
void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}
```

The first line provides information about the function, like its name, "setup". The text before and after the name specify its return type and parameters: these will be explained later. The code between the { and } is called the *body* of the function: what the function does.

You can *call* a function that's already been defined (either in your sketch or as part of the Arduino language). For example, the line `pinMode(ledPin, OUTPUT);` calls the `pinMode()` function, passing it two *parameters*: `ledPin` and `OUTPUT`. These parameters are used by the `pinMode()` function to decide which pin and mode to set.

### pinMode(), digitalWrite(), and delay()

The `pinMode()` function configures a pin as either an input or an output. To use it, you pass it the number of the pin to configure and the constant INPUT or OUTPUT. When configured as an input, a pin can detect the state of a sensor like a pushbutton; this is discussed in a later tutorial?. As an output, it can drive an actuator like an LED.

The `digitalWrite()` functions outputs a value on a pin. For example, the line:

```
digitalWrite(ledPin, HIGH);
```

set the `ledPin` (pin 13) to HIGH, or 5 volts. Writing a LOW to pin connects it to ground, or 0 volts.

The `delay()` causes the Arduino to wait for the specified number of milliseconds before continuing on to the next line. There are 1000 milliseconds in a second, so the line:

```
delay(1000);
```

creates a delay of one second.

### setup() and loop()

There are two special functions that are a part of every Arduino sketch: `setup()` and `loop()`. The `setup()` is called once, when the sketch starts. It's a good place to do setup tasks like setting pin modes or initializing libraries. The `loop()` function is called over and over and is heart of most sketches. You need to include both functions in your sketch, even if you don't need them for anything.

### Exercises

**1. Change the code so that the LED is on for 100 milliseconds and off for 1000.**

**2. Change the code so that the LED turns on when the sketch starts and stays on.**

**See Also**

- setup()
- loop()
- pinMode()
- digitalWrite()
- delay()

# Pins

### Pins Configured as INPUT

Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode(). Pins configured as inputs are said to be in a high-impedance state. One way of explaining this is that input pins make extremely small demands on the circuit that they are sampling, say equivalent to a series resistor of 100 Megohms in front of the pin. This means that it takes very little current to move the input pin from one state to another, and can make the pins useful for such tasks as implementing a capacitive touch sensor.

This also means however that input pins with nothing connected to them, or with wires connected to them that are not connected to other circuits, will report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin for example.

### Pullup Resistors

Often it is useful to steer an input pin to a known state if no input is present. This can be done by adding a pullup resistor(to +5V), or pulldown resistor (resistor to ground) on the input, with 10K being a common value.

There are also convenient 20K pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed in the following manner.

```
pinMode(pin, INPUT);           // set pin to input
digitalWrite(pin, HIGH);       // turn on pullup resistors
```

Note that the pullup resistors provide enough current to dimly light an LED connected to a pin that has been configured as an input. If LED's in a project seem to be working, but very dimly, this is likely what is going on, and the programmer has forgotten to use pinMode() to set the pins to outputs.

Note also that the pullup resistors are controlled by the same registers (internal chip memory locations) that control whether a pin is HIGH or LOW. Consequently a pin that is configured to have pullup resistors turned on when the pin is an INPUT, will have the pin configured as HIGH if the pin is then swtiched to an OUTPUT with pinMode(). This works in the other direction as well, and an output pin that is left in a HIGH state will have the pullup resistors set if switched to an input with pinMode().

### Pins Configured as OUTPUT

Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), or run many sensors, for example, but not enough current to run most relays, solenoids, or motors.

Short circuits on Arduino pins, or attempting to run high current devices from them, can damage or destroy the output transistors in the pin, or damage the entire Atmega chip. Often this will result in a "dead" pin in the microcontroller but the remaining chip will still function adequately. For this reason it is a good idea to connect OUTPUT pins to other devices with 470O or 1k resistors, unless maximum current draw from the pins is required for a particular application.

Foundations

# Analog Pins

A description of the analog input pins on an Atmega168 (Arduino chip).

### A/D converter

The Atmega168 contains an onboard 6 channel analog-to-digital (A/D) converter. The converter has 10 bit resolution, returning integers from 0 to 1023. While the main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general purpose input/output (GPIO) pins (the same as digital pins 0 - 13).

Consequently, if a user needs more general purpose input output pins, and all the analog pins are not in use, the analog pins may be used for GPIO.

### Pin mapping

The Arduino pin numbers corresponding to the analog pins are 14 through 19. Note that these are Arduino pin numbers, and do not correspond to the physical pin numbers on the Atmega168 chip. The analog pins can be used identically to the digital pins, so for example, to set analog pin 0 to an output, and to set it HIGH, the code would look like this:

```
pinMode(14, OUTPUT);
digitalWrite(14, HIGH);
```

### Pullup resistors

The analog pins also have pullup resistors, which work identically to pullup resistors on the digital pins. They are enabled by issuing a command such as

```
digitalWrite(14, HIGH);  // set pullup on analog pin 0
```

while the pin is an *input*.

Be aware however that turning on a pullup will affect the value reported by analogRead() when using some sensors if done inadvertently. Most users will want to use the pullup resistors only when using an analog pin in its digital mode.

### Details and Caveats

The analogRead command will not work correctly if a pin has been previously set to an output, so if this is the case, set it back to an input before using analogRead. Similarly if the pin has been set to HIGH as an output.

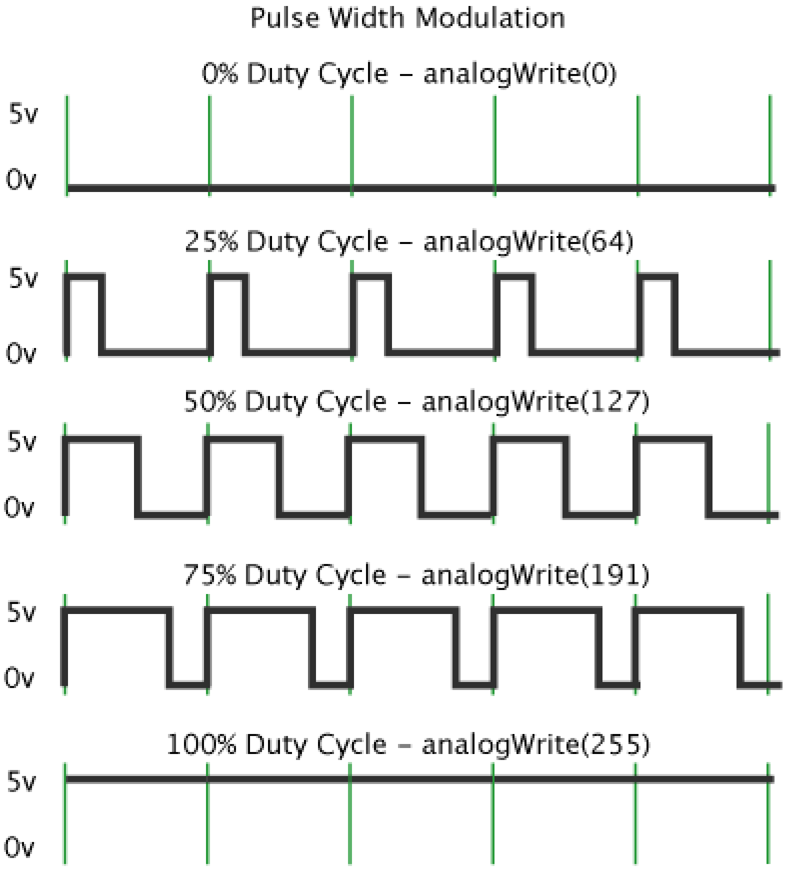
The Atmega168 datasheet also cautions against switching digital pins in close temporal proximity to making A/D readings (analogRead) on other analog pins. This can cause electrical noise and introduce jitter in the analog system. It may be desirable, after manipulating analog pins (in digital mode), to add a short delay before using analogRead() to read other analog pins.

## PWM

The Fading example demonstrates the use of analog output (PWM) to fade an LED. It is available in the File->Sketchbook->Examples->Analog menu of the Arduino software.

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
5v
0v
25% Duty Cycle – analogWrite(64)
5v
0v
50% Duty Cycle – analogWrite(127)
5v
0v
75% Duty Cycle – analogWrite(191)
5v
0v
100% Duty Cycle – analogWrite(255)
5v
0v

Once you get this example running, grab your arduino and shake it back and forth. What you are doing here is essentially mapping time across the space. To our eyes, the movement blurs each LED blink into a line. As the LED fades in and out, those little lines will grow and shrink in length. Now you are seeing the pulse width.

*Written by Timothy Hirzel*

Foundations

Edit Page | Page History | Printable View | All Recent Site Changes

## Memory

There are three pools of memory in the microcontroller used on Arduino boards (ATmega168):

- Flash memory (program space), is where the Arduino sketch is stored.
- SRAM (static random access memory) is where the sketch creates and manipulates variables when it runs.
- EEPROM is memory space that programmers can use to store long-term information.

Flash memory and EEPROM memory are non-volatile (the information persists after the power is turned off). SRAM is volatile and will be lost when the power is cycled.

The ATmega168 chip has the following amounts of memory:

```
Flash  16k bytes (of which 2k is used for the bootloader)
SRAM   1024 bytes
EEPROM 512 bytes
```

Notice that there's not much SRAM available. It's easy to use it all up by having lots of strings in your program. For example, a declaration like:

char message[] = "I support the Cape Wind project.";

puts 32 bytes into SRAM (each character takes a byte). This might not seem like a lot, but it doesn't take long to get to 1024, especially if you have a large amount of text to send to a display, or a large lookup table, for example.

If you run out of SRAM, your program may fail in unexpected ways; it will appear to upload successfully, but not run, or run strangely. To check if this is happening, you can try commenting out or shortening the strings or other data structures in your sketch (without changing the code). If it then runs successfully, you're probably running out of SRAM. There are a few things you can do to address this problem:

- If your sketch talks to a program running on a (desktop/laptop) computer, you can try shifting data or calculations to the computer, reducing the load on the Arduino.

- If you have lookup tables or other large arrays, use the smallest data type necessary to store the values you need; for example, an int takes up two bytes, while a byte uses only one (but can store a smaller range of values).

- If you don't need to modify the strings or data while your sketch is running, you can store them in flash (program) memory instead of SRAM; to do this, use the PROGMEM keyword.

To use the EEPROM, see the EEPROM library.

Foundations

# Bootloader

The bootloader is a small piece of software that we've burned onto the chips that come with your Arduino boards. It allows you to upload sketches to the board without external hardware.

When you reset the Arduino board, it runs the bootloader (if present). The bootloader pulses digital pin 13 (you can connect an LED to make sure that the bootloader is installed). The bootloader then listens for commands or data to arrive from the the computer. Usually, this is a sketch that the bootloader writes to the flash memory on the ATmega168 or ATmega8 chip. Then, the bootloader launches the newly-uploaded program. If, however, no data arrives from the computer, the bootloader launches whatever program was last uploaded onto the chip. If the chip is still "virgin" the bootloader is the only program in memory and will start itself again.

### Why are we using a bootloader?

The use of a bootloader allows us to avoid the use of external hardware programmers. (Burning the bootloader onto the chip, however, requires one of these external programmers.)

### Why doesn't my sketch start?

It's possible to "confuse" the bootloader so that it never starts your sketch. In particular, if you send serial data to the board just after it resets (when the bootloader is running), it may think you're talking to it and never quit. In particular, the auto-reset feature on the Diecimila means that the board resets (and the bootloader starts) whenever you open a serial connection to it. To avoid this problem, you should wait for two seconds or so after opening the connection before sending any data. On the NG, the board doesn't reset when you open a serial connection to it, but when it does reset it takes longer - about 8-10 seconds - to timeout.

### Looking for more information?

See the bootloader development page for information on burning a bootloader and other ways to configure a chip.

## Variables

A variable is a place to store a piece of data. It has a name, a value, and a type. For example, this statement (called a *declaration*):

```
int pin = 13;
```

creates a variable whose name is pin, whose value is 13, and whose type is int. Later on in the program, you can refer to this variable by its name, at which point its value will be looked up and used. For example, in this statement:

```
pinMode(pin, OUTPUT);
```

it is the value of pin (13) that will be passed to the pinMode() function. In this case, you don't actually need to use a variable, this statement would work just as well:

```
pinMode(13, OUTPUT);
```

The advantage of a variable in this case is that you only need to specify the actual number of the pin once, but you can use it lots of times. So if you later decide to change from pin 13 to pin 12, you only need to change one spot in the code. Also, you can use a descriptive name to make the significance of the variable clear (e.g. a program controlling an RGB LED might have variables called redPin, greenPin, and bluePin).

A variable has other advantages over a value like a number. Most importantly, you can change the value of a variable using an *assignment* (indicated by an equals sign). For example:

```
pin = 12;
```

will change the value of the variable to 12. Notice that we don't specify the type of the variable: it's not changed by the assignment. That is, the name of the variable is permanently associated with a type; only its value changes. [1] Note that you have to declare a variable before you can assign a value to it. If you include the preceding statement in a program without the first statement above, you'll get a message like: "error: pin was not declared in this scope".

When you assign one variable to another, you're making a copy of its value and storing that copy in the location in memory associated with the other variable. Changing one has no effect on the other. For example, after:

```
int pin = 13;
int pin2 = pin;
pin = 12;
```

only pin has the value 12; pin2 is still 13.

Now what, you might be wondering, did the word "scope" in that error message above mean? It refers to the part of your program in which the variable can be used. This is determined by where you declare it. For example, if you want to be able to use a variable anywhere in your program, you can declare at the top of your code. This is called a *global* variable; here's an example:

```
int pin = 13;

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

void loop()
{
  digitalWrite(pin, HIGH);
}
```

As you can see, pin is used in both the setup() and loop() functions. Both functions are referring to the same variable, so that changing it one will affect the value it has in the other, as in:

```
int pin = 13;

void setup()
{
  pin = 12;
  pinMode(pin, OUTPUT);
}

void loop()
{
  digitalWrite(pin, HIGH);
}
```

Here, the digitalWrite() function called from loop() will be passed a value of 12, since that's the value that was assigned to the variable in the setup() function.

If you only need to use a variable in a single function, you can declare it there, in which case its scope will be limited to that function. For example:

```
void setup()
{
  int pin = 13;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, HIGH);
}
```

In this case, the variable pin can only be used inside the setup() function. If you try to do something like this:

```
void loop()
{
  digitalWrite(pin, LOW); // wrong: pin is not in scope here.
}
```

you'll get the same message as before: "error: 'pin' was not declared in this scope". That is, even though you've declared pin somewhere in your program, you're trying to use it somewhere outside its scope.

Why, you might be wondering, wouldn't you make all your variables global? After all, if I don't know where I might need a variable, why should I limit its scope to just one function? The answer is that it can make it easier to figure out what happens to it. If a variable is global, its value could be changed anywhere in the code, meaning that you need to understand the whole program to know what will happen to the variable. For example, if your variable has a value you didn't expect, it can be much easier to figure out where the value came from if the variable has a limited scope.

[block scope] [size of variables]

[1] In some languages, like Python, types are associated with values, not variable names, and you can assign values of any type to a variable. This is referred to as *dynamic typing*.

# Tutorial.Foundations History

Hide minor edits - Show changes to markup

June 01, 2008, at 08:38 PM by David A. Mellis -
Added lines 5-8:

### Basics

- Sketch: The various components of a sketch and how they work.

Restore
April 29, 2008, at 10:33 AM by David A. Mellis -
Deleted lines 0-2:

Reference | Extended Reference

Deleted lines 2-3:
Restore
April 29, 2008, at 07:47 AM by Paul Badger -
Changed lines 31-35 from:

test

to:

Restore
April 29, 2008, at 07:39 AM by Paul Badger -
Changed lines 31-35 from:

test

to:

test

Restore
April 29, 2008, at 07:38 AM by Paul Badger -
Changed lines 31-35 from:

test

to:

test

Restore
April 29, 2008, at 07:34 AM by Paul Badger -
Changed lines 31-35 from:

test

to:

test

Restore
April 29, 2008, at 07:31 AM by Paul Badger -
Added lines 29-35:

test

Restore
April 29, 2008, at 07:28 AM by Paul Badger -
Changed lines 51-52 from:

GroupHead

to:
Restore
April 29, 2008, at 07:27 AM by Paul Badger -
Changed lines 51-52 from:

GroupHead

to:

GroupHead

Restore
April 29, 2008, at 07:26 AM by Paul Badger -
Added lines 51-52:

GroupHead

Restore
April 22, 2008, at 05:57 PM by Paul Badger -
Restore
April 10, 2008, at 09:44 AM by David A. Mellis - adding pwm tutorial
Added lines 16-17:

- PWM: How the analogWrite() function simulates an analog output using pulse-width modulation.

Added line 27:
Restore
April 06, 2008, at 05:33 PM by Paul Badger -
Changed lines 25-26 from:
to:

- Port Manipulation: Manipulating ports directly for faster manipulation of multiple pins

Restore
March 31, 2008, at 06:23 AM by Paul Badger -
Changed lines 1-4 from:

Reference | Reference| Extended Reference

to:

Reference | Extended Reference

Restore
March 31, 2008, at 06:22 AM by Paul Badger -
Added lines 1-4:

Reference | Reference| Extended Reference

Changed lines 8-9 from:

Reference

to:
Restore
March 31, 2008, at 06:20 AM by Paul Badger -
Added lines 3-5:

Reference

Restore
March 21, 2008, at 06:27 PM by Paul Badger -
Changed lines 13-14 from:

### Arduino Software

to:

### Arduino Firmware

Restore
March 09, 2008, at 11:15 PM by Paul Badger -
Changed lines 5-6 from:

### Chip-Level Documentation and Techniques

to:

### Microcontrollers

Changed lines 17-18 from:

### Programming Techniques

to:

### Programming Technique

Restore
March 09, 2008, at 11:12 PM by Paul Badger -
Changed lines 5-6 from:

### Chip Level Documentation and Techniques

to:

### Chip-Level Documentation and Techniques

Restore
March 09, 2008, at 11:11 PM by Paul Badger -
Changed lines 5-6 from:

### Chip Hardware

to:

### Chip Level Documentation and Techniques

Restore
March 09, 2008, at 11:09 PM by Paul Badger -
Changed lines 5-6 from:

### Hardware

to:

### Chip Hardware

Restore
March 09, 2008, at 10:49 PM by Paul Badger -
Changed lines 13-14 from:

Arduino Software

to:

### Arduino Software

Restore
March 09, 2008, at 10:49 PM by Paul Badger -
Added lines 13-14:

Arduino Software

Restore
March 09, 2008, at 10:48 PM by Paul Badger -
Deleted lines 6-7:

- Memory: The various types of memory available on the Arduino board.

Added lines 11-12:

- Memory: The various types of memory available on the Arduino board.

Changed lines 15-16 from:

### Programming

to:

### Programming Techniques

Restore
March 09, 2008, at 10:47 PM by Paul Badger -
Added lines 5-6:

### Hardware

Added lines 15-16:

### Programming

Restore
March 09, 2008, at 07:19 PM by David A. Mellis -
Changed lines 3-4 from:

This page contains chip-level reference material and low-level hardare and software techniques used in the Arduino language. Page Discussion

to:

This page contains explanations of some of the elements of the Arduino hardware and software and the concepts behind them. Page Discussion

Added lines 11-12:

- Bootloader: A small program pre-loaded on the Arduino board to allow uploading sketches.

Restore
March 09, 2008, at 07:16 PM by David A. Mellis -
Changed lines 11-12 from:
to:

- Variables: How to define and use variables.

Deleted lines 25-26:

- Variables: How to define and use variables.

Restore
March 07, 2008, at 09:46 PM by Paul Badger -
Changed lines 3-4 from:

This page contains general chip-level reference material as it relates to basic low-level hardare and software techniques used in the Arduino language. Page Discussion

to:

This page contains chip-level reference material and low-level hardare and software techniques used in the Arduino language. Page Discussion

Restore
March 07, 2008, at 09:25 PM by Paul Badger -
Changed lines 11-12 from:

- **Foundations**

to:
Restore
March 07, 2008, at 09:24 PM by Paul Badger -
Changed lines 3-37 from:

This page contains general chip-level reference material as it relates to basic low-level hardare and software techniques used in the Arduino language. Page Discussion

to:

This page contains general chip-level reference material as it relates to basic low-level hardare and software techniques used in the Arduino language. Page Discussion

- Memory: The various types of memory available on the Arduino board.
- Digital Pins: How the pins work and what it means for them to be configured as inputs or outputs.
- Analog Input Pins: Details about the analog-to-digital conversion and other uses of the pins.
- **Foundations**

(:if false:)

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.
- Communication?: An overview of the various ways in which an Arduino board can communicate with other devices (serial, I2C, SPI, Midi, etc.)
- Serial Communication?: How to send serial data from an Arduino board to a computer or other device (including via the USB connection).
- Interrupts?: Code that interrupts other code under certain conditions.
- Numbers?: The various types of numbers available and how to use them.
- Variables: How to define and use variables.
- Arrays?: How to store multiple values of the same type.
- Pointers?:
- Functions?: How to write and call functions.
- Optimization?: What to do when your program runs too slowly.
- Debugging?: Figuring out what's wrong with your hardware or software and how to fix it.

(:ifend:)

Restore
March 07, 2008, at 09:21 PM by Paul Badger -
Changed line 3 from:

This page contains general chip-level reference material as it relates to basic low-level techniques. Page Discussion

to:

This page contains general chip-level reference material as it relates to basic low-level hardare and software techniques used in the Arduino language. Page Discussion

Restore
March 07, 2008, at 09:12 PM by Paul Badger -
Added lines 1-3:

## Foundations

This page contains general chip-level reference material as it relates to basic low-level techniques. Page Discussion

Restore

**Learning** Examples | Foundations | Hacking | Links

# Foundations

This page contains explanations of some of the elements of the Arduino hardware and software and the concepts behind them. Page Discussion

## Basics

- Sketch: The various components of a sketch and how they work.

## Microcontrollers

- Digital Pins: How the pins work and what it means for them to be configured as inputs or outputs.
- Analog Input Pins: Details about the analog-to-digital conversion and other uses of the pins.
- PWM: How the analogWrite() function simulates an analog output using pulse-width modulation.
- Memory: The various types of memory available on the Arduino board.

## Arduino Firmware

- Bootloader: A small program pre-loaded on the Arduino board to allow uploading sketches.

## Programming Technique

- Variables: How to define and use variables.
- Port Manipulation: Manipulating ports directly for faster manipulation of multiple pins

# Tutorial.Links History

Hide minor edits - Show changes to markup

June 08, 2008, at 11:30 AM by David A. Mellis -
Added lines 62-65:

Wiring electronics reference: circuit diagrams for connecting a variety of basic electronic components.

Schematics to circuits: from Wiring, a guide to transforming circuit diagrams into physical circuits.

Restore
June 08, 2008, at 11:29 AM by David A. Mellis -
Changed lines 18-27 from:

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

to:
Added lines 41-49:

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

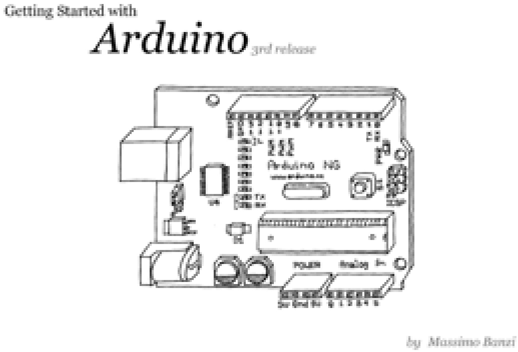
- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Restore
June 08, 2008, at 11:28 AM by David A. Mellis -
Added lines 8-28:

### Books and Manuals

Make:
Making Things Talk
Practical Methods for Connecting Physical Objects
PROJECTS AND IDEAS TO CREATE TALKING OBJECTS FROM ANYTHING
O'REILLY
Tom Igoe

Making Things Talk (by Tom Igoe): teaches you how to get your creations to communicate with one another by forming networks of smart devices that carry on conversations with you and your environment.



Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

(:cell width=50%:)

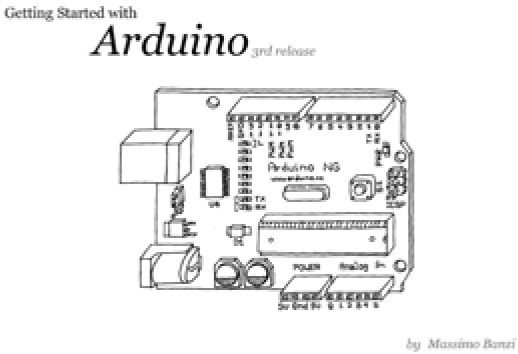
Deleted lines 66-86:

(:cell width=50%:)

### Books and Manuals

Make
Making Things Talk
Practical Methods for Connecting Physical Objects
PROJECTS AND IDEAS TO CREATE TALKING OBJECTS FROM ANYTHING
O'REILLY
Tom Igoe

Making Things Talk (by Tom Igoe): teaches you how to get your creations to communicate with one another by forming networks of smart devices that carry on conversations with you and your environment.



Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

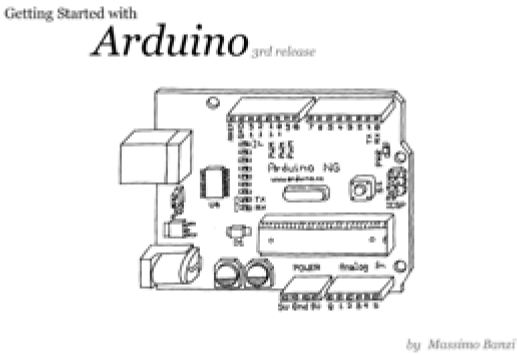
Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

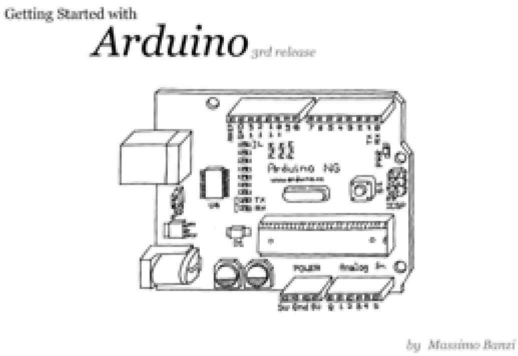
Restore
May 06, 2008, at 01:23 PM by David A. Mellis -
Added lines 29-30:

Tom Igoe's Physical Computing Site: lots of information on electronics, microcontrollers, sensors, actuators, books, etc.

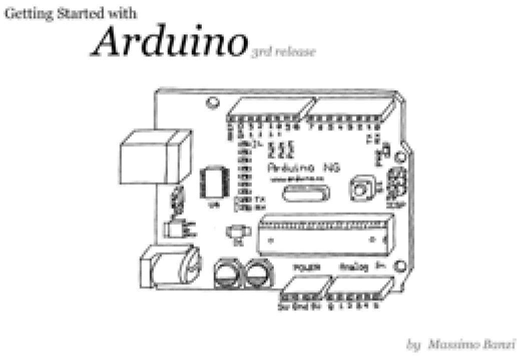
Restore
May 06, 2008, at 01:21 PM by David A. Mellis -
Changed lines 51-52 from:



to:



Restore
May 06, 2008, at 01:20 PM by David A. Mellis -
Added lines 51-52:



Restore
May 06, 2008, at 01:14 PM by David A. Mellis -
Changed lines 47-48 from:

Making
Things Talk
Practical Methods for Connecting Physical Objects
PROJECTS AND IDEAS TO CREATE TALKING OBJECTS FROM ANYTHING
O'REILLY
Tom Igoe

Making Things Talk (by Tom Igoe): teaches you how to get your creations to communicate

with one another by forming networks of smart devices that carry on conversations with you and your environment.
to:

Making Things Talk (by Tom Igoe): teaches you how to get your creations to communicate with one another by forming networks of smart devices that carry on conversations with you and your environment.

Restore
May 06, 2008, at 01:13 PM by David A. Mellis -
Added lines 27-43:

### Other Examples and Tutorials

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Examples from Tom Igoe

Examples from Jeff Gray

Changed lines 45-47 from:

### Manuals, Curricula, and Other Resources

to:

### Books and Manuals

Making Things Talk (by Tom Igoe): teaches you how to get your creations to communicate with one another by forming networks of smart devices that carry on conversations with you and your environment.
Changed lines 60-73 from:

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Examples from Tom Igoe

Examples from Jeff Gray

to:
Restore
April 29, 2008, at 06:44 PM by David A. Mellis -
Added lines 5-7:

(:table width=100% border=0 cellpadding=5 cellspacing=0:) (:cell width=50%:)

Added lines 27-28:

(:cell width=50%:)

Changed lines 54-56 from:

Examples from Jeff Gray

to:

Examples from Jeff Gray

(:tableend:)

Restore
April 29, 2008, at 06:43 PM by David A. Mellis -
Added lines 1-49:

## Links

Arduino examples, tutorials, and documentation elsewhere on the web.

### Community Documentation

Tutorials created by the Arduino community. Hosted on the publicly-editable playground wiki.

Board Setup and Configuration: Information about the components and usage of Arduino hardware.

Interfacing With Hardware: Code, circuits, and instructions for using various electronic components with an Arduino board.

- Output
- Input
- Interaction
- Storage
- Communication

Interfacing with Software: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

Code Library and Tutorials: Arduino functions for performing specific tasks and other programming tutorials.

Electronics Techniques: tutorials on soldering and other electronics resources.

### Manuals, Curricula, and Other Resources

Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light

- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Examples from Tom Igoe

Examples from Jeff Gray

Restore

# Arduino : Tutorial / Links

**Learning** Examples | Foundations | Hacking | Links

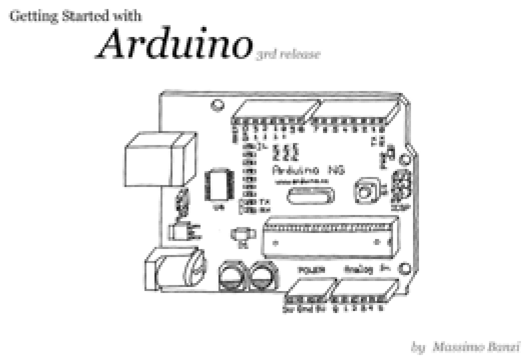
# Links

Arduino examples, tutorials, and documentation elsewhere on the web.

## Books and Manuals

Making Things Talk
Practical Methods for Connecting Physical Objects
PROJECTS AND IDEAS TO CREATE TALKING OBJECTS FROM ANYTHING
O'REILLY
Tom Igoe

Making Things Talk (by Tom Igoe): teaches you how to get your creations to communicate with one another by forming networks of smart devices that carry on conversations with you and your environment.



Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

## Community Documentation

Tutorials created by the Arduino community. Hosted on the publicly-editable playground wiki.

Board Setup and Configuration: Information about the components and usage of Arduino hardware.

Interfacing With Hardware: Code, circuits, and instructions for using various electronic components with an Arduino board.

- Output
- Input
- Interaction
- Storage
- Communication

Interfacing with Software: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

Code Library and Tutorials: Arduino functions for performing specific tasks and other programming tutorials.

Electronics Techniques: tutorials on soldering and other electronics resources.

## Other Examples and Tutorials

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Tom Igoe's Physical Computing Site: lots of information on electronics, microcontrollers, sensors, actuators, books, etc.

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Wiring electronics reference: circuit diagrams for connecting a variety of basic electronic components.

Schematics to circuits: from Wiring, a guide to transforming circuit diagrams into physical circuits.

Examples from Tom Igoe

Examples from Jeff Gray

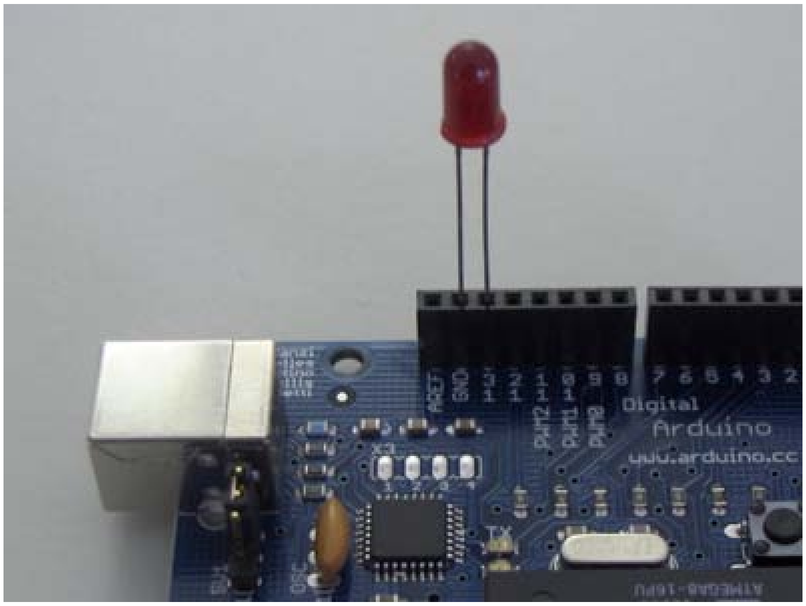
## The "Hello World!" of Physical Computing

The first program every programmer learns consists in writing enough code to make their code show the sentence "Hello World!" on a screen.

As a microcontroller, Arduino doesn't have any pre-established output devices. Willing to provide newcomers with some help while debugging programs, we propose the use of one of the board's pins plugging a LED that we will make blink indicating the right functionallity of the program.

We have added a 1K resistor to pin 13, what allows the immediate connection of a LED between that pin and ground.

LEDs have polarity, which means they will only light up if you orient the legs properly. The long leg is typically positive, and should connect to pin 13. The short leg connects to GND; the bulb of the LED will also typically have a flat edge on this side. If the LED doesn't light up, trying reversing the legs (you won't hurt the LED if you plug it in backwards for a short period of time).



## Code

The example code is very simple, credits are to be found in the comments.

```
/* Blinking LED
 * ------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, in intervals of 2 seconds. Ideally we use pin 13 on the Arduino
 * board because it has a resistor attached to it, needing only an LED

 *
 * Created 1 June 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 * based on an orginal by H. Barragan for the Wiring i/o board
 */

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

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

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

```
/*
* Code for cross-fading 3 LEDs, red, green and blue, or one tri-color LED, using PWM
* The program cross-fades slowly from red to green, green to blue, and blue to red
* The debugging code assumes Arduino 0004, as it uses the new Serial.begin()-style functions
* Clay Shirky <clay.shirky@nyu.edu>
*/

// Output
int redPin   = 9;   // Red LED,   connected to digital pin 9
int greenPin = 10;  // Green LED, connected to digital pin 10
int bluePin  = 11;  // Blue LED,  connected to digital pin 11

// Program variables
int redVal   = 255; // Variables to store the values to send to the pins
int greenVal = 1;   // Initial values are Red full, Green and Blue off
int blueVal  = 1;

int i = 0;     // Loop counter
int wait = 50; // 50ms (.05 second) delay; shorten for faster fades
int DEBUG = 0; // DEBUG counter; if set to 1, will write values back via serial

void setup()
{
  pinMode(redPin,   OUTPUT);   // sets the pins as output
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin,  OUTPUT);
  if (DEBUG) {           // If we want to see the pin values for debugging...
    Serial.begin(9600);  // ...set up the serial ouput on 0004 style
  }
}

// Main program
void loop()
{
  i += 1;      // Increment counter
  if (i < 255) // First phase of fades
  {
    redVal   -= 1; // Red down
    greenVal += 1; // Green up
    blueVal   = 1; // Blue low
  }
  else if (i < 509) // Second phase of fades
  {
    redVal    = 1; // Red low
    greenVal -= 1; // Green down
    blueVal  += 1; // Blue up
  }
  else if (i < 763) // Third phase of fades
  {
    redVal  += 1; // Red up
    greenVal = 1; // Green low
    blueVal -= 1; // Blue down
  }
  else // Re-set the counter, and start the fades again
  {
    i = 1;
  }

  analogWrite(redPin,   redVal);   // Write current values to LED pins
  analogWrite(greenPin, greenVal);
  analogWrite(bluePin,  blueVal);

  if (DEBUG) { // If we want to read the output
    DEBUG += 1;     // Increment the DEBUG counter
    if (DEBUG > 10) // Print every 10 loops
    {
      DEBUG = 1;     // Reset the counter

      Serial.print(i);       // Serial commands in 0004 style
      Serial.print("\t");    // Print a tab
      Serial.print("R:");    // Indicate that output is red value
      Serial.print(redVal);  // Print red value
      Serial.print("\t");    // Print a tab
      Serial.print("G:");    // Repeat for green and blue...
      Serial.print(greenVal);
      Serial.print("\t");
      Serial.print("B:");
      Serial.println(blueVal); // println, to end with a carriage return
    }
  }
  delay(wait); // Pause for 'wait' milliseconds before resuming the loop
}
```

```
/*
* Code for cross-fading 3 LEDs, red, green and blue (RGB)
* To create fades, you need to do two things:
*  1. Describe the colors you want to be displayed
*  2. List the order you want them to fade in
*
* DESCRIBING A COLOR:
* A color is just an array of three percentages, 0-100,
*  controlling the red, green and blue LEDs
*
* Red is the red LED at full, blue and green off
*   int red = { 100, 0, 0 }
* Dim white is all three LEDs at 30%
*   int dimWhite = {30, 30, 30}
* etc.
*
* Some common colors are provided below, or make your own
*
* LISTING THE ORDER:
* In the main part of the program, you need to list the order
*  you want to colors to appear in, e.g.
*  crossFade(red);
*  crossFade(green);
*  crossFade(blue);
*
* Those colors will appear in that order, fading out of
*    one color and into the next
*
* In addition, there are 5 optional settings you can adjust:
* 1. The initial color is set to black (so the first color fades in), but
*    you can set the initial color to be any other color
* 2. The internal loop runs for 1020 interations; the 'wait' variable
*    sets the approximate duration of a single crossfade. In theory,
*    a 'wait' of 10 ms should make a crossFade of ~10 seconds. In
*    practice, the other functions the code is performing slow this
*    down to ~11 seconds on my board. YMMV.
* 3. If 'repeat' is set to 0, the program will loop indefinitely.
*    if it is set to a number, it will loop that number of times,
*    then stop on the last color in the sequence. (Set 'return' to 1,
*    and make the last color black if you want it to fade out at the end.)
* 4. There is an optional 'hold' variable, which pasues the
*    program for 'hold' milliseconds when a color is complete,
*    but before the next color starts.
* 5. Set the DEBUG flag to 1 if you want debugging output to be
*    sent to the serial monitor.
*
*    The internals of the program aren't complicated, but they
*    are a little fussy -- the inner workings are explained
*    below the main loop.
*
* April 2007, Clay Shirky <clay.shirky@nyu.edu>
*/
```

```
// Output
int redPin = 9;   // Red LED,   connected to digital pin 9
int grnPin = 10;  // Green LED, connected to digital pin 10
int bluPin = 11;  // Blue LED,  connected to digital pin 11

// Color arrays
int black[3]  = { 0, 0, 0 };
int white[3]  = { 100, 100, 100 };
int red[3]    = { 100, 0, 0 };
int green[3]  = { 0, 100, 0 };
int blue[3]   = { 0, 0, 100 };
int yellow[3] = { 40, 95, 0 };
int dimWhite[3] = { 30, 30, 30 };
// etc.

// Set initial color
int redVal = black[0];
int grnVal = black[1];
int bluVal = black[2];

int wait = 10;      // 10ms internal crossFade delay; increase for slower fades
int hold = 0;       // Optional hold when a color is complete, before the next crossFade
int DEBUG = 1;      // DEBUG counter; if set to 1, will write values back via serial
int loopCount = 60; // How often should DEBUG report?
int repeat = 3;     // How many times should we loop before stopping? (0 for no stop)
int j = 0;          // Loop counter for repeat

// Initialize color variables
int prevR = redVal;
int prevG = grnVal;
int prevB = bluVal;

// Set up the LED outputs
void setup()
{
  pinMode(redPin, OUTPUT);   // sets the pins as output
  pinMode(grnPin, OUTPUT);
  pinMode(bluPin, OUTPUT);

  if (DEBUG) {           // If we want to see values for debugging...
    Serial.begin(9600);  // ...set up the serial ouput
  }
}

// Main program: list the order of crossfades
void loop()
{
  crossFade(red);
  crossFade(green);
  crossFade(blue);
  crossFade(yellow);

  if (repeat) { // Do we loop a finite number of times?
    j += 1;
    if (j >= repeat) { // Are we there yet?
      exit(j);         // If so, stop.
    }
  }
}

/* BELOW THIS LINE IS THE MATH -- YOU SHOULDN'T NEED TO CHANGE THIS FOR THE BASICS
*
* The program works like this:
* Imagine a crossfade that moves the red LED from 0-10,
*   the green from 0-5, and the blue from 10 to 7, in
*   ten steps.
*   We'd want to count the 10 steps and increase or
*   decrease color values in evenly stepped increments.
*   Imagine a + indicates raising a value by 1, and a -
*   equals lowering it. Our 10 step fade would look like:
*
*   1 2 3 4 5 6 7 8 9 10
* R + + + + + + + + + +
* G   +   +   +   +   +
* B     -     -     -
*
* The red rises from 0 to 10 in ten steps, the green from
* 0-5 in 5 steps, and the blue falls from 10 to 7 in three steps.
*
* In the real program, the color percentages are converted to
* 0-255 values, and there are 1020 steps (255*4).
*
* To figure out how big a step there should be between one up- or
* down-tick of one of the LED values, we call calculateStep(),
* which calculates the absolute gap between the start and end values,
* and then divides that gap by 1020 to determine the size of the step
* between adjustments in the value.
*/

int calculateStep(int prevValue, int endValue) {
  int step = endValue - prevValue; // What's the overall gap?
  if (step) {                      // If its non-zero,
    step = 1020/step;              //   divide by 1020
  }
  return step;
}

/* The next function is calculateVal. When the loop value, i,
*  reaches the step size appropriate for one of the
*  colors, it increases or decreases the value of that color by 1.
*  (R, G, and B are each calculated separately.)
*/

int calculateVal(int step, int val, int i) {

  if ((step) && i % step == 0) { // If step is non-zero and its time to change a value,
    if (step > 0) {              //   increment the value if step is positive...
      val += 1;
    }
    else if (step < 0) {         //   ...or decrement it if step is negative
      val -= 1;
    }
  }
  // Defensive driving: make sure val stays in the range 0-255
  if (val > 255) {
    val = 255;
  }
  else if (val < 0) {
    val = 0;
  }
  return val;
}

/* crossFade() converts the percentage colors to a
*  0-255 range, then loops 1020 times, checking to see if
*  the value needs to be updated each time, then writing
*  the color values to the correct pins.
*/
```

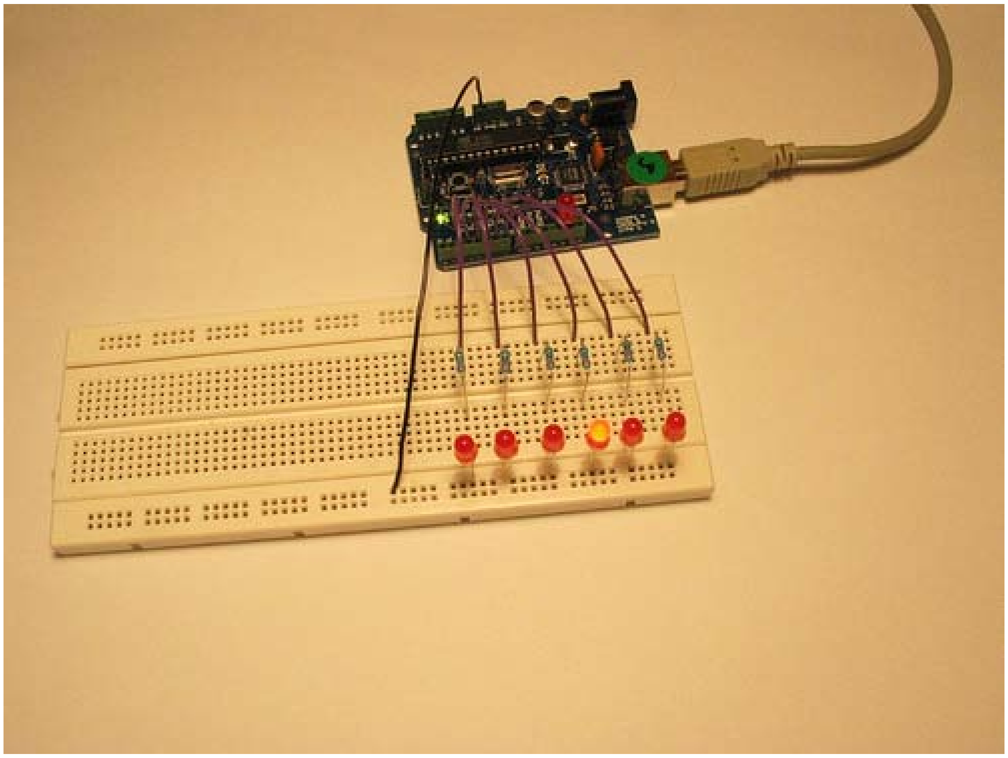
```
void crossFade(int color[3]) {
  // Convert to 0-255
  int R = (color[0] * 255) / 100;
  int G = (color[1] * 255) / 100;
  int B = (color[2] * 255) / 100;

  int stepR = calculateStep(prevR, R);
  int stepG = calculateStep(prevG, G);
  int stepB = calculateStep(prevB, B);

  for (int i = 0; i <= 1020; i++) {
    redVal = calculateVal(stepR, redVal, i);
    grnVal = calculateVal(stepG, grnVal, i);
    bluVal = calculateVal(stepB, bluVal, i);

    analogWrite(redPin, redVal);   // Write current values to LED pins
    analogWrite(grnPin, grnVal);
    analogWrite(bluPin, bluVal);

    delay(wait); // Pause for 'wait' milliseconds before resuming the loop

    if (DEBUG) { // If we want serial output, print it at the
      if (i == 0 or i % loopCount == 0) { // beginning, and every loopCount times
        Serial.print("Loop/RGB: #");
        Serial.print(i);
        Serial.print(" | ");
        Serial.print(redVal);
        Serial.print(" / ");
        Serial.print(grnVal);
        Serial.print(" / ");
        Serial.println(bluVal);
      }
      DEBUG += 1;
    }
  }
  // Update current values for next loop
  prevR = redVal;
  prevG = grnVal;
  prevB = bluVal;
  delay(hold); // Pause for optional 'wait' milliseconds before resuming the loop
}
```

## Knight Rider

We have named this example in memory to a TV-series from the 80's where the famous David Hasselhoff had an AI machine driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects.

Thus we decided that in order to learn more about sequential programming and good programming techniques for the I/O board, it would be interesting to use the **Knight Rider** as a metaphor.

This example makes use of 6 LEDs connected to the pins 2 - 7 on the board using 220 Ohm resistors. The first code example will make the LEDs blink in a sequence, one by one using only **digitalWrite(pinNum,HIGH/LOW)** and **delay(time)**. The second example shows how to use a **for(;;)** construction to perform the very same thing, but in fewer lines. The third and last example concentrates in the visual effect of turning the LEDs on/off in a more softer way.



*Example for Hasselhoff's fans*

### Knight Rider 1

```
/* Knight Rider 1
 * --------------
 *
 * Basically an extension of Blink_LED.
 *
 *
 * (cleft) 2005 K3, Malmo University
 * @author: David Cuartielles
 * @hardware: David Cuartielles, Aaron Hallborg
 */
```

```
int pin2 = 2;
int pin3 = 3;
int pin4 = 4;
int pin5 = 5;
int pin6 = 6;
int pin7 = 7;
int timer = 100;

void setup(){
  pinMode(pin2, OUTPUT);
  pinMode(pin3, OUTPUT);
  pinMode(pin4, OUTPUT);
  pinMode(pin5, OUTPUT);
  pinMode(pin6, OUTPUT);
  pinMode(pin7, OUTPUT);
}

void loop() {
   digitalWrite(pin2, HIGH);
   delay(timer);
   digitalWrite(pin2, LOW);
   delay(timer);

   digitalWrite(pin3, HIGH);
   delay(timer);
   digitalWrite(pin3, LOW);
   delay(timer);

   digitalWrite(pin4, HIGH);
   delay(timer);
   digitalWrite(pin4, LOW);
   delay(timer);

   digitalWrite(pin5, HIGH);
   delay(timer);
   digitalWrite(pin5, LOW);
   delay(timer);

   digitalWrite(pin6, HIGH);
   delay(timer);
   digitalWrite(pin6, LOW);
   delay(timer);

   digitalWrite(pin7, HIGH);
   delay(timer);
   digitalWrite(pin7, LOW);
   delay(timer);

   digitalWrite(pin6, HIGH);
   delay(timer);
   digitalWrite(pin6, LOW);
   delay(timer);

   digitalWrite(pin5, HIGH);
   delay(timer);
   digitalWrite(pin5, LOW);
   delay(timer);

   digitalWrite(pin4, HIGH);
   delay(timer);
   digitalWrite(pin4, LOW);
   delay(timer);

   digitalWrite(pin3, HIGH);
   delay(timer);
   digitalWrite(pin3, LOW);
   delay(timer);
}
```

### Knight Rider 2

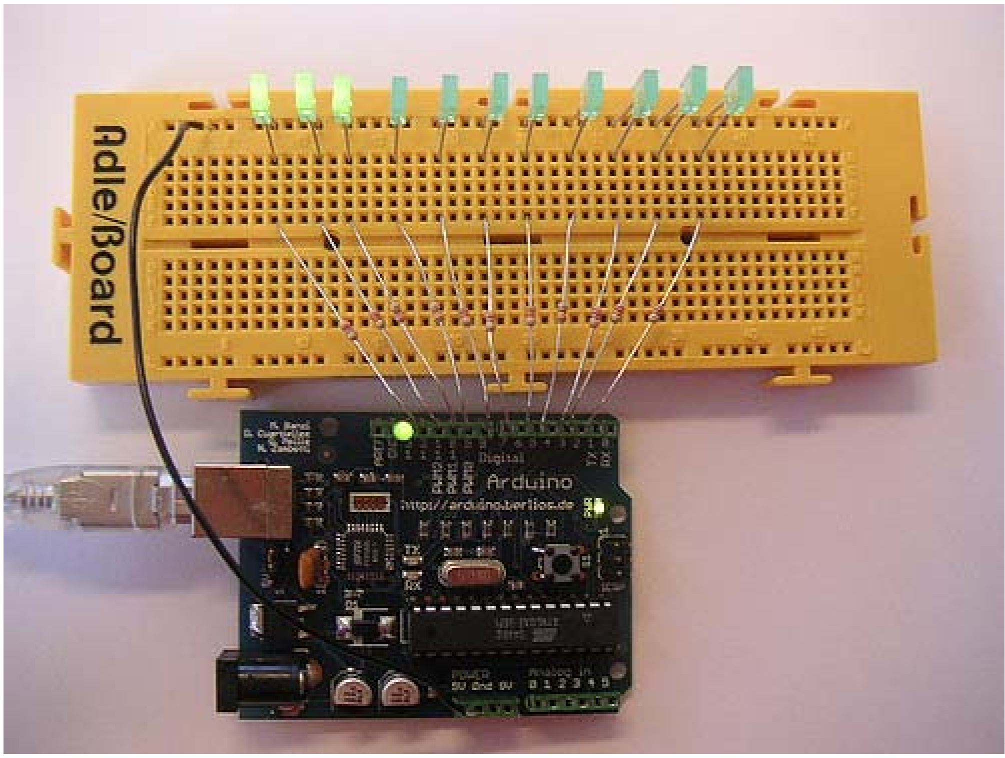
```
/* Knight Rider 2
 * --------------
 *
 * Reducing the amount of code using for(;;).
 *
 *
 * (cleft) 2005 K3, Malmo University
 * @author: David Cuartielles
 * @hardware: David Cuartielles, Aaron Hallborg
 */

int pinArray[] = {2, 3, 4, 5, 6, 7};
int count = 0;
int timer = 100;

void setup(){
  // we make all the declarations at once
  for (count=0;count<6;count++) {
    pinMode(pinArray[count], OUTPUT);
  }
}

void loop() {
  for (count=0;count<6;count++) {
   digitalWrite(pinArray[count], HIGH);
   delay(timer);
   digitalWrite(pinArray[count], LOW);
   delay(timer);
  }
  for (count=5;count>=0;count--) {
   digitalWrite(pinArray[count], HIGH);
   delay(timer);
   digitalWrite(pinArray[count], LOW);
   delay(timer);
  }
}
```

### Knight Rider 3

```
/* Knight Rider 3
 * --------------
 *
 * This example concentrates on making the visuals fluid.
 *
 *
 * (cleft) 2005 K3, Malmo University
 * @author: David Cuartielles
 * @hardware: David Cuartielles, Aaron Hallborg
 */

int pinArray[] = {2, 3, 4, 5, 6, 7};
int count = 0;
int timer = 30;

void setup(){
  for (count=0;count<6;count++) {
    pinMode(pinArray[count], OUTPUT);
  }
}

void loop() {
  for (count=0;count<5;count++) {
   digitalWrite(pinArray[count], HIGH);
   delay(timer);
   digitalWrite(pinArray[count + 1], HIGH);
   delay(timer);
   digitalWrite(pinArray[count], LOW);
   delay(timer*2);
  }
  for (count=5;count>0;count--) {
   digitalWrite(pinArray[count], HIGH);
   delay(timer);
   digitalWrite(pinArray[count - 1], HIGH);
   delay(timer);
   digitalWrite(pinArray[count], LOW);
   delay(timer*2);
  }
}
```

## Shooting Star

This example shows how to make a ray of light, or more poetically a Shooting Star, go through a line of LEDs. You will be able to controle how fast the 'star' shoots, and how long its 'tail' is. It isn't very elegant because the tail is as bright as the star, and in the end it seems like a solid ray that crosses the LED line.


?v=0

### How this works

You connect 11 LEDs to 11 arduino digital pins, through a 220 Ohm resistance (see image above).

The program starts lighting up LEDs until it reaches the number of LEDs equal to the size you have stablished for the tail. Then it will continue lighting LEDs towards the left (if you mount it like and look at it like the image) to make the star keep movint, and will start erasing from the right, to make sure we see the tail (otherwise we would just light up the whole line of leds, this will happen also if the tail size is equal or bigger than 11)

The tail size should be relatively small in comparison with the number of LEDs in order to see it. Of course you can increase the number of LEDs using an LED driver (see tutorial) and therefore, allow longer tails.

### Code

```
/* ShootingStar
 * ------------
 * This program is kind of a variation of the KnightRider
 * It plays in a loop a "Shooting Star" that is displayed
 * on a line of 11 LEDs directly connected to Arduino
 *
 * You can control how fast the star shoots thanx to the
 * variable called "waitNextLed"
```

```
  *
  * You can also control the length of the star's "tail"
  * through the variable "tail length"
  * First it reads two analog pins that are connected
  * to a joystick made of two potentiometers
  *
  * @author: Cristina Hoffmann
  * @hardware: Cristina Hofmann
  *
  */
  // Variable declaration

  int pinArray [] = { 2,3,4,5,6,7,8,9,10,11,12 };    // Array where I declare the pins connected to the
LEDs
  int controlLed = 13;
  int waitNextLed = 100;  // Time before I light up the next LED
  int tailLength = 4;     // Number of LEDs that stay lit befor I start turning   them off, thus the tail
  int lineSize = 11;       // Number of LEDs connected (which also is the size of the pinArray)

  // I asign the sens of the different Pins
  void setup()
  {
    int i;
    pinMode (controlLed, OUTPUT);
    for (i=0; i< lineSize; i++)
    {
      pinMode(pinArray[i], OUTPUT);
    }
  }

  // Main loop
  void loop()
  {
    int i;
    int tailCounter = tailLength;   // I set up the tail length in a counter
    digitalWrite(controlLed, HIGH); // I make sure I enter the loop indicating it with this LED

     for (i=0; i<lineSize; i++)
   {
     digitalWrite(pinArray[i],HIGH); // I light up consecutively the LEDs
     delay(waitNextLed);            // This time variable controles how fast I light them up
     if (tailCounter == 0)
      {
        digitalWrite(pinArray[i-tailLength],LOW); // I turn off the LEDs depending on my tailLength
      }
     else
       if (tailCounter > 0)
         tailCounter--;
   }

    for (i=(lineSize-tailLength); i<lineSize; i++)
   {
     digitalWrite(pinArray[i],LOW); // I turn off the LEDs
     delay(waitNextLed);            // This time variable controles how fast I light them upm, and turn off
as well
   }

}
```

@idea: Cristina Hoffmann

@code: Cristina Hoffmann

@pictures and graphics: Cristina Hoffmann

@date: 20060203 - Rennes - France

# Arduino

search

Buy | Download | Getting Started | Learning | Reference | Hardware | FAQ   Blog » | Forum » | Playground »

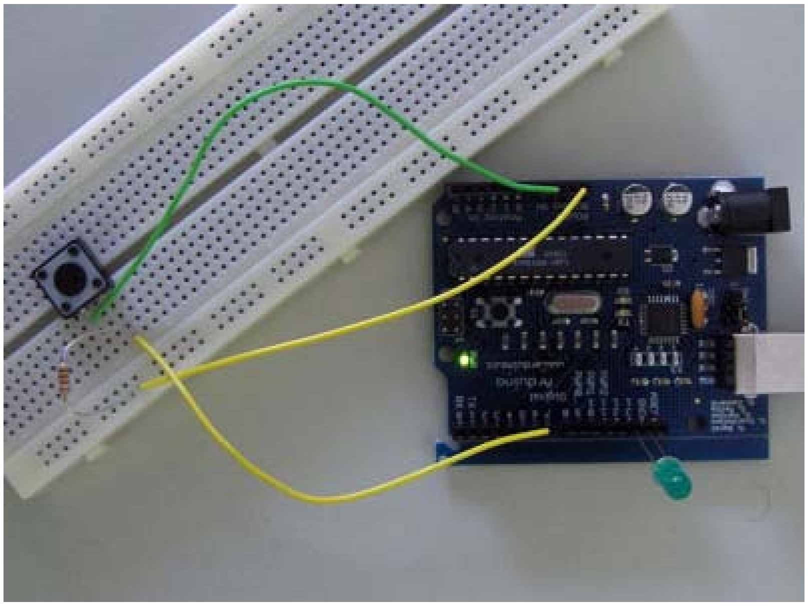
**Learning**   Examples | Foundations | Hacking | Links

## Pushbutton

The pushbutton is a component that connects two points in a circuit when you press it. The example turns on an LED when you press the button.

We connect three wires to the Arduino board. The first goes from one leg of the pushbutton through a pull-up resistor (here 2.2 KOhms) to the 5 volt supply. The second goes from the corresponding leg of the pushbutton to ground. The third connects to a digital i/o pin (here pin 7) which reads the button's state.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to 5 volts (through the pull-up resistor) and we read a HIGH. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to ground, so that we read a LOW. (The pin is still connected to 5 volts, but the resistor in-between them means that the pin is "closer" to ground.)



```
/* Basic Digital Read
 * ------------------
 *
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13, when pressing a pushbutton attached to pin 7. It illustrates the
 * concept of Active-Low, which consists in connecting buttons using a
 * 1K to 10K pull-up resistor.
 *
 * Created 1 December 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 */

int ledPin = 13; // choose the pin for the LED
int inPin = 7;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status
```

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

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

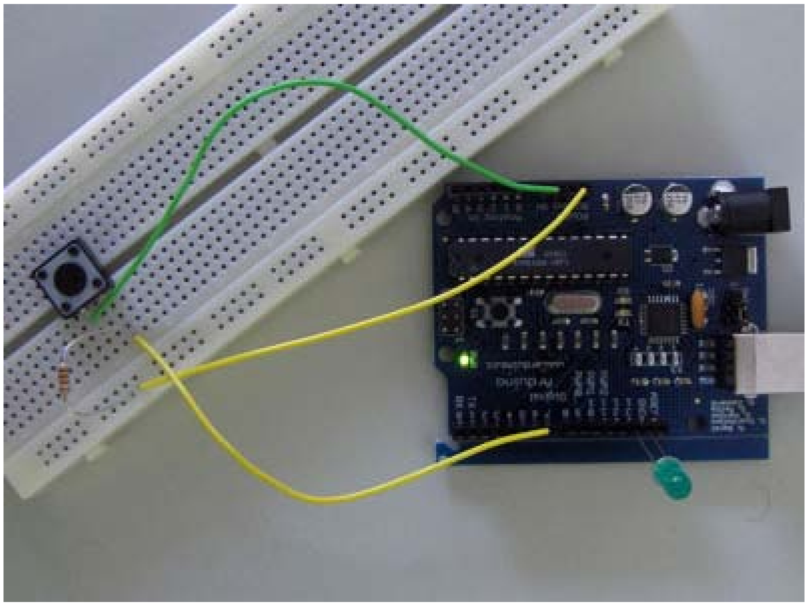
*Examples > Digital I/O*

## Switch

This example demonstrates the use of a pushbutton as a switch: each time you press the button, the LED (or whatever) is turned on (if it's off) or off (if on). It also debounces the input, without which pressing the button once would appear to the code as multiple presses.

### Circuit

A push-button on pin 2 and an LED on pin 13.



### Code

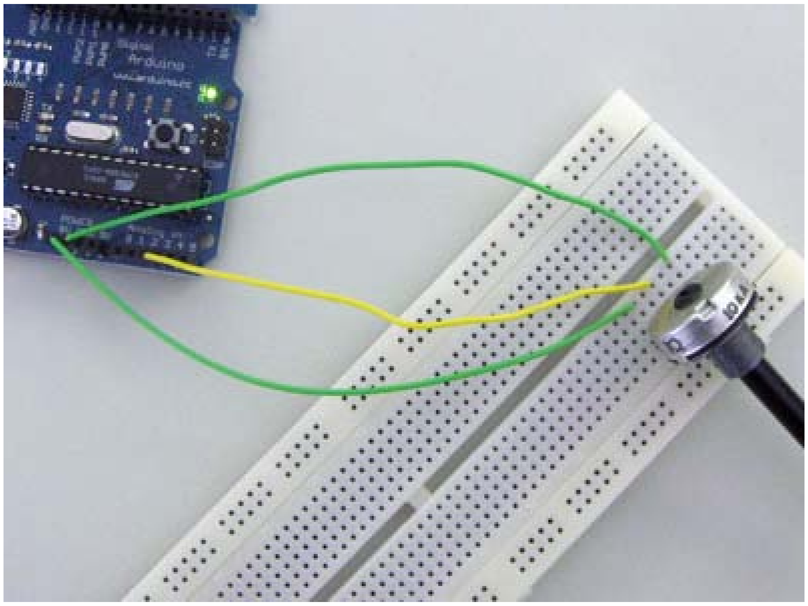
```
/* switch
 *
 * Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
 * press), the output pin is toggled from LOW to HIGH or HIGH to LOW.  There's
 * a minimum delay between toggles to debounce the circuit (i.e. to ignore
 * noise).
 *
 * David A. Mellis
 * 21 November 2006
 */

int inPin = 2;         // the number of the input pin
int outPin = 13;       // the number of the output pin

int state = HIGH;      // the current state of the output pin
int reading;          // the current reading from the input pin
int previous = LOW;    // the previous reading from the input pin

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 200;   // the debounce time, increase if the output flickers

void setup()
{
  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
}

void loop()
{
  reading = digitalRead(inPin);

  // if the input just went from LOW and HIGH and we've waited long enough
  // to ignore any noise on the circuit, toggle the output pin and remember
  // the time
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;

    time = millis();
  }

  digitalWrite(outPin, state);

  previous = reading;
}
```

## Reading a Potentiometer (analog input)

A potentiometer is a simple knob that provides a variable resistance, which we can read into the Arduino board as an analog value. In this example, that value controls the rate at which an LED blinks.

We connect three wires to the Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, we change the amount of resistence on either side of the wiper which is connected to the center pin of the potentiometer. This changes the relative "closeness" of that pin to 5 volts and ground, giving us a different analog input. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft is turned all the way in the other direction, there are 5 volts going to the pin and we read 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.



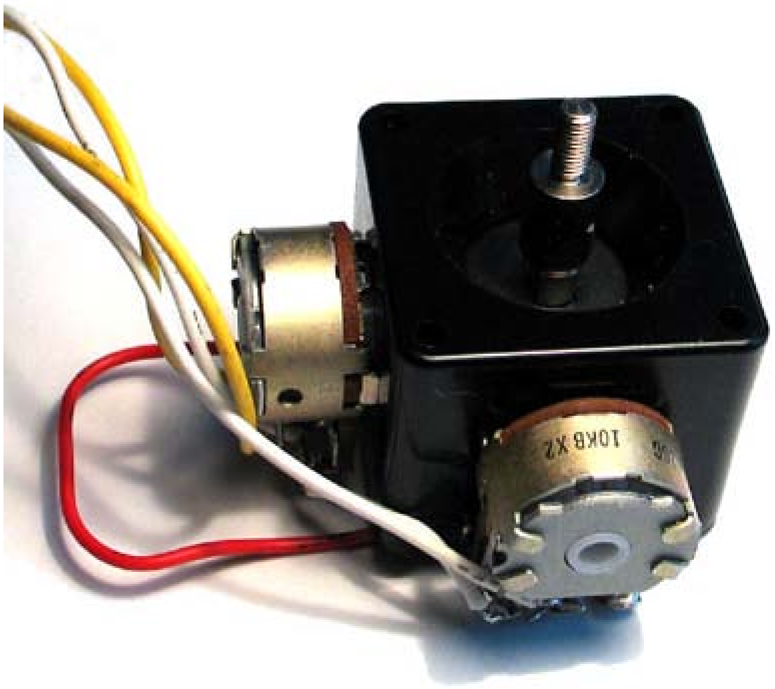
### Code

```
/* Analog Read to LED
 * ------------------
 *
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead(). In the easiest case we connect
 * a potentiometer to analog pin 2.
 *
 * Created 1 December 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 */
```

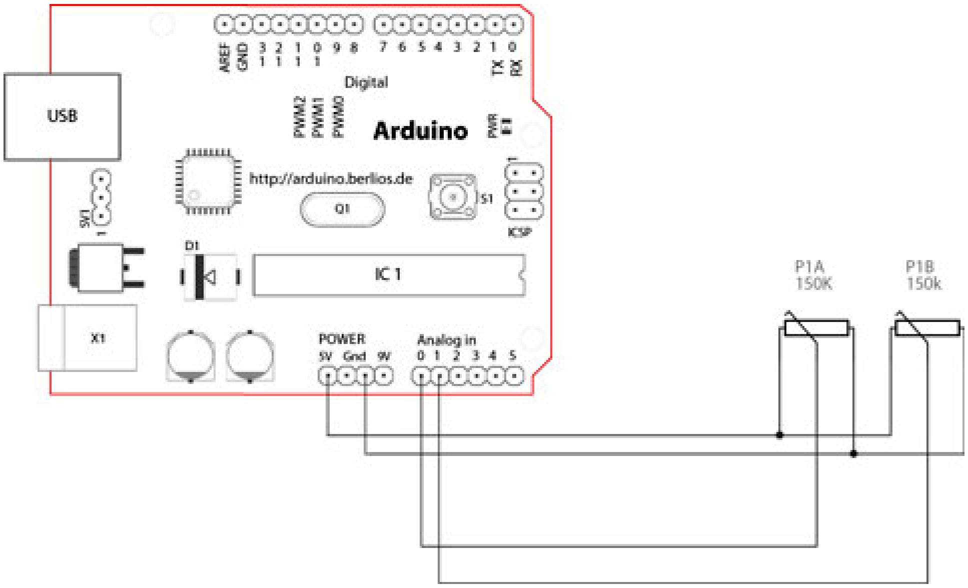
```
int potPin = 2;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor

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

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  digitalWrite(ledPin, HIGH);  // turn the ledPin on
  delay(val);                  // stop the program for some time
  digitalWrite(ledPin, LOW);   // turn the ledPin off
  delay(val);                  // stop the program for some time
}
```

# Interfacing a Joystick

### The Joystick



### Schematic



**How this works**

The joystick in the picture is nothing but two potentiometers that allow us to messure the movement of the stick in 2-D. Potentiometers are variable resistors and, in a way, they act as sensors providing us with a variable voltage depending on the rotation of the device around its shaft.

The kind of program that we need to monitor the joystick has to make a polling to two of the analog pins. We can send these values back to the computer, but then we face the classic problem that the transmission over the communication port has to be made with 8bit values, while our DAC (Digital to Analog Converter - that is messuring the values from the potentiometers in the joystick) has a resolution of 10bits. In other words this means that our sensors are characterized with a value between 0 and 1024.

The following code includes a method called *treatValue()* that is transforming the sensor's messurement into a value between 0 and 9 and sends it in ASCII back to the computer. This allows to easily send the information into e.g. Flash and parse it inside your own code.

Finally we make the LED blink with the values read from the sensors as a direct visual feedback of how we control the joystick.

```
/* Read Jostick
 * ------------
 *
 * Reads two analog pins that are supposed to be
 * connected to a jostick made of two potentiometers
 *
 * We send three bytes back to the comp: one header and two
 * with data as signed bytes, this will take the form:
 *            Jxy\r\n
 *
 * x and y are integers and sent in ASCII
 *
 * http://www.0j0.org | http://arduino.berlios.de
 * copyleft 2005 DojoDave for DojoCorp
 */

int ledPin = 13;
int joyPin1 = 0;                 // slider variable connecetd to analog pin 0
int joyPin2 = 1;                 // slider variable connecetd to analog pin 1
int value1 = 0;                  // variable to read the value from the analog pin 0
int value2 = 0;                  // variable to read the value from the analog pin 1

void setup() {
 pinMode(ledPin, OUTPUT);              // initializes digital pins 0 to 7 as outputs
 beginSerial(9600);
}

int treatValue(int data) {
 return (data * 9 / 1024) + 48;
}

void loop() {
 // reads the value of the variable resistor
 value1 = analogRead(joyPin1);
 // this small pause is needed between reading
 // analog pins, otherwise we get the same value twice
 delay(100);
 // reads the value of the variable resistor
 value2 = analogRead(joyPin2);

 digitalWrite(ledPin, HIGH);
 delay(value1);
 digitalWrite(ledPin, LOW);
 delay(value2);
 serialWrite('J');
 serialWrite(treatValue(value1));
 serialWrite(treatValue(value2));
 serialWrite(10);
 serialWrite(13);
}
```

@idea: the order of the blinking LED

@code: David Cuartielles

@pictures and graphics: Massimo Banzi

@date: 20050820 - Malmo - Sweden

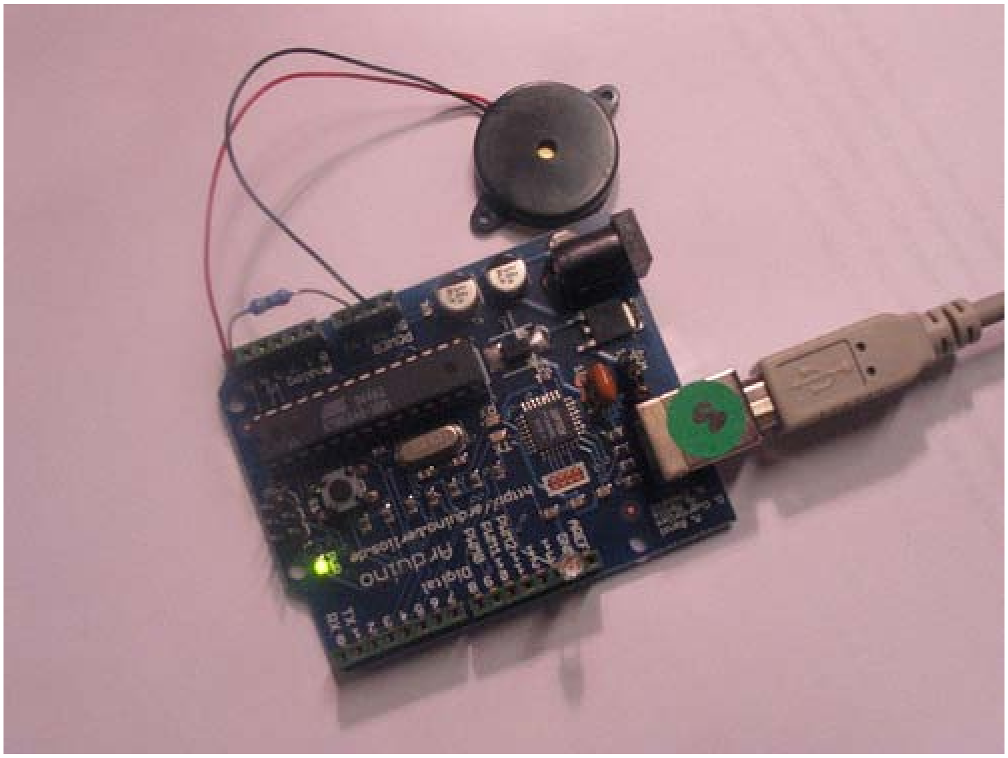
## Knock Sensor

Here we use a Piezo element to detect sound, what will allow us to use it as a knock sensor. We are taking advantage of the processors capability to read analog signals through its ADC - analog to digital converter. These converters read a voltage value and transform it into a value encoded digitally. In the case of the Arduino boards, we transform the voltage into a value in the range 0..1024. 0 represents 0volts, while 1024 represents 5volts at the input of one of the six analog pins.

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the analog input pin number 0, that supports the functionality of reading a value between 0 and 5volts, and not just a plain HIGH or LOW.

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the input. We also have to connect a resistor in the range of the Megaohms in parallel to the Piezo element; in the example we have plugged it directly in the female connectors. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc and are easier to use as input sensors.

The code example will capture the knock and if it is stronger than a certain threshold, it will send the string "Knock!" back to the computer over the serial port. In order to see this text you could either use a terminal program, which will read data from the serial port and show it in a window, or make your own program in e.g. Processing. Later in this article we propose a program that works for the software designed by Reas and Fry.



*Example of connection of a Piezo to analog pin 0 with a resistor*

```
/* Knock Sensor
 * ----------------
 *
 * Program using a Piezo element as if it was a knock sensor.
 *
 * We have to basically listen to an analog pin and detect
 * if the signal goes over a certain threshold. It writes
 * "knock" to the serial port if the Threshold is crossed,
 * and toggles the LED on pin 13.
 *
 * (cleft) 2005 D. Cuartielles for K3
 */

int ledPin = 13;
int knockSensor = 0;
byte val = 0;
int statePin = LOW;
int THRESHOLD = 100;

void setup() {
 pinMode(ledPin, OUTPUT);
 beginSerial(9600);
}

void loop() {
  val = analogRead(knockSensor);
  if (val >= THRESHOLD) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    printString("Knock!");
    printByte(10);
    printByte(13);
  }
  delay(100);  // we have to make a delay to avoid overloading the serial port
}
```

### Representing the Knock in Processing

If, e.g. we would like to capture this "knock" from the Arduino board, we have to look into how the information is transferred from the board over the serial port. First we see that whenever there is a knock bigger that the threshold, the program is printing (thus sending) "Knock!" over the serial port. Directly after sends the byte 10, what stands for EOLN or End Of LiNe, and byte 13, or CR - Carriage Return. Those two symbols will be useful to determine when the message sent by the board is over. Once that happens, the processing program will toggle the background color of the screen and print out "Knock!" in the command line.

```
// Knock In
// by David Cuartielles <http://www.0j0.org>
// based on Analog In by Josh Nimoy <http://itp.jtnimoy.com>

// Reads a value from the serial port and makes the background
// color toggle when there is a knock on a piezo used as a knock
// sensor.
// Running this example requires you have an Arduino board
// as peripheral hardware sending values and adding an EOLN + CR
// in the end. More information can be found on the Arduino
// pages: http://www.arduino.cc

// Created 23 November 2005
// Updated 23 November 2005

import processing.serial.*;

String buff = "";
int val = 0;
int NEWLINE = 10;

Serial port;
```

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

  // Open your serial port
  port = new Serial(this, "COMXX", 9600);  // <-- SUBSTITUTE COMXX with your serial port name!!
}

void draw()
{
  // Process each one of the serial port events
  while (port.available() > 0) {
    serialEvent(port.read());
  }
  background(val);
}

void serialEvent(int serial)
{
  if(serial != NEWLINE) {
    buff += char(serial);
  } else {
    buff = buff.substring(1, buff.length()-1);
    // Capture the string and print it to the commandline
    // we have to take from position 1 because
    // the Arduino sketch sends EOLN (10) and CR (13)
    if (val == 0) {
      val = 255;
    } else {
      val = 0;
    }
    println(buff);
    // Clear the value of "buff"
    buff = "";
   }
}
```

Edit Page | Page History | Printable View | All Recent Site Changes

```
/*
* Code for making one potentiometer control 3 LEDs, red, grn and blu, or one tri-color LED
* The program cross-fades from red to grn, grn to blu, and blu to red
* Debugging code assumes Arduino 0004, as it uses Serial.begin()-style functions
* Clay Shirky <clay.shirky@nyu.edu>
*/

// INPUT: Potentiometer should be connected to 5V and GND
int potPin = 3; // Potentiometer output connected to analog pin 3
int potVal = 0; // Variable to store the input from the potentiometer

// OUTPUT: Use digital pins 9-11, the Pulse-width Modulation (PWM) pins
// LED's cathodes should be connected to digital GND
int redPin = 9;   // Red LED,   connected to digital pin 9
int grnPin = 10;  // Green LED, connected to digital pin 10
int bluPin = 11;  // Blue LED,  connected to digital pin 11

// Program variables
int redVal = 0;   // Variables to store the values to send to the pins
int grnVal = 0;
int bluVal = 0;

int DEBUG = 1;          // Set to 1 to turn on debugging output

void setup()
{
  pinMode(redPin, OUTPUT);   // sets the pins as output
  pinMode(grnPin, OUTPUT);
  pinMode(bluPin, OUTPUT);

  if (DEBUG) {           // If we want to see the pin values for debugging...
    Serial.begin(9600);  // ...set up the serial ouput in 0004 format
  }
}

// Main program
void loop()
{
  potVal = analogRead(potPin);   // read the potentiometer value at the input pin

  if (potVal < 341)  // Lowest third of the potentiometer's range (0-340)
  {
    potVal = (potVal * 3) / 4; // Normalize to 0-255

    redVal = 256 - potVal;  // Red from full to off
    grnVal = potVal;        // Green from off to full
    bluVal = 1;             // Blue off
  }
  else if (potVal < 682) // Middle third of potentiometer's range (341-681)
  {
    potVal = ( (potVal-341) * 3) / 4; // Normalize to 0-255
    redVal = 1;            // Red off
    grnVal = 256 - potVal; // Green from full to off
    bluVal = potVal;       // Blue from off to full
  }
  else  // Upper third of potentiometer"s range (682-1023)
  {
    potVal = ( (potVal-683) * 3) / 4; // Normalize to 0-255

    redVal = potVal;       // Red from off to full
    grnVal = 1;            // Green off
    bluVal = 256 - potVal; // Blue from full to off
  }
  analogWrite(redPin, redVal);   // Write values to LED pins
  analogWrite(grnPin, grnVal);
  analogWrite(bluPin, bluVal);

  if (DEBUG) { // If we want to read the output
    DEBUG += 1;      // Increment the DEBUG counter
    if (DEBUG > 100) // Print every hundred loops
    {
      DEBUG = 1;     // Reset the counter
                             // Serial output using 0004-style functions
      Serial.print("R:");    // Indicate that output is red value
      Serial.print(redVal);  // Print red value
      Serial.print("\t");    // Print a tab
      Serial.print("G:");    // Repeat for grn and blu...
      Serial.print(grnVal);
      Serial.print("\t");
      Serial.print("B:");
      Serial.println(bluVal); // println, to end with a carriage return
    }
  }
}
```

# Arduino

search

Buy | Download | Getting Started | Learning | Reference | Hardware | FAQ Blog » | Forum » | Playground »

**Learning** Examples | Foundations | Hacking | Links

# Parallel to Serial Shifting-In with a CD4021BE

Started By Carlyn Maw and Tom Igoe Jan, '07

## Shifting In & the CD4021B

Sometimes you'll end up needing more digital input than the 13 pins on your Arduino board can readily handle. Using a parallel to serial shift register allows you collect information from 8 or more switches while only using 3 of the pins on your Arduino.

An example of a parallel to serial register is the CD4021B, sometimes referred to as an “8-Stage Static Shift Register.” This means you can read the state of up to 8 digital inputs attached to the register all at once. This is called Asynchronous Parallel Input. “Input” because you are collecting information. “Parallel” because it is all at once, like hearing a musical cord. “Asynchronous” because the CD4021B is doing all this data collection at its own pace without coordinating with the Arduino.

That happens in the next step when those 8 pin states are translated into a series of HIGH and LOW pulses on the serial-out pin of the shift register. This pin should be connected to an input pin on your Arduino Board, referred to as the **data pin**. The transfer of information itself is called Synchronous Serial Output because the shift register waits to deliver linear sequence of data to the Arduino until the Arduino asks for it. Synchronous Serial communication, input or output, is heavily reliant on what is referred to as a **clock pin.** That is what makes it “synchronous.” The clock pin is the metronome of the conversation between the shift register and the Arduino. Every time the Arduino sends the clock pin from LOW to HIGH it is telling the shift register “change the state of your Serial Output pin to tell me about the next switch.”

The third pin attached to the Arduino is a “Parallel to Serial Control” pin. You can think of it as a **latch pin**. When the latch pin is HIGH the shift register is listening to its 8 parallel ins. When it is LOW it is listening to the clock pin and passing information serially. That means every time the latch pin transitions from HIGH to LOW the shift register will start passing its most current switch information.

The pseudo code to coordinate this all looks something like this:

1. Make sure the register has the latest information from its parallel inputs (i.e. that the latch pin is HIGH)
2. Tell the register that I’m ready to get the information serially (latch pin LOW)
3. For each of the inputs I’m expecting, pulse the clockPin and then check to see if the data pin is low or high

This is a basic diagram.

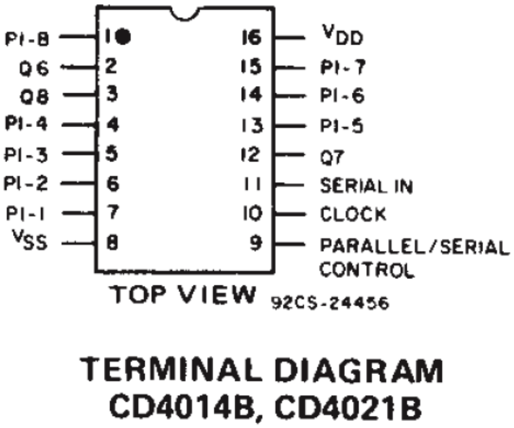
```
           _______
switch ->  |     |
switch ->  |  C  |
switch ->  |  D  |
switch ->  |  4  | -> Serial Data to Arduino
switch ->  |  0  |
switch ->  |  2  |
switch ->  |  1  | <- Clock Data from Arduino
switch ->  |_____| <- Latch Data from Arduino
```

If supplementing your Arduino with an additional 8 digital-ins isn’t going to be enough for your project you can have a second CD4021 pass its information on to that first CD4021 which will then be streaming all 16 bits of information to the Arduino in turn. If you know you will need to use multiple shift registers like this check that any shift registers you buy can handle Synchronous Serial Input as well as the standard Synchronous Serial Output capability. Synchronous Serial Input is the feature that allows the first shift register to receive and transmit the serial-output from the second one linked to it. The second example will cover this situation. You can keep extending this daisy-chain of shift registers until you have all the inputs you need, within reason.

```
           _______
```

```
switch ->  |     |
switch ->  |  C  |
switch ->  |  D  |
switch ->  |  4  | -> Serial Data to Arduino
switch ->  |  0  |
switch ->  |  2  | <- Clock Data from Arduino
switch ->  |  1  | <- Latch Data from Arduino
switch ->  |_____| <------
                         |
                         |
                         |
           _______       |  Serial Data Passed to First
switch ->  |     |        |  Shift Register
switch ->  |  C  |        |
switch ->  |  D  |        |
switch ->  |  4  | ______|
switch ->  |  0  |
switch ->  |  2  | <- Clock Data from Arduino
switch ->  |  1  | <- Latch Data from Arduino
switch ->  |_____|
```

There is more information about shifting in the ShiftOut tutorial, and before you start wiring up your board here is the pin diagram of the CD4021 from the Texas Instruments Datasheet

| | | | |
|---|---|---|---|
|  PI-8<br>Q6<br>Q8<br>PI-4<br>PI-3<br>PI-2<br>PI-1<br>VSS<br>1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>16<br>15<br>14<br>13<br>12<br>11<br>10<br>9<br>VDD<br>PI-7<br>PI-6<br>PI-5<br>Q7<br>SERIAL IN<br>CLOCK<br>PARALLEL/SERIAL CONTROL<br>TOP VIEW<br>92CS-24456<br>TERMINAL DIAGRAM<br>CD4014B, CD4021B | PINS 1,4-7, 13-15 | P1 – P8 (Pins 0-7) | Parallel Inputs |
| | PINS 2, 12, 3 | Q6, Q7, Q8 | Serial Output Pins from different steps in the sequence. Q7 is a pulse behind Q8 and Q6 is a pulse behind Q7. Q8 is the only one used in these examples. |
| | PIN 8 | Vss | GND |
| | PIN 9 | P/S C | Parallel/Serial Control (latch pin) |
| | PIN 10 | CLOCK | Shift register clock pin |
| | PIN 11 | SERIAL-IN | Serial data input |
| | PIN 16 | VDD | DC supply voltage |

## Example 1: One Shift Register

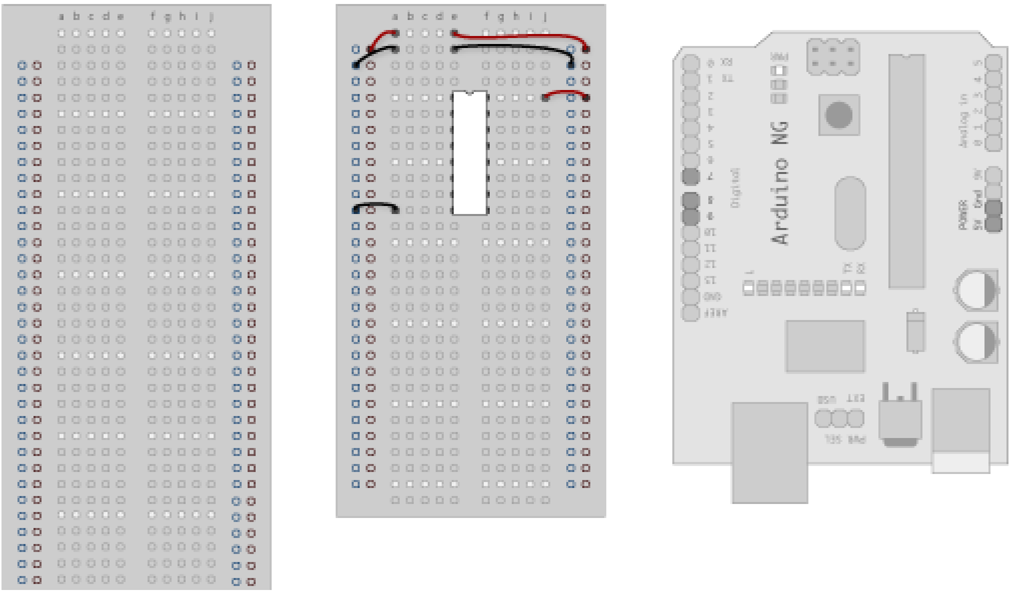
The first step is to extend your Arduino with one shift register.

### The Circuit

#### 1. Power Connections

Make the following connections:

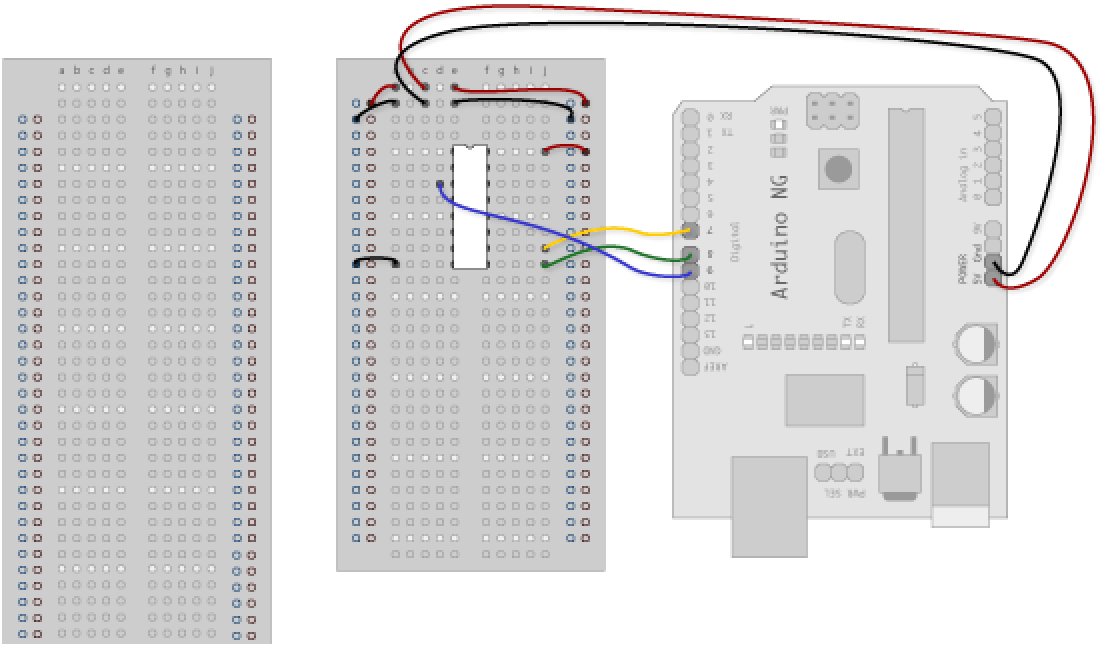
- GND (pin 8) to ground,
- VDD (pin 16) to 5V



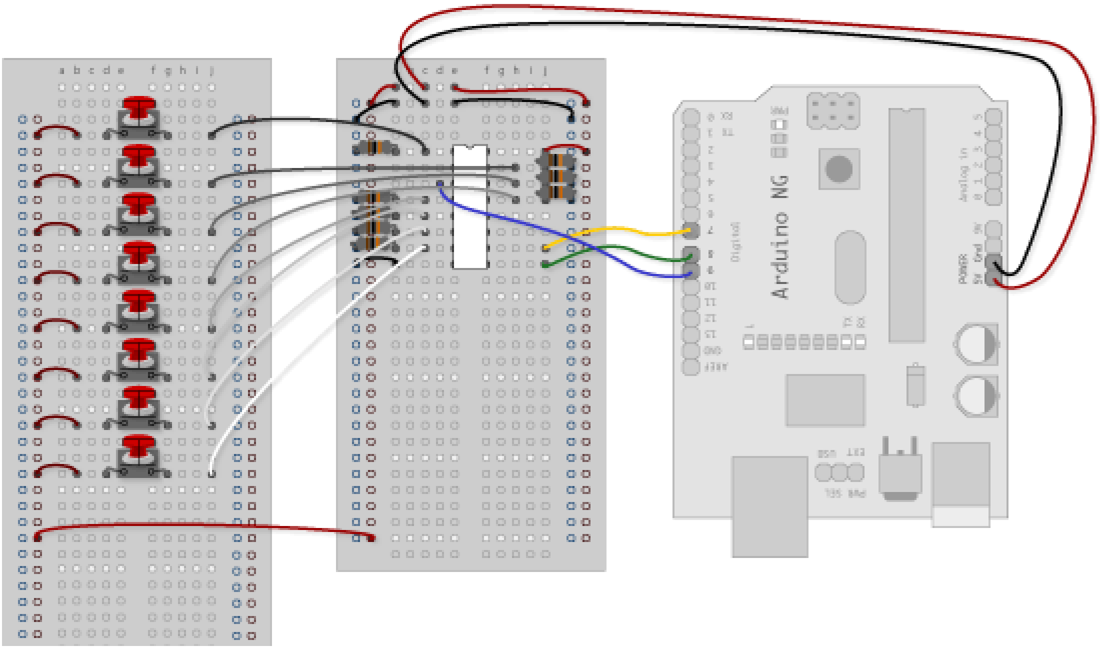
**2.Connect to Arduino**

- Q8 (pin 3) to Ardunio DigitalPin 9 (blue wire)
- CLOCK (pin 10) to to Ardunio DigitalPin 7 (yellow wire)
- P/S C (pin 9) to Ardunio DigitalPin 8 (green wire)

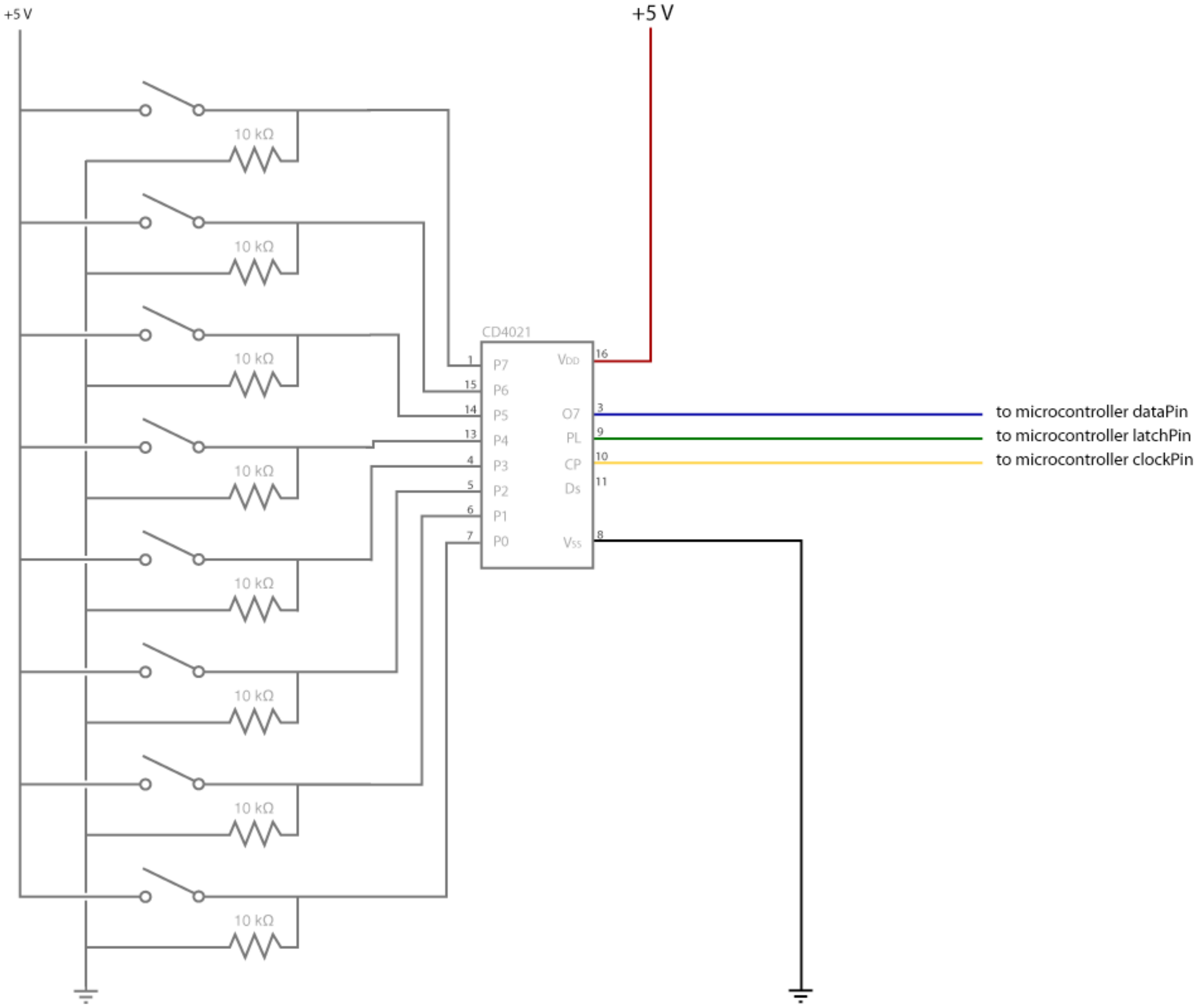
From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively.



**3. Add 8 Switches**



## Diagram



## The Code

Code Sample 1.1 – Hello World
Code Sample 1.2 – What is Pressed?
Code Sample 1.3 – Button Combination Check

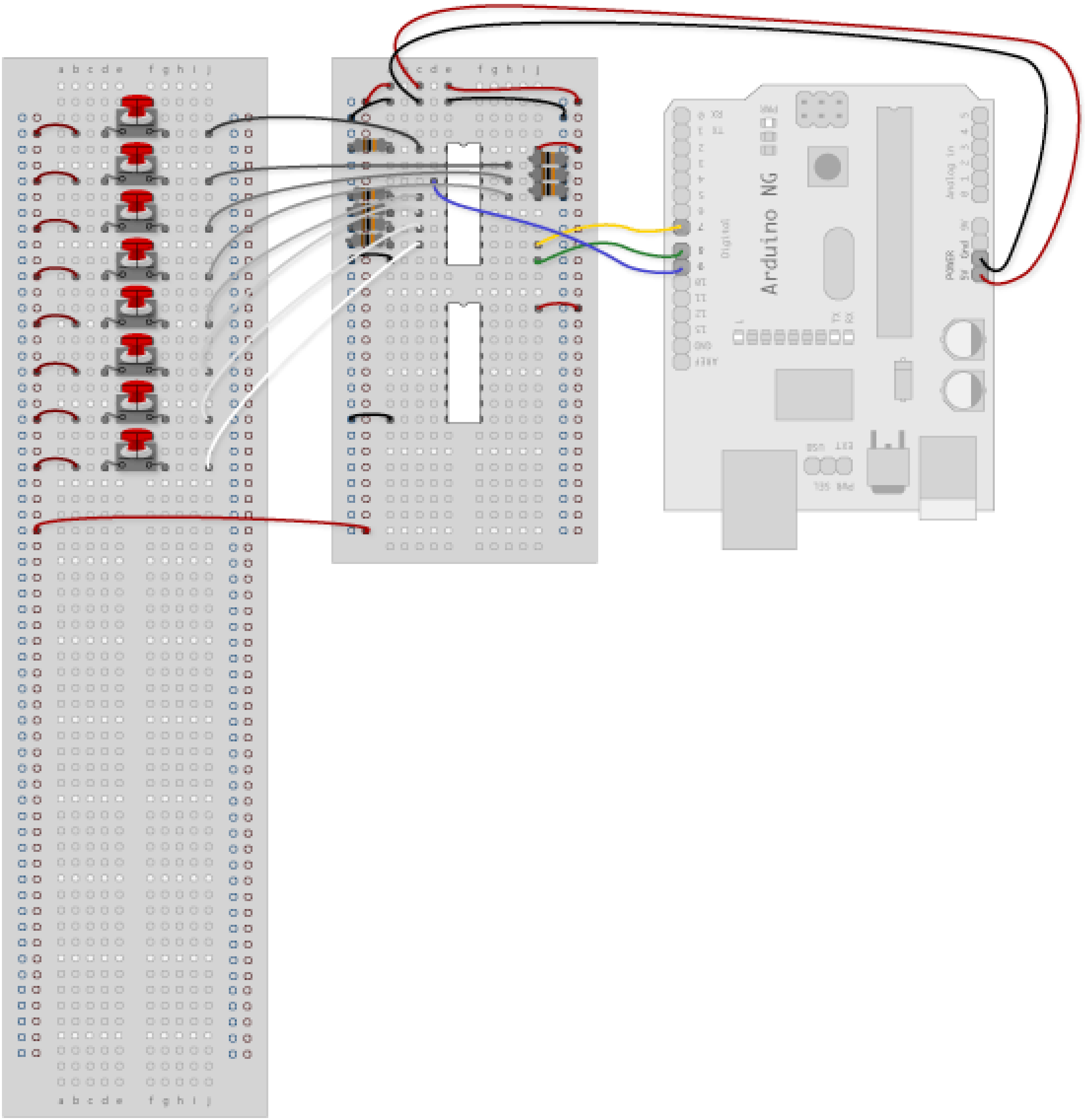
Code Sample 1.4 – Is it pressed? (sub-function)

## Example 2: Daisy Chained

In this example you'll add a second shift register, doubling the number of input pins while still using the same number of pins on the Arduino.

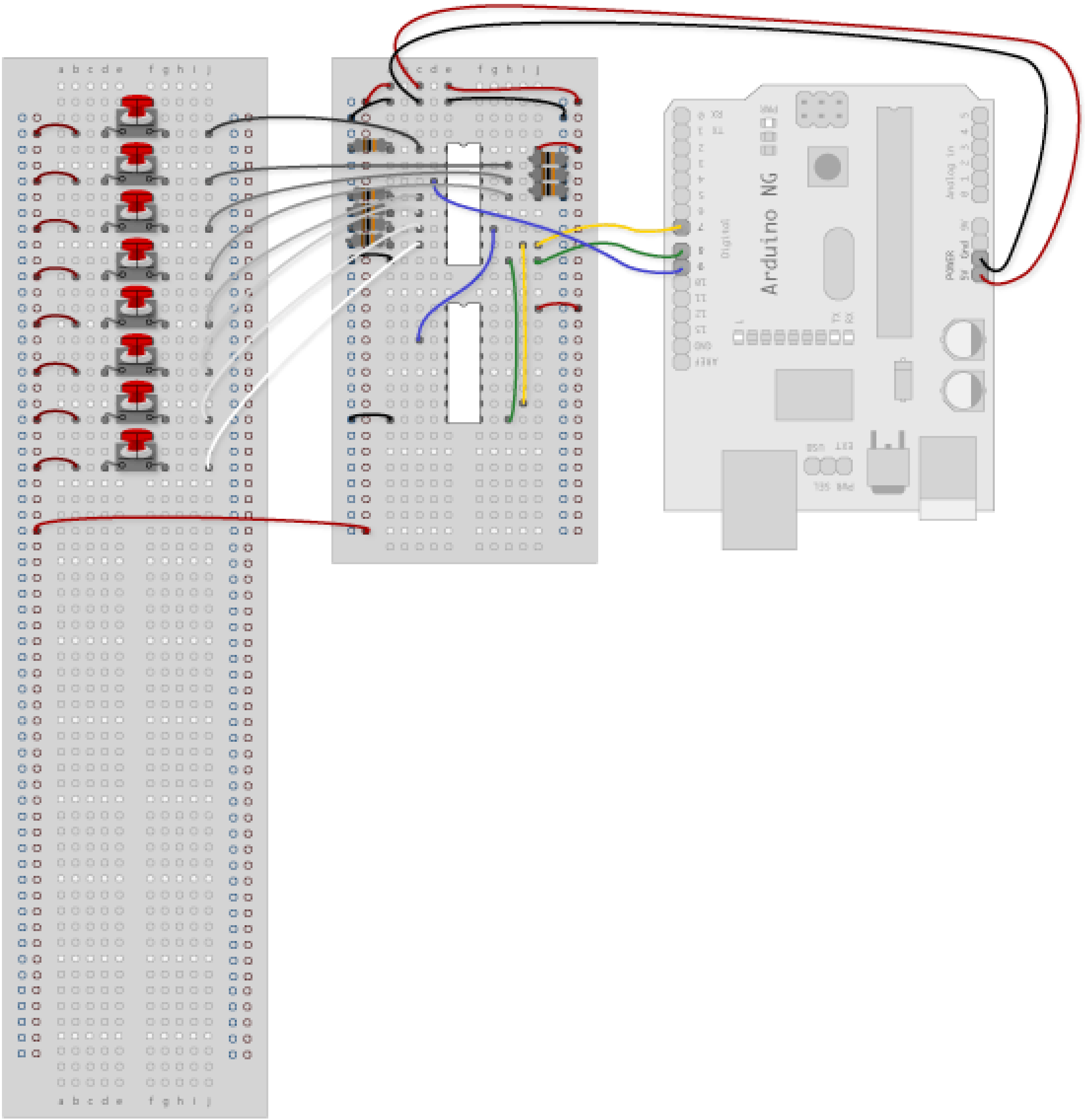
### The Circuit

**1. Add a second shift register.**



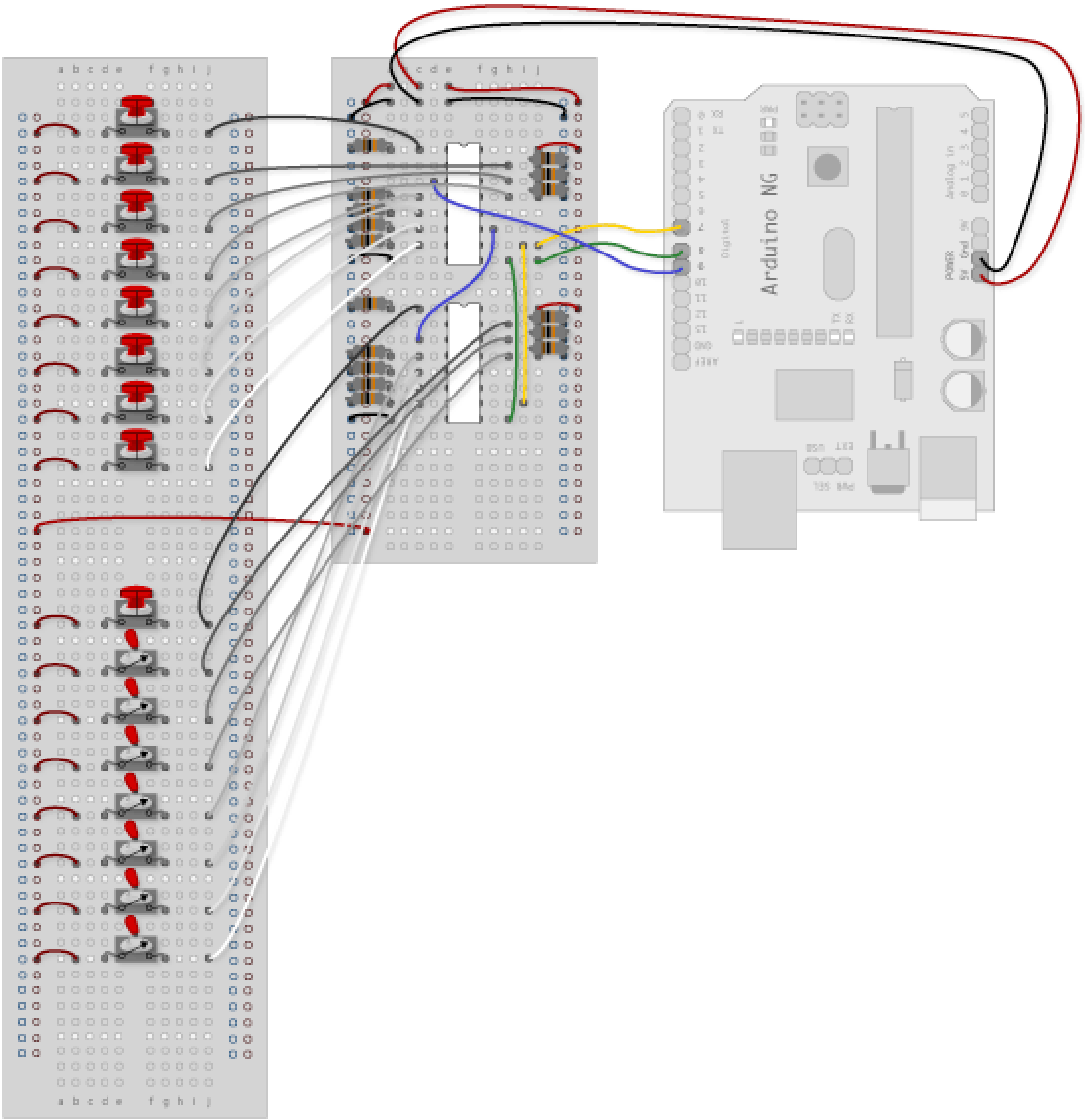
**2. Connect the 2 registers.**

Two of these connections simply extend the same clock and latch signal from the Arduino to the second shift register (yellow and green wires). The blue wire is going from the serial out pin (pin 9) of the first shift register to the serial data input (pin 14) of the second register.

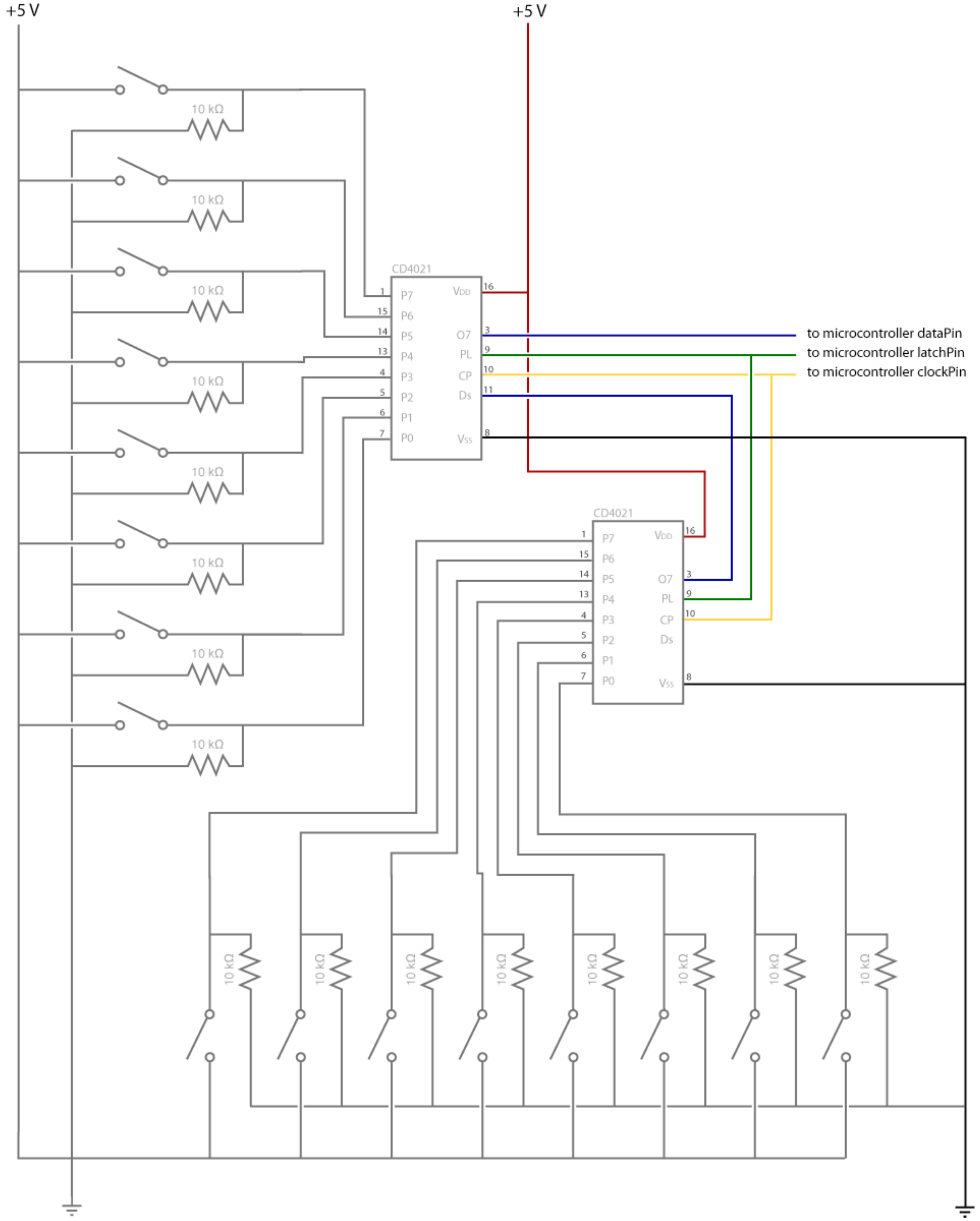


**3. Add a second set of Switches.**

Notice that there is one momentary switch and the rest are toggle switches. This is because the code examples will be using the switches attached to the second shift register as settings, like a preference file, rather than as event triggers. The one momentary switch will be telling the microcontroller that the setting switches are being changed.


a b c d e f g h i j
Arduino NG
Digital
Analog in
POWER
5V Gnd
GND
AREF
USB EXT
PWR SEL

**Diagram**


+5 V
+5 V
10 kΩ
CD4021
P7
P6
P5
P4
P3
P2
P1
P0
VDD
O7
PL
CP
Ds
VSS
1
15
14
13
4
5
6
7
16
3
9
10
11
8
to microcontroller dataPin
to microcontroller latchPin
to microcontroller clockPin

### The Code

Code Sample 2.1 – Hello World
Code Sample 2.2 – Using the second byte for settings, Print all
Code Sample 2.3 – Using the second byte for settings, Print on only (uses sub-function)

## Tutorial.HomePage-0007 History

Hide minor edits - Show changes to markup

June 11, 2007, at 11:10 PM by David A. Mellis - backing up the 0007 tutorials page
Added lines 1-91:

(:title Tutorials:)

## Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide.

(:table width=90% border=0 cellpadding=5 cellspacing=0:) (:cell width=50%:)

### Examples

**Digital Output**

- Blinking LED
- Blinking an LED without using the delay() function
- Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- More complex dimming/color crossfader
- Knight Rider example
- Shooting star
- PWM all of the digital pins in a sinewave pattern

**Digital Input**

- Digital Input and Output (from ITP physcomp labs)
- Read a Pushbutton
- Using a pushbutton as a switch
- Read a Tilt Sensor

**Analog Input**

- Read a Potentiometer
- Interfacing a Joystick
- Controlling an LED circle with a joystick
- Read a Piezo Sensor
- 3 LED cross-fades with a potentiometer
- 3 LED color mixer with 3 potentiometers

**Complex Sensors**

- Read an Accelerometer
- Read an Ultrasonic Range Finder (ultrasound sensor)
- Reading the qprox qt401 linear touch sensor
- Use two Arduino pins as a capacitive sensor

**Sound**

- Play Melodies with a Piezo Speaker
- More sound ideas
- Play Tones from the Serial Connection

- MIDI Output (from ITP physcomp labs) and from Spooky Arduino

**Interfacing w/ Hardware**

- Multiply the Amount of Outputs with an LED Driver
- Interfacing an LCD display with 8 bits
  - LCD interface library
- Driving a DC Motor with an L293 (from ITP physcomp labs).
- Driving a Unipolar Stepper Motor
- Implement a software serial connection
  - RS-232 computer interface
- Interface with a serial EEPROM using SPI
- Control a digital potentiometer using SPI
- Multiple digital outs with a 595 Shift Register
- Multiple digital inputs with a CD4021 Shift Register

### Other Arduino Examples

- Example labs from ITP
- Examples from Tom Igoe
- Examples from Jeff Gray

(:cell width=50%:)

### Interfacing with Other Software

- Introduction to Serial Communication (from ITP physcomp labs)
- Arduino + Flash
- Arduino + Processing
- Arduino + PD
- Arduino + MaxMSP
- Arduino + VVVV
- Arduino + Director
- Arduino + Ruby
- Arduino + C

### Tech Notes (from the forums or playground)

- Software serial (serial on pins besides 0 and 1)
- L297 motor driver
- Hex inverter
- Analog multiplexer
- Power supplies
- The components on the Arduino board
- Arduino build process
- AVRISP mkII on the Mac
- Non-volatile memory (EEPROM)
- Bluetooth
- Zigbee
- LED as light sensor (en Francais)
- Arduino and the Asuro robot
- Using Arduino from the command line

(:tableend:)

Restore

# Arduino : Tutorial / Tutorials

**Learning** Examples | Foundations | Hacking | Links

## Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide.

## Examples

### Digital Output

- Blinking LED
- Blinking an LED without using the delay() function
- Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- More complex dimming/color crossfader
- Knight Rider example
- Shooting star
- PWM all of the digital pins in a sinewave pattern

### Digital Input

- Digital Input and Output (from ITP physcomp labs)
- Read a Pushbutton
- Using a pushbutton as a switch
- Read a Tilt Sensor

### Analog Input

- Read a Potentiometer
- Interfacing a Joystick
- Controlling an LED circle with a joystick
- Read a Piezo Sensor
- 3 LED cross-fades with a potentiometer
- 3 LED color mixer with 3 potentiometers

### Complex Sensors

- Read an Accelerometer
- Read an Ultrasonic Range Finder (ultrasound sensor)
- Reading the qprox qt401 linear touch sensor
- Use two Arduino pins as a capacitive sensor

### Sound

- Play Melodies with a Piezo Speaker
- More sound ideas
- Play Tones from the Serial Connection
- MIDI Output (from ITP physcomp labs) and

## Interfacing with Other Software

- Introduction to Serial Communication (from ITP physcomp labs)
- Arduino + Flash
- Arduino + Processing
- Arduino + PD
- Arduino + MaxMSP
- Arduino + VVVV
- Arduino + Director
- Arduino + Ruby
- Arduino + C

## Tech Notes (from the forums or playground)

- Software serial (serial on pins besides 0 and 1)
- L297 motor driver
- Hex inverter
- Analog multiplexer
- Power supplies
- The components on the Arduino board
- Arduino build process
- AVRISP mkII on the Mac
- Non-volatile memory (EEPROM)
- Bluetooth
- Zigbee
- LED as light sensor (en Francais)
- Arduino and the Asuro robot
- Using Arduino from the command line

from Spooky Arduino

### Interfacing w/ Hardware

- Multiply the Amount of Outputs with an LED Driver
- Interfacing an LCD display with 8 bits
  - LCD interface library
- Driving a DC Motor with an L293 (from ITP physcomp labs).
- Driving a Unipolar Stepper Motor
- Implement a software serial connection
  - RS-232 computer interface
- Interface with a serial EEPROM using SPI
- Control a digital potentiometer using SPI
- Multiple digital outs with a 595 Shift Register
- Multiple digital inputs with a CD4021 Shift Register

## Other Arduino Examples

- Example labs from ITP
- Examples from Tom Igoe
- Examples from Jeff Gray

# Tutorial.Blink History

Hide minor edits - Show changes to markup

February 15, 2008, at 04:59 PM by David A. Mellis - clarifying that some boards have built-in leds, others have 1 KB resistor on pin 13
Changed lines 5-10 from:

The first program every programmer learns consists in writing enough code to make their code show the sentence "Hello World!" on a screen.

As a microcontroller, Arduino doesn't have any pre-established output devices. Willing to provide newcomers with some help while debugging programs, we propose the use of one of the board's pins plugging a LED that we will make blink indicating the right functionallity of the program.

We have added a 1K resistor to pin 13, what allows the immediate connection of a LED between that pin and ground.

to:

In most programming languages, the first program you write prints "hello world" to the screen. Since an Arduino board doesn't have a screen, we blink an LED instead.

The boards are designed to make it easy to blink an LED using digital pin 13. Some (like the Diecimila and LilyPad) have the LED built-in to the board. On most others (like the Mini and BT), there is a 1 KB resistor on the pin, allowing you to connect an LED directly. (To connect an LED to another digital pin, you should use an external resistor.)

Restore
February 03, 2007, at 08:32 AM by David A. Mellis -
Changed lines 3-4 from:

## Blinking LED

to:

## Blink

Restore
February 03, 2007, at 03:51 AM by David A. Mellis -
Changed lines 3-4 from:

## blink

to:

## Blinking LED

Restore
February 03, 2007, at 03:51 AM by David A. Mellis -
Changed lines 3-4 from:

## Blinking LED

to:

## blink

Restore
January 29, 2007, at 11:39 AM by David A. Mellis -
Changed lines 3-4 from:

## Blink

to:

## Blinking LED

Restore
January 28, 2007, at 05:06 AM by David A. Mellis -
Changed lines 3-4 from:

## blink

to:

## Blink

Added lines 13-14:

### Circuit

Changed lines 17-18 from:

## Code

to:

### Code

Restore
January 28, 2007, at 04:27 AM by David A. Mellis -
Changed line 21 from:

```
 * -----------
```

to:

```
 * ------------
```

Restore
January 28, 2007, at 04:27 AM by David A. Mellis -
Changed line 21 from:

```
 * ------------
```

to:

```
 * -----------
```

Deleted line 43:

```
  if (1 & 0)
```

Restore
January 28, 2007, at 04:27 AM by David A. Mellis -
Changed line 44 from:

```
  if (1 < 0)
```

to:

```
  if (1 & 0)
```

Restore
January 28, 2007, at 04:26 AM by David A. Mellis -
Changed line 19 from:

[=

to:

[@

Added line 44:

```
  if (1 < 0)
```

Changed line 50 from:

=]

to:

@]

Restore
January 28, 2007, at 04:26 AM by David A. Mellis -
Changed lines 15-19 from:

[@ // blink // <http://www.arduino.cc/en/Tutorial/Blink> int pin = 13;

to:

## Code

The example code is very simple, credits are to be found in the comments.

[= /* Blinking LED

```
 * ------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, in intervals of 2 seconds. Ideally we use pin 13 on the Arduino
 * board because it has a resistor attached to it, needing only an LED

 *
 * Created 1 June 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 * based on an orginal by H. Barragan for the Wiring i/o board
 */
```

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

Changed line 39 from:

```
  pinMode(pin, OUTPUT);
```

to:

```
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
```

Changed lines 44-47 from:

```
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(1000);
```

to:

```
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
```

Changed line 49 from:

@]

to:

=]

Restore
January 28, 2007, at 04:25 AM by David A. Mellis -
Changed lines 5-6 from:

This example blinks the LED on pin 13, turning it on for one second, then off for one second, and so on.

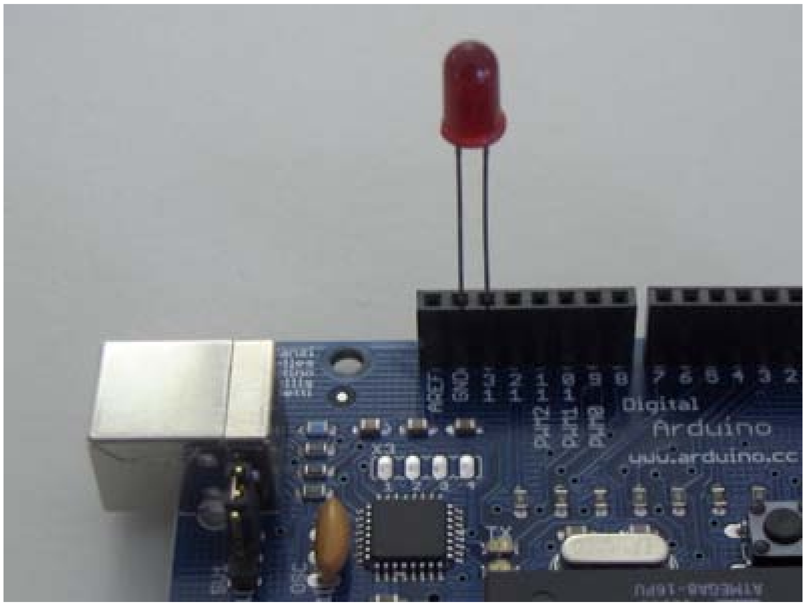
to:

The first program every programmer learns consists in writing enough code to make their code show the sentence "Hello World!" on a screen.

As a microcontroller, Arduino doesn't have any pre-established output devices. Willing to provide newcomers with some help while debugging programs, we propose the use of one of the board's pins plugging a LED that we will make blink indicating the right functionallity of the program.

We have added a 1K resistor to pin 13, what allows the immediate connection of a LED between that pin and ground.

LEDs have polarity, which means they will only light up if you orient the legs properly. The long leg is typically positive, and should connect to pin 13. The short leg connects to GND; the bulb of the LED will also typically have a flat edge on this side. If the LED doesn't light up, trying reversing the legs (you won't hurt the LED if you plug it in backwards for a short period of time).



Restore
January 28, 2007, at 04:14 AM by David A. Mellis -
Changed lines 1-2 from:

Examples > Digital I/O

to:

*Examples > Digital I/O*

Added lines 8-9:

// blink // <http://www.arduino.cc/en/Tutorial/Blink>

Restore
January 28, 2007, at 04:03 AM by David A. Mellis -
Changed lines 1-2 from:

## Examples > Digital I/O > blink

to:

Examples > Digital I/O

## blink

Restore
January 28, 2007, at 04:03 AM by David A. Mellis -
Changed lines 1-2 from:

## examples > digital > blink

to:

## Examples > Digital I/O > blink

Restore
January 28, 2007, at 04:02 AM by David A. Mellis -
Added lines 1-4:

## examples > digital > blink

This example blinks the LED on pin 13, turning it on for one second, then off for one second, and so on.

Restore
January 14, 2007, at 08:24 AM by David A. Mellis -
Added lines 1-16:

```
int pin = 13;

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

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

Restore

# Arduino : Tutorial / Blink

**Learning** Examples | Foundations | Hacking | Links

*Examples > Digital I/O*

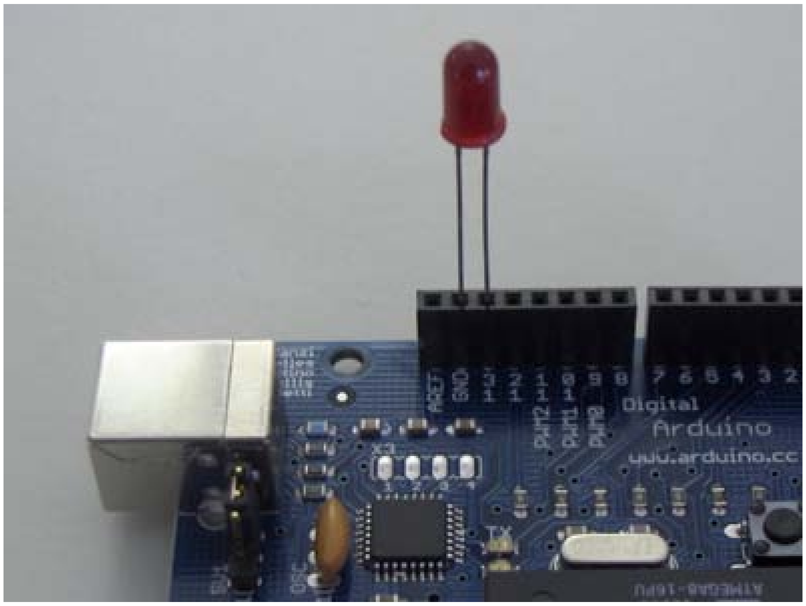
## Blink

In most programming languages, the first program you write prints "hello world" to the screen. Since an Arduino board doesn't have a screen, we blink an LED instead.

The boards are designed to make it easy to blink an LED using digital pin 13. Some (like the Diecimila and LilyPad) have the LED built-in to the board. On most others (like the Mini and BT), there is a 1 KB resistor on the pin, allowing you to connect an LED directly. (To connect an LED to another digital pin, you should use an external resistor.)

LEDs have polarity, which means they will only light up if you orient the legs properly. The long leg is typically positive, and should connect to pin 13. The short leg connects to GND; the bulb of the LED will also typically have a flat edge on this side. If the LED doesn't light up, trying reversing the legs (you won't hurt the LED if you plug it in backwards for a short period of time).

### Circuit



### Code

The example code is very simple, credits are to be found in the comments.

```
/* Blinking LED
 * ------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, in intervals of 2 seconds. Ideally we use pin 13 on the Arduino
 * board because it has a resistor attached to it, needing only an LED

 *
 * Created 1 June 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 * based on an orginal by H. Barragan for the Wiring i/o board
 */

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

```
void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

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

# Tutorial.BlinkWithoutDelay History

Hide minor edits - Show changes to markup

December 19, 2007, at 11:41 PM by David A. Mellis -
Changed lines 7-8 from:

## Code

to:

### Code

Restore
December 19, 2007, at 11:41 PM by David A. Mellis -
Changed lines 1-2 from:

## Blinking an LED without using the delay() function.

to:

*Examples > Digital I/O*

## Blink Without Delay

Restore
December 19, 2007, at 11:40 PM by David A. Mellis -
Deleted lines 7-18:

/* Blinking LED without using delay

```
 * --------------------------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, without using the delay() function.  this means that other code
 * can run at the same time without being interrupted by the LED code.
 *
 * Created 14 February 2006
 * David A. Mellis
 * http://arduino.berlios.de
 */
```

Restore
May 09, 2006, at 05:14 AM by David A. Mellis - int -> long
Deleted line 20:

int previousMillis = 0; // will store last time LED was updated

Changed lines 22-23 from:

int interval = 1000; // interval at which to blink (milliseconds)

to:

long previousMillis = 0; // will store last time LED was updated long interval = 1000; // interval at which to blink (milliseconds)

Restore
February 14, 2006, at 11:08 AM by 85.18.81.162 -

Added lines 1-49:

## Blinking an LED without using the delay() function.

Sometimes you need to blink an LED (or some other time sensitive function) at the same time as something else (like watching for a button press). That means you can't use delay(), or you'd stop everything else the program while the LED blinked. Here's some code that demonstrates how to blink the LED without using delay(). It keeps track of the last time it turned the LED on or off. Then, each time through loop() it checks if a sufficient interval has passed - if it has, it turns the LED off if it was on and vice-versa.

## Code

```
/* Blinking LED without using delay
 * --------------------------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, without using the delay() function.  this means that other code
 * can run at the same time without being interrupted by the LED code.
 *
 * Created 14 February 2006
 * David A. Mellis
 * http://arduino.berlios.de
 */

int ledPin = 13;                // LED connected to digital pin 13
int previousMillis = 0;         // will store last time LED was updated
int value = LOW;                // previous value of the LED
int interval = 1000;            // interval at which to blink (milliseconds)

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, is the difference
  // between the current time and last time we blinked the LED bigger than
  // the interval at which we want to blink the LED.
  if (millis() - previousMillis > interval) {
    previousMillis = millis();   // remember the last time we blinked the LED

    // if the LED is off turn it on and vice-versa.
    if (value == LOW)
      value = HIGH;
    else
      value = LOW;

    digitalWrite(ledPin, value);
  }
}
```

Restore

**Arduino : Tutorial / Blink Without Delay**

**Learning** Examples | Foundations | Hacking | Links

*Examples > Digital I/O*

# Blink Without Delay

Sometimes you need to blink an LED (or some other time sensitive function) at the same time as something else (like watching for a button press). That means you can't use delay(), or you'd stop everything else the program while the LED blinked. Here's some code that demonstrates how to blink the LED without using delay(). It keeps track of the last time it turned the LED on or off. Then, each time through loop() it checks if a sufficient interval has passed - if it has, it turns the LED off if it was on and vice-versa.

## Code

```
int ledPin = 13;                // LED connected to digital pin 13
int value = LOW;                // previous value of the LED
long previousMillis = 0;        // will store last time LED was updated
long interval = 1000;           // interval at which to blink (milliseconds)

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, is the difference
  // between the current time and last time we blinked the LED bigger than
  // the interval at which we want to blink the LED.
  if (millis() - previousMillis > interval) {
    previousMillis = millis();   // remember the last time we blinked the LED

    // if the LED is off turn it on and vice-versa.
    if (value == LOW)
      value = HIGH;
    else
      value = LOW;

    digitalWrite(ledPin, value);
  }
}
```

## Tutorial.Button History

Hide minor edits - Show changes to markup

January 17, 2008, at 09:02 AM by David A. Mellis -
Added lines 14-15:

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will more-or-less randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resister in the circuit.

Restore
June 12, 2007, at 08:56 AM by David A. Mellis - mentioning pull-down resistors (in addition to pull-up)
Added lines 12-13:

You can also wire this circuit the opposite way, with a pull-down resistor keeping the input LOW, and going HIGH when the button is pressed. If so, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.

Restore
April 11, 2007, at 11:10 AM by David A. Mellis -
Deleted lines 18-26:

/* invert

```
 * <http://www.arduino.cc/en/Tutorial/Invert>
 *
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13, when pressing a pushbutton attached to pin 7. It illustrates the
 * concept of Active-Low, which consists in connecting buttons using a
 * 1K to 10K pull-up resistor.
 *
 */
```

Restore
February 03, 2007, at 08:03 AM by David A. Mellis -
Added lines 1-45:

*Examples > Digital I/O*

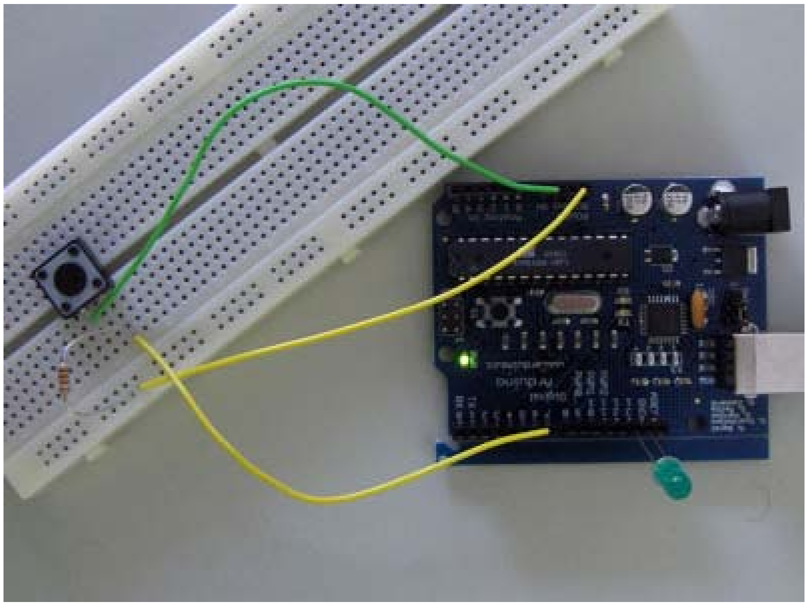
## Button

The pushbutton is a component that connects two points in a circuit when you press it. The example turns on an LED when you press the button.

We connect three wires to the Arduino board. The first goes from one leg of the pushbutton through a pull-up resistor (here 2.2 KOhms) to the 5 volt supply. The second goes from the corresponding leg of the pushbutton to ground. The third connects to a digital i/o pin (here pin 7) which reads the button's state.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to 5 volts (through the pull-up resistor) and we read a HIGH. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to ground, so that we read a LOW. (The pin is still connected to 5 volts, but the resistor in-between them means that the pin is "closer" to ground.)

### Circuit



## Code

```
/* invert
 * <http://www.arduino.cc/en/Tutorial/Invert>
 *
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13, when pressing a pushbutton attached to pin 7. It illustrates the
 * concept of Active-Low, which consists in connecting buttons using a
 * 1K to 10K pull-up resistor.
 *
 */
int ledPin = 13; // choose the pin for the LED
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

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

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

Restore

# Button

The pushbutton is a component that connects two points in a circuit when you press it. The example turns on an LED when you press the button.

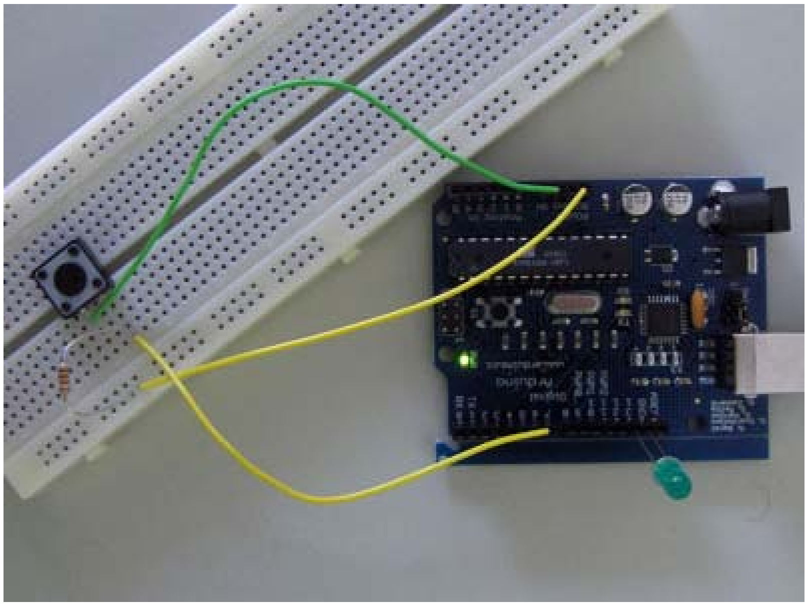
We connect three wires to the Arduino board. The first goes from one leg of the pushbutton through a pull-up resistor (here 2.2 KOhms) to the 5 volt supply. The second goes from the corresponding leg of the pushbutton to ground. The third connects to a digital i/o pin (here pin 7) which reads the button's state.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to 5 volts (through the pull-up resistor) and we read a HIGH. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to ground, so that we read a LOW. (The pin is still connected to 5 volts, but the resistor in-between them means that the pin is "closer" to ground.)

You can also wire this circuit the opposite way, with a pull-down resistor keeping the input LOW, and going HIGH when the button is pressed. If so, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will more-or-less randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resister in the circuit.

## Circuit



## Code

```
int ledPin = 13; // choose the pin for the LED
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

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

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

(Printable View of http://www.arduino.cc/en/Tutorial/Button)

## Tutorial.Debounce History

Hide minor edits - Show changes to markup

June 16, 2007, at 10:17 AM by David A. Mellis -
Changed lines 5-6 from:

This example demonstrates the use of a pushbutton as a switch: each time you press the button, the LED (or whatever) is turned on (if it's off) or off (if on). It also debounces the input, without which pressing the button once would appear to the code as multiple presses.

to:

This example demonstrates the use of a pushbutton as a switch: each time you press the button, the LED (or whatever) is turned on (if it's off) or off (if on). It also debounces the input, without which pressing the button once would appear to the code as multiple presses. Makes use of the millis() function to keep track of the time when the button is pressed.

Restore
June 16, 2007, at 10:14 AM by David A. Mellis -
Changed lines 9-10 from:

A push-button on pin 2 and an LED on pin 13.

to:

A push-button on pin 7 and an LED on pin 13.

Changed line 16 from:

int inPin = 2; // the number of the input pin

to:

int inPin = 7; // the number of the input pin

Restore
June 16, 2007, at 10:13 AM by David A. Mellis -
Changed lines 9-10 from:

A push-button on pin 7 and an LED on pin 13.

to:

A push-button on pin 2 and an LED on pin 13.

Changed lines 16-28 from:

/*

```
 * Debounce
 * by David A. Mellis
 *
 * Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
 * press), the output pin is toggled from LOW to HIGH or HIGH to LOW.  There's
 * a minimum delay between toggles to debounce the circuit (i.e. to ignore
 * noise).
 *
 * http://www.arduino.cc/en/Tutorial/Debounce
 */
```

int inPin = 7; // the number of the input pin

to:

int inPin = 2; // the number of the input pin

Changed lines 37-40 from:

```
  // if the input just went from LOW and HIGH and we've waited long enough
  // to ignore any noise on the circuit, toggle the output pin and remember
  // the time
```

to:

```
  // if we just pressed the button (i.e. the input went from LOW to HIGH),
  // and we've waited long enough since the last press to ignore any noise...
```

Added line 41:

```
    // ... invert the output
```

Changed lines 46-48 from:

```
    digitalWrite(outPin, state);
    time = millis();
```

to:

```
    // ... and remember when the last button press was
    time = millis();
```

Added lines 51-52:

```
  digitalWrite(outPin, state);
```

Restore
March 25, 2007, at 02:47 AM by David A. Mellis -
Changed lines 9-10 from:

A push-button on pin 2 and an LED on pin 13.

to:

A push-button on pin 7 and an LED on pin 13.

Changed line 28 from:

int inPin = 2; // the number of the input pin

to:

int inPin = 7; // the number of the input pin

Restore
March 25, 2007, at 02:46 AM by David A. Mellis -
Changed lines 16-18 from:

/* switch

to:

/*

```
 * Debounce
 * by David A. Mellis
```

Changed lines 25-26 from:

```
 * David A. Mellis
 * 21 November 2006
```

to:

```
 * http://www.arduino.cc/en/Tutorial/Debounce
```

Deleted lines 41-43:

```
  if (DEBUG)
    Serial.begin(19200);
```

Changed lines 59-60 from:

```
    time = millis();
```

to:

```
    digitalWrite(outPin, state);
    time = millis();
```

Changed lines 62-64 from:

```
  digitalWrite(outPin, state);
```

to:
Restore
March 25, 2007, at 02:45 AM by David A. Mellis -
Changed lines 3-4 from:

## Switch

to:

## Debounce

Restore
March 25, 2007, at 02:45 AM by David A. Mellis - Renaming Switch example
Added lines 1-68:

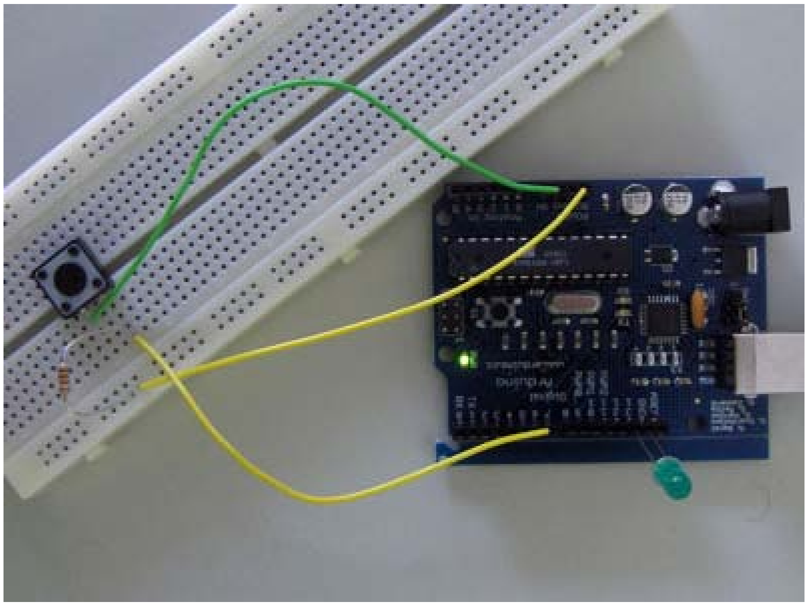
*Examples > Digital I/O*

## Switch

This example demonstrates the use of a pushbutton as a switch: each time you press the button, the LED (or whatever) is turned on (if it's off) or off (if on). It also debounces the input, without which pressing the button once would appear to the code as multiple presses.

### Circuit

A push-button on pin 2 and an LED on pin 13.



### Code

```
/* switch
 *
 * Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
 * press), the output pin is toggled from LOW to HIGH or HIGH to LOW.  There's
 * a minimum delay between toggles to debounce the circuit (i.e. to ignore
 * noise).
 *
 * David A. Mellis
 * 21 November 2006
 */

int inPin = 2;          // the number of the input pin
int outPin = 13;        // the number of the output pin

int state = HIGH;       // the current state of the output pin
int reading;            // the current reading from the input pin
int previous = LOW;     // the previous reading from the input pin

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;          // the last time the output pin was toggled
long debounce = 200;    // the debounce time, increase if the output flickers

void setup()
{
  if (DEBUG)
    Serial.begin(19200);

  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
}

void loop()
{
  reading = digitalRead(inPin);

  // if the input just went from LOW and HIGH and we've waited long enough
  // to ignore any noise on the circuit, toggle the output pin and remember
  // the time
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;

    time = millis();
  }

  digitalWrite(outPin, state);

  previous = reading;
}
```

Restore

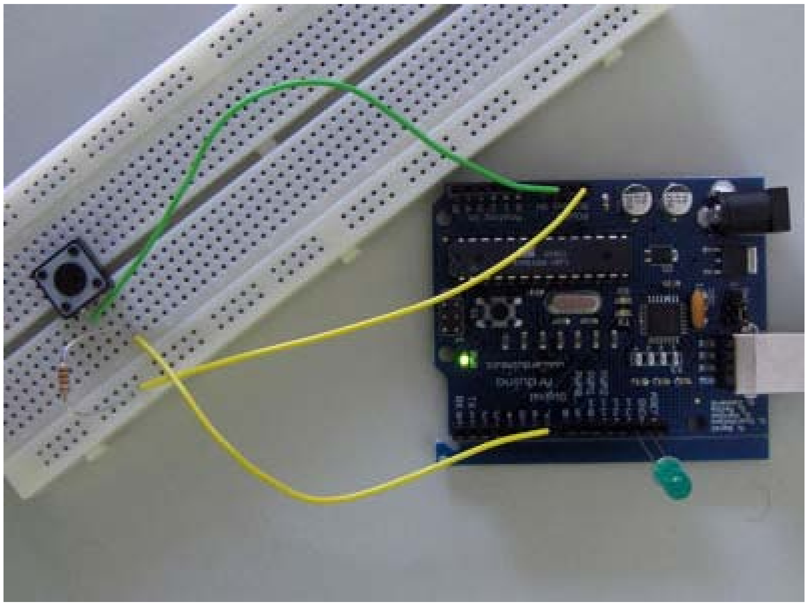
*Examples > Digital I/O*

# Debounce

This example demonstrates the use of a pushbutton as a switch: each time you press the button, the LED (or whatever) is turned on (if it's off) or off (if on). It also debounces the input, without which pressing the button once would appear to the code as multiple presses. Makes use of the millis() function to keep track of the time when the button is pressed.

## Circuit

A push-button on pin 7 and an LED on pin 13.



## Code

```
int inPin = 7;         // the number of the input pin
int outPin = 13;       // the number of the output pin

int state = HIGH;      // the current state of the output pin
int reading;           // the current reading from the input pin
int previous = LOW;    // the previous reading from the input pin

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 200;   // the debounce time, increase if the output flickers

void setup()
{
  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
}

void loop()
{
  reading = digitalRead(inPin);

  // if we just pressed the button (i.e. the input went from LOW to HIGH),
  // and we've waited long enough since the last press to ignore any noise...
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    // ... invert the output
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;

    // ... and remember when the last button press was
    time = millis();
  }

  digitalWrite(outPin, state);

  previous = reading;
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/Debounce)

## Tutorial.Loop History

Hide minor edits - Show changes to markup

September 23, 2007, at 08:37 PM by David A. Mellis -
Changed line 40 from:

```
    digitalWrite(i, HIGH);
```

to:

```
    digitalWrite(pins[i], HIGH);
```

Changed line 42 from:

```
    digitalWrite(i, LOW);
```

to:

```
    digitalWrite(pins[i], LOW);
```

Restore
April 22, 2007, at 12:20 PM by David A. Mellis -
Changed lines 18-19 from:

int timer = 100;

to:

int timer = 100; // The higher the number, the slower the timing. int pins[] = { 2, 3, 4, 5, 6, 7 }; // an array of pin numbers int num_pins = 6; // the number of pins (i.e. the length of the array)

Changed lines 26-27 from:

```
  for (i = 2; i <= 7; i++)
    pinMode(i, OUTPUT);
```

to:

```
  for (i = 0; i < num_pins; i++)   // the array elements are numbered from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output
```

Changed lines 34-39 from:

```
  for (i = 2; i <= 7; i++) {
```

to:

```
  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);   // turning it on,
    delay(timer);                  // pausing,
    digitalWrite(pins[i], LOW);    // and turning it off.
  }
  for (i = num_pins - 1; i >= 0; i--) {
```

Deleted line 42:

```
    delay(timer);
```

Deleted lines 43-48:

```
  for (i = 7; i >= 2; i--) {
    digitalWrite(i, HIGH);
    delay(timer);
    digitalWrite(i, LOW);
    delay(timer);
  }
```

Restore
April 15, 2007, at 10:33 AM by David A. Mellis -
Changed lines 3-6 from:

## Knight Rider

We have named this example in memory to a TV-series from the 80's where the famous David Hasselhoff had an AI machine driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects.

to:

## Loop

We also call this example "Knight Rider" in memory to a TV-series from the 80's where the famous David Hasselhoff had an AI machine driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects.

Restore
January 29, 2007, at 11:39 AM by David A. Mellis -
Changed lines 3-4 from:

## Loop

to:

## Knight Rider

Restore
January 28, 2007, at 05:01 AM by David A. Mellis -
Changed lines 1-4 from:

Examples > Digital I/O

## loop

to:

*Examples > Digital I/O*

## Loop

Restore
January 28, 2007, at 04:40 AM by David A. Mellis -
Added lines 11-12:

### Circuit

Added lines 15-16:

### Code

Restore
January 28, 2007, at 04:40 AM by David A. Mellis -
Changed lines 1-2 from:

## Knight Rider

to:

Examples > Digital I/O

## loop

Restore
January 28, 2007, at 04:28 AM by David A. Mellis -

Added lines 1-10:

## Knight Rider

We have named this example in memory to a TV-series from the 80's where the famous David Hasselhoff had an AI machine driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects.

Thus we decided that in order to learn more about sequential programming and good programming techniques for the I/O board, it would be interesting to use the **Knight Rider** as a metaphor.

This example makes use of 6 LEDs connected to the pins 2 - 7 on the board using 220 Ohm resistors. The first code example will make the LEDs blink in a sequence, one by one using only **digitalWrite(pinNum,HIGH/LOW)** and **delay(time)**. The second example shows how to use a **for(;;)** construction to perform the very same thing, but in fewer lines. The third and last example concentrates in the visual effect of turning the LEDs on/off in a more softer way.

http://static.flickr.com/27/61933851_3b9a25ab42.jpg

Restore
January 14, 2007, at 08:25 AM by David A. Mellis -
Added lines 1-29:

```
int timer = 100;

void setup()
{
  int i;

  for (i = 2; i <= 7; i++)
    pinMode(i, OUTPUT);
}

void loop()
{
  int i;

  for (i = 2; i <= 7; i++) {
    digitalWrite(i, HIGH);
    delay(timer);
    digitalWrite(i, LOW);
    delay(timer);
  }
  for (i = 7; i >= 2; i--) {
    digitalWrite(i, HIGH);
    delay(timer);
    digitalWrite(i, LOW);
    delay(timer);
  }
}
```

Restore

**Learning** Examples | Foundations | Hacking | Links

*Examples > Digital I/O*

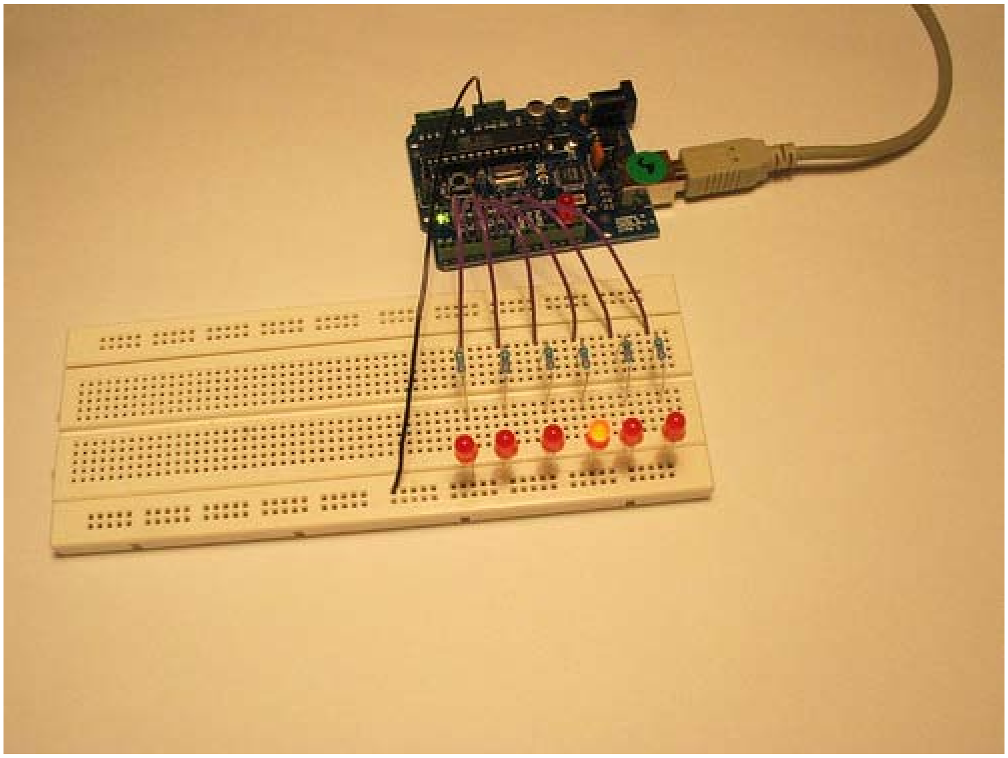
# Loop

We also call this example "Knight Rider" in memory to a TV-series from the 80's where the famous David Hasselhoff had an AI machine driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects.

Thus we decided that in order to learn more about sequential programming and good programming techniques for the I/O board, it would be interesting to use the **Knight Rider** as a metaphor.

This example makes use of 6 LEDs connected to the pins 2 - 7 on the board using 220 Ohm resistors. The first code example will make the LEDs blink in a sequence, one by one using only **digitalWrite(pinNum,HIGH/LOW)** and **delay(time)**. The second example shows how to use a **for(;;)** construction to perform the very same thing, but in fewer lines. The third and last example concentrates in the visual effect of turning the LEDs on/off in a more softer way.

## Circuit



## Code

```
int timer = 100;                    // The higher the number, the slower the timing.
int pins[] = { 2, 3, 4, 5, 6, 7 }; // an array of pin numbers
int num_pins = 6;                   // the number of pins (i.e. the length of the array)

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)   // the array elements are numbered from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output
}
```

```
void loop()
{
  int i;

  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);   // turning it on,
    delay(timer);                  // pausing,
    digitalWrite(pins[i], LOW);    // and turning it off.
  }
  for (i = num_pins - 1; i >= 0; i--) {
    digitalWrite(pins[i], HIGH);
    delay(timer);
    digitalWrite(pins[i], LOW);
  }
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/Loop)

# Tutorial.AnalogInput History

Hide minor edits - Show changes to markup

March 25, 2007, at 02:49 AM by David A. Mellis -
Added lines 1-43:

*Examples > Analog I/O*

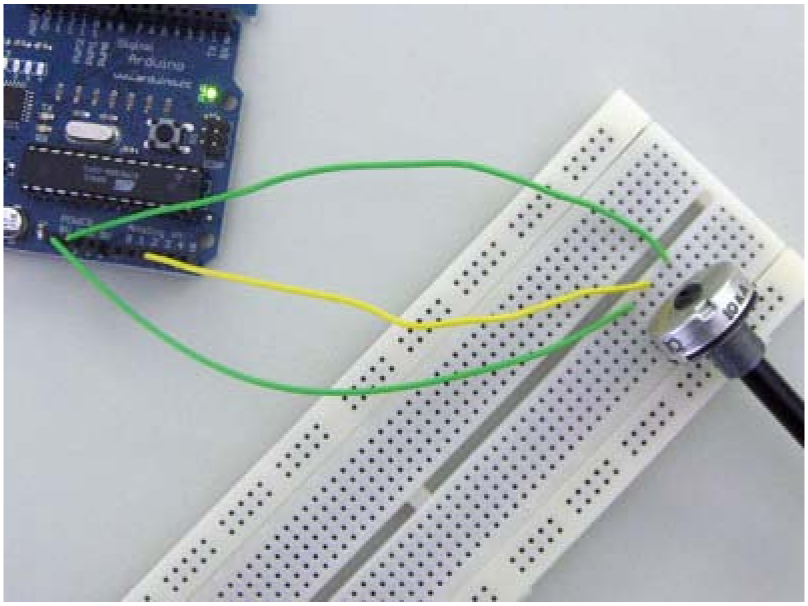
## Analog Input

A potentiometer is a simple knob that provides a variable resistance, which we can read into the Arduino board as an analog value. In this example, that value controls the rate at which an LED blinks.

We connect three wires to the Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, we change the amount of resistence on either side of the wiper which is connected to the center pin of the potentiometer. This changes the relative "closeness" of that pin to 5 volts and ground, giving us a different analog input. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft is turned all the way in the other direction, there are 5 volts going to the pin and we read 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

### Circuit



### Code

```
/*
 * AnalogInput
 * by DojoDave <http://www.0j0.org>
 *
 * Turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead(). In the easiest case we connect
 * a potentiometer to analog pin 2.
 */

int potPin = 2;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor

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

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  digitalWrite(ledPin, HIGH);  // turn the ledPin on
  delay(val);                  // stop the program for some time
  digitalWrite(ledPin, LOW);   // turn the ledPin off
  delay(val);                  // stop the program for some time
}
```

Restore

*Examples > Analog I/O*

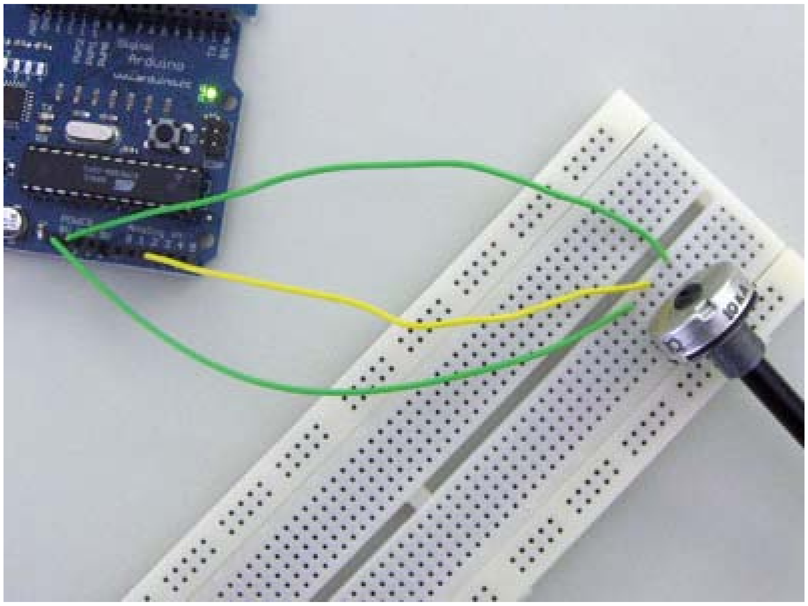
# Analog Input

A potentiometer is a simple knob that provides a variable resistance, which we can read into the Arduino board as an analog value. In this example, that value controls the rate at which an LED blinks.

We connect three wires to the Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, we change the amount of resistence on either side of the wiper which is connected to the center pin of the potentiometer. This changes the relative "closeness" of that pin to 5 volts and ground, giving us a different analog input. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft is turned all the way in the other direction, there are 5 volts going to the pin and we read 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

## Circuit



## Code

```
/*
 * AnalogInput
 * by DojoDave <http://www.0j0.org>
 *
 * Turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead(). In the easiest case we connect
 * a potentiometer to analog pin 2.
 */

int potPin = 2;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor

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

```
void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  digitalWrite(ledPin, HIGH);  // turn the ledPin on
  delay(val);                  // stop the program for some time
  digitalWrite(ledPin, LOW);   // turn the ledPin off
  delay(val);                  // stop the program for some time
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/AnalogInput)

## Tutorial.Fading History

Hide minor edits - Show changes to markup

March 26, 2007, at 06:54 AM by David A. Mellis -
Added lines 5-6:

Demonstrates the use of analog output (PWM) to fade an LED.

Added lines 9-10:

An LED connected to digital pin 9.

Restore
March 26, 2007, at 06:52 AM by David A. Mellis -
Added lines 1-31:

*Examples > Analog I/O*

## Fading

### Circuit

### Code

```
int value = 0;                           // variable to keep the actual value
int ledpin = 9;                          // light connected to digital pin 9

void setup()
{
  // nothing for setup
}

void loop()
{
  for(value = 0 ; value <= 255; value+=5) // fade in (from min to max)
  {
    analogWrite(ledpin, value);           // sets the value (range from 0 to 255)
    delay(30);                            // waits for 30 milli seconds to see the dimming effect
  }
  for(value = 255; value >=0; value-=5)   // fade out (from max to min)
  {
    analogWrite(ledpin, value);
    delay(30);
  }
}
```

Restore

## Arduino : Tutorial / Fading

**Learning** Examples | Foundations | Hacking | Links

*Examples > Analog I/O*

# Fading

Demonstrates the use of analog output (PWM) to fade an LED.

## Circuit

An LED connected to digital pin 9.

## Code

```
int value = 0;                            // variable to keep the actual value
int ledpin = 9;                           // light connected to digital pin 9

void setup()
{
  // nothing for setup
}

void loop()
{
  for(value = 0 ; value <= 255; value+=5) // fade in (from min to max)
  {
    analogWrite(ledpin, value);           // sets the value (range from 0 to 255)
    delay(30);                            // waits for 30 milli seconds to see the dimming effect
  }
  for(value = 255; value >=0; value-=5)   // fade out (from max to min)
  {
    analogWrite(ledpin, value);
    delay(30);
  }
}
```

# Arduino

search

Buy | Download | Getting Started | Learning | Reference | Hardware | FAQ    Blog » | Forum » | Playground »

## Tutorial.Knock History

Hide minor edits - Show changes to markup

April 08, 2008, at 06:21 PM by Paul Badger -
Changed line 49 from:

```
  }
```

to:

```
   }
```

Restore
April 08, 2008, at 06:21 PM by Paul Badger -
Changed line 48 from:

```
  delay(10);                        // short delay to avoid overloading the serial port
```

to:

```
    delay(10);                      // short delay to avoid overloading the serial port
```

Restore
April 08, 2008, at 06:21 PM by Paul Badger -
Changed lines 47-49 from:

```
    Serial.println("Knock!");          // send the string "Knock!" back to the computer, followed by
newline
```

```
  delay(10);  // short delay to avoid overloading the serial port
```

to:

```
    Serial.println("Knock!");       // send the string "Knock!" back to the computer, followed by newline
  delay(10);                        // short delay to avoid overloading the serial port
```

Restore
April 08, 2008, at 06:20 PM by Paul Badger -
Added lines 48-49:

```
  delay(10);  // short delay to avoid overloading the serial port
```

Deleted line 50:

```
  delay(100);  // we have to make a delay to avoid overloading the serial port
```

Restore
March 25, 2007, at 03:42 AM by David A. Mellis -
Changed lines 1-2 from:

## Knock Sensor

to:

*Examples > Analog I/O*

## Knock

Restore
March 25, 2007, at 03:42 AM by David A. Mellis -

Changed lines 9-10 from:

The code example will capture the knock and if it is stronger than a certain threshold, it will send the string "Knock!" back to the computer over the serial port. In order to see this text you could either use a terminal program, which will read data from the serial port and show it in a window, or make your own program in e.g. Processing. Later in this article we propose a program that works for the software designed by Reas and Fry.

to:

The code example will capture the knock and if it is stronger than a certain threshold, it will send the string "Knock!" back to the computer over the serial port. In order to see this text you can use the Arduino serial monitor.

Changed line 15 from:

[=

to:

[@

Changed lines 50-117 from:

=]

### Representing the Knock in Processing

If, e.g. we would like to capture this "knock" from the Arduino board, we have to look into how the information is transferred from the board over the serial port. First we see that whenever there is a knock bigger that the threshold, the program is printing (thus sending) "Knock!" over the serial port. Directly after sends the byte 10, what stands for EOLN or End Of LiNe, and byte 13, or CR - Carriage Return. Those two symbols will be useful to determine when the message sent by the board is over. Once that happens, the processing program will toggle the background color of the screen and print out "Knock!" in the command line.

```
// Knock In
// by David Cuartielles <http://www.0j0.org>
// based on Analog In by Josh Nimoy <http://itp.jtnimoy.com>

// Reads a value from the serial port and makes the background
// color toggle when there is a knock on a piezo used as a knock
// sensor.
// Running this example requires you have an Arduino board
// as peripheral hardware sending values and adding an EOLN + CR
// in the end. More information can be found on the Arduino
// pages: http://www.arduino.cc

// Created 23 November 2005
// Updated 23 November 2005

import processing.serial.*;

String buff = "";
int val = 0;
int NEWLINE = 10;

Serial port;

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

  // Open your serial port
  port = new Serial(this, "COMXX", 9600);  // <-- SUBSTITUTE COMXX with your serial port name!!
}

void draw()
{
  // Process each one of the serial port events
  while (port.available() > 0) {
    serialEvent(port.read());
  }
  background(val);
}

void serialEvent(int serial)
{
  if(serial != NEWLINE) {
    buff += char(serial);
  } else {
    buff = buff.substring(1, buff.length()-1);
    // Capture the string and print it to the commandline
    // we have to take from position 1 because
    // the Arduino sketch sends EOLN (10) and CR (13)
    if (val == 0) {
      val = 255;
    } else {
      val = 0;
    }
    println(buff);
    // Clear the value of "buff"
    buff = "";
   }
}
```

to:

@]

Restore
March 25, 2007, at 03:25 AM by David A. Mellis -
Added lines 1-117:

## Knock Sensor

Here we use a Piezo element to detect sound, what will allow us to use it as a knock sensor. We are taking advantage of the processors capability to read analog signals through its ADC - analog to digital converter. These converters read a voltage value and transform it into a value encoded digitally. In the case of the Arduino boards, we transform the voltage into a value in the range 0..1024. 0 represents 0volts, while 1024 represents 5volts at the input of one of the six analog pins.

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the analog input pin number 0, that supports the functionality of reading a value between 0 and 5volts, and not just a plain HIGH or LOW.

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the input. We also have to connect a resistor in the range of the Megaohms in parallel to the Piezo element; in the example we have plugged it directly in the female connectors. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc and are easier to use as input sensors.

The code example will capture the knock and if it is stronger than a certain threshold, it will send the string "Knock!" back to the computer over the serial port. In order to see this text you could either use a terminal program, which will read data from the serial port and show it in a window, or make your own program in e.g. Processing. Later in this article we propose a program that works for the software designed by Reas and Fry.

http://static.flickr.com/28/53535494_73f63436cb.jpg

*Example of connection of a Piezo to analog pin 0 with a resistor*

```
/* Knock Sensor
 * by DojoDave <http://www.0j0.org>
 *
 * Program using a Piezo element as if it was a knock sensor.
 *
 * We have to basically listen to an analog pin and detect
 * if the signal goes over a certain threshold. It writes
 * "knock" to the serial port if the Threshold is crossed,
 * and toggles the LED on pin 13.
 *
 * http://www.arduino.cc/en/Tutorial/Knock
 */

int ledPin = 13;      // led connected to control pin 13
int knockSensor = 0;  // the knock sensor will be plugged at analog pin 0
byte val = 0;         // variable to store the value read from the sensor pin
int statePin = LOW;   // variable used to store the last LED status, to toggle the light
int THRESHOLD = 100;  // threshold value to decide when the detected sound is a knock or not

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

void loop() {
  val = analogRead(knockSensor);   // read the sensor and store it in the variable "val"
  if (val >= THRESHOLD) {
    statePin = !statePin;          // toggle the status of the ledPin (this trick doesn't use time cycles)
    digitalWrite(ledPin, statePin); // turn the led on or off
    Serial.println("Knock!");         // send the string "Knock!" back to the computer, followed by
newline
  }
  delay(100);  // we have to make a delay to avoid overloading the serial port
}
```

### Representing the Knock in Processing

If, e.g. we would like to capture this "knock" from the Arduino board, we have to look into how the information is transferred from the board over the serial port. First we see that whenever there is a knock bigger that the threshold, the program is printing (thus sending) "Knock!" over the serial port. Directly after sends the byte 10, what stands for EOLN or End Of LiNe, and byte 13, or CR - Carriage Return. Those two symbols will be useful to determine when the message sent by the board is over. Once that happens, the processing program will toggle the background color of the screen and print out "Knock!" in the command line.

```
// Knock In
// by David Cuartielles <http://www.0j0.org>
// based on Analog In by Josh Nimoy <http://itp.jtnimoy.com>

// Reads a value from the serial port and makes the background
// color toggle when there is a knock on a piezo used as a knock
// sensor.
// Running this example requires you have an Arduino board
// as peripheral hardware sending values and adding an EOLN + CR
// in the end. More information can be found on the Arduino
// pages: http://www.arduino.cc

// Created 23 November 2005
// Updated 23 November 2005

import processing.serial.*;

String buff = "";
int val = 0;
int NEWLINE = 10;

Serial port;

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

  // Open your serial port
  port = new Serial(this, "COMXX", 9600);  // <-- SUBSTITUTE COMXX with your serial port name!!
}

void draw()
{
  // Process each one of the serial port events
  while (port.available() > 0) {
    serialEvent(port.read());
  }
  background(val);
}

void serialEvent(int serial)
{
  if(serial != NEWLINE) {
    buff += char(serial);
  } else {
    buff = buff.substring(1, buff.length()-1);
    // Capture the string and print it to the commandline
    // we have to take from position 1 because
    // the Arduino sketch sends EOLN (10) and CR (13)
    if (val == 0) {
      val = 255;
    } else {
      val = 0;
    }
    println(buff);
    // Clear the value of "buff"
    buff = "";
   }
}
```

Restore

*Examples > Analog I/O*

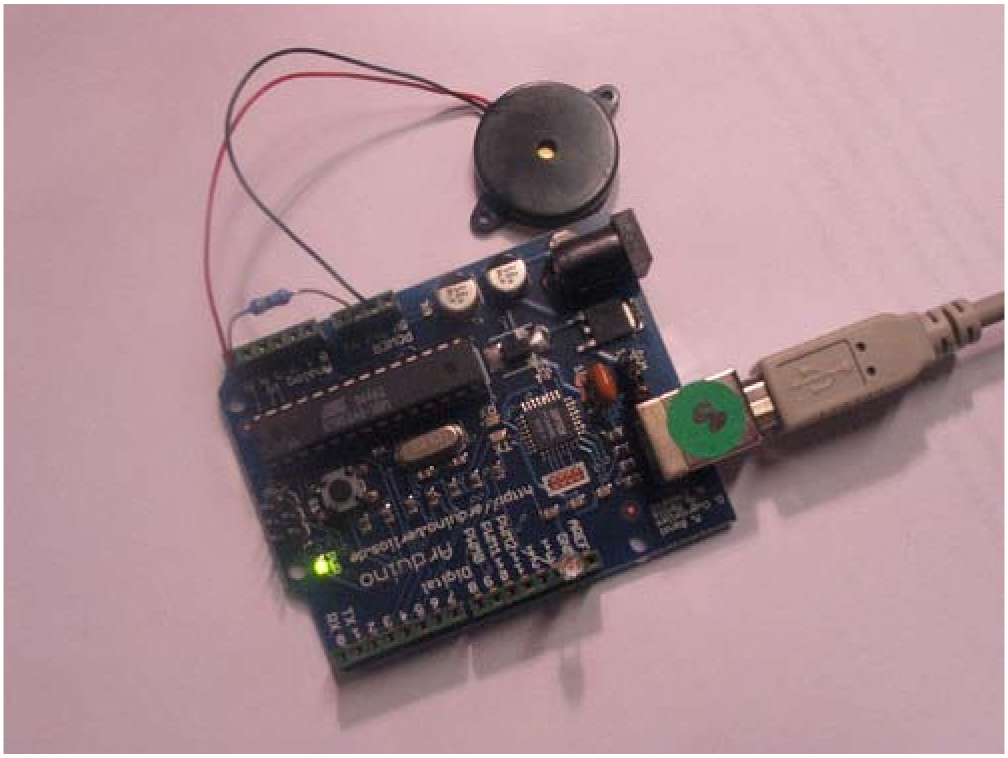
# Knock

Here we use a Piezo element to detect sound, what will allow us to use it as a knock sensor. We are taking advantage of the processors capability to read analog signals through its ADC - analog to digital converter. These converters read a voltage value and transform it into a value encoded digitally. In the case of the Arduino boards, we transform the voltage into a value in the range 0..1024. 0 represents 0volts, while 1024 represents 5volts at the input of one of the six analog pins.

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the analog input pin number 0, that supports the functionality of reading a value between 0 and 5volts, and not just a plain HIGH or LOW.

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the input. We also have to connect a resistor in the range of the Megaohms in parallel to the Piezo element; in the example we have plugged it directly in the female connectors. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc and are easier to use as input sensors.

The code example will capture the knock and if it is stronger than a certain threshold, it will send the string "Knock!" back to the computer over the serial port. In order to see this text you can use the Arduino serial monitor.



*Example of connection of a Piezo to analog pin 0 with a resistor*

```
/* Knock Sensor
 * by DojoDave <http://www.0j0.org>
 *
 * Program using a Piezo element as if it was a knock sensor.
 *
 * We have to basically listen to an analog pin and detect
 * if the signal goes over a certain threshold. It writes
 * "knock" to the serial port if the Threshold is crossed,
 * and toggles the LED on pin 13.
 *
 * http://www.arduino.cc/en/Tutorial/Knock
 */

int ledPin = 13;      // led connected to control pin 13
int knockSensor = 0;  // the knock sensor will be plugged at analog pin 0
byte val = 0;         // variable to store the value read from the sensor pin
int statePin = LOW;   // variable used to store the last LED status, to toggle the light
int THRESHOLD = 100;  // threshold value to decide when the detected sound is a knock or not

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

void loop() {
  val = analogRead(knockSensor);    // read the sensor and store it in the variable "val"
  if (val >= THRESHOLD) {
    statePin = !statePin;           // toggle the status of the ledPin (this trick doesn't use
time cycles)
    digitalWrite(ledPin, statePin); // turn the led on or off
    Serial.println("Knock!");       // send the string "Knock!" back to the computer, followed by
newline
    delay(10);                      // short delay to avoid overloading the serial port
   }
}
```

## Tutorial.Smoothing History

Hide minor edits - Show changes to markup

April 22, 2007, at 12:19 PM by David A. Mellis -
Changed lines 5-6 from:

Reads repeatedly from an analog input, calculating a running average and outputting it to an analog output. Demonstrates the use of arrays.

to:

Reads repeatedly from an analog input, calculating a running average and printing it to the computer. Demonstrates the use of arrays.

Changed lines 9-10 from:

Potentiometer on analog input pin 0, LED on pin 9.

to:

Potentiometer on analog input pin 0.

Deleted lines 13-22:

```
/*
```

```
 * Smoothing
 * David A. Mellis <dam@mellis.org>
 *
 * Reads repeatedly from an analog input, calculating a running average
 * and outputting it to an analog output.
 *
 * http://www.arduino.cc/en/Tutorial/Smoothing
 */
```

Changed lines 26-27 from:

int outputPin = 9;

to:
Added line 29:

```
  Serial.begin(9600);                      // initialize serial communication with computer
```

Changed line 45 from:

```
  analogWrite(outputPin, average / 4);    // analog inputs go up to 1023, outputs to 255
```

to:

```
  Serial.println(average);                 // send it to the computer (as ASCII digits)
```

Restore
March 25, 2007, at 03:02 AM by David A. Mellis -
Added lines 1-12:

*Examples > Analog I/O*

## Smoothing

Reads repeatedly from an analog input, calculating a running average and outputting it to an analog output. Demonstrates the use of arrays.

### Circuit

Potentiometer on analog input pin 0, LED on pin 9.

### Code

Changed lines 14-22 from:

1. define NUMSAMPLES 10

int samples[NUMSAMPLES]; int index = 0; int total = 0;

int sensor = 0; int actuator = 9;

to:

/*

```
 * Smoothing
 * David A. Mellis <dam@mellis.org>
 *
 * Reads repeatedly from an analog input, calculating a running average
 * and outputting it to an analog output.
 *
 * http://www.arduino.cc/en/Tutorial/Smoothing
 */
```

// Define the number of samples to keep track of. The higher the number, // the more the readings will be smoothed, but the slower the output will // respond to the input. Using a #define rather than a normal variable lets // use this value to determine the size of the readings array.

1. define NUMREADINGS 10

int readings[NUMREADINGS]; // the readings from the analog input int index = 0; // the index of the current reading int total = 0; // the running total int average = 0; // the average

int inputPin = 0; int outputPin = 9;

Changed lines 40-41 from:

```
  for (int i = 0; i < NUMSAMPLES; i++)
    samples[i] = 0;
```

to:

```
  for (int i = 0; i < NUMREADINGS; i++)
    readings[i] = 0;                      // initialize all the readings to 0
```

Changed lines 46-50 from:

```
  total -= samples[index];
  samples[index] = analogRead(sensor);
  total += samples[index];
  index = (index + 1) % NUMSAMPLES;
  analogWrite(actuator, total / NUMSAMPLES);
```

to:

```
  total -= readings[index];               // subtract the last reading
  readings[index] = analogRead(inputPin); // read from the sensor
  total += readings[index];               // add the reading to the total
  index = (index + 1);                    // advance to the next index

  if (index >= NUMREADINGS)               // if we're at the end of the array...
    index = 0;                            // ...wrap around to the beginning

  average = total / NUMREADINGS;          // calculate the average
  analogWrite(outputPin, average / 4);    // analog inputs go up to 1023, outputs to 255
```

Restore

January 14, 2007, at 08:28 AM by David A. Mellis -
Added line 1:

[@

Added line 25:

@]

Restore
January 14, 2007, at 08:28 AM by David A. Mellis -
Added lines 1-23:

1. define NUMSAMPLES 10

int samples[NUMSAMPLES]; int index = 0; int total = 0;

int sensor = 0; int actuator = 9;

void setup() {

```
  for (int i = 0; i < NUMSAMPLES; i++)
    samples[i] = 0;
```

}

void loop() {

```
  total -= samples[index];
  samples[index] = analogRead(sensor);
  total += samples[index];
  index = (index + 1) % NUMSAMPLES;
  analogWrite(actuator, total / NUMSAMPLES);
```

}

Restore

**Learning** Examples | Foundations | Hacking | Links

*Examples > Analog I/O*

# Smoothing

Reads repeatedly from an analog input, calculating a running average and printing it to the computer. Demonstrates the use of arrays.

## Circuit

Potentiometer on analog input pin 0.

## Code

```
// Define the number of samples to keep track of.  The higher the number,
// the more the readings will be smoothed, but the slower the output will
// respond to the input.  Using a #define rather than a normal variable lets
// use this value to determine the size of the readings array.
#define NUMREADINGS 10

int readings[NUMREADINGS];                // the readings from the analog input
int index = 0;                            // the index of the current reading
int total = 0;                            // the running total
int average = 0;                          // the average

int inputPin = 0;

void setup()
{
  Serial.begin(9600);                     // initialize serial communication with computer
  for (int i = 0; i < NUMREADINGS; i++)
    readings[i] = 0;                      // initialize all the readings to 0
}

void loop()
{
  total -= readings[index];               // subtract the last reading
  readings[index] = analogRead(inputPin); // read from the sensor
  total += readings[index];               // add the reading to the total
  index = (index + 1);                    // advance to the next index

  if (index >= NUMREADINGS)               // if we're at the end of the array...
    index = 0;                            // ...wrap around to the beginning

  average = total / NUMREADINGS;          // calculate the average
  Serial.println(average);                // send it to the computer (as ASCII digits)
}
```

## Tutorial.ASCIITable History

Hide minor edits - Show changes to markup

April 11, 2007, at 10:16 AM by David A. Mellis -
Added line 74:

...

Deleted lines 75-76:

...

Restore
April 11, 2007, at 10:16 AM by David A. Mellis -
Changed lines 60-76 from:

@]

to:

@]

### Output

```
ASCII Table ~ Character Map
!, dec: 33, hex: 21, oct: 41, bin: 100001
", dec: 34, hex: 22, oct: 42, bin: 100010
#, dec: 35, hex: 23, oct: 43, bin: 100011
$, dec: 36, hex: 24, oct: 44, bin: 100100
%, dec: 37, hex: 25, oct: 45, bin: 100101
&, dec: 38, hex: 26, oct: 46, bin: 100110
', dec: 39, hex: 27, oct: 47, bin: 100111
(, dec: 40, hex: 28, oct: 50, bin: 101000
```

...

Restore
March 26, 2007, at 07:02 AM by David A. Mellis -
Added lines 1-60:

*Examples > Communication*

## ASCII Table

Demonstrates the advanced serial printing functions by generating a table of characters and their ASCII values in decimal, hexadecimal, octal, and binary.

### Circuit

None, but the Arduino has to be connected to the computer.

### Code

```
// ASCII Table
// by Nicholas Zambetti <http://www.zambetti.com>

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

  // prints title with ending line break
  Serial.println("ASCII Table ~ Character Map");

  // wait for the long string to be sent
  delay(100);
}

int number = 33; // first visible character '!' is #33

void loop()
{
  Serial.print(number, BYTE);    // prints value unaltered, first will be '!'

  Serial.print(", dec: ");
  Serial.print(number);          // prints value as string in decimal (base 10)
  // Serial.print(number, DEC);  // this also works

  Serial.print(", hex: ");
  Serial.print(number, HEX);     // prints value as string in hexadecimal (base 16)

  Serial.print(", oct: ");
  Serial.print(number, OCT);     // prints value as string in octal (base 8)

  Serial.print(", bin: ");
  Serial.println(number, BIN);   // prints value as string in binary (base 2)
                                 // also prints ending line break

  // if printed last visible character '~' #126 ...
  if(number == 126) {
    // loop forever
    while(true) {
      continue;
    }
  }

  number++; // to the next character

  delay(100); // allow some time for the Serial data to be sent
}
```

Restore

*Examples > Communication*

# ASCII Table

Demonstrates the advanced serial printing functions by generating a table of characters and their ASCII values in decimal, hexadecimal, octal, and binary.

## Circuit

None, but the Arduino has to be connected to the computer.

## Code

```
// ASCII Table
// by Nicholas Zambetti <http://www.zambetti.com>

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

  // prints title with ending line break
  Serial.println("ASCII Table ~ Character Map");

  // wait for the long string to be sent
  delay(100);
}

int number = 33; // first visible character '!' is #33

void loop()
{
  Serial.print(number, BYTE);    // prints value unaltered, first will be '!'

  Serial.print(", dec: ");
  Serial.print(number);          // prints value as string in decimal (base 10)
  // Serial.print(number, DEC);  // this also works

  Serial.print(", hex: ");
  Serial.print(number, HEX);     // prints value as string in hexadecimal (base 16)

  Serial.print(", oct: ");
  Serial.print(number, OCT);     // prints value as string in octal (base 8)

  Serial.print(", bin: ");
  Serial.println(number, BIN);   // prints value as string in binary (base 2)
                                 // also prints ending line break

  // if printed last visible character '~' #126 ...
  if(number == 126) {
    // loop forever
    while(true) {
      continue;
    }
  }

  number++; // to the next character

  delay(100); // allow some time for the Serial data to be sent
}
```

## Output

```
ASCII Table ~ Character Map
!, dec: 33, hex: 21, oct: 41, bin: 100001
", dec: 34, hex: 22, oct: 42, bin: 100010
#, dec: 35, hex: 23, oct: 43, bin: 100011
```

```
$, dec: 36, hex: 24, oct: 44, bin: 100100
%, dec: 37, hex: 25, oct: 45, bin: 100101
&, dec: 38, hex: 26, oct: 46, bin: 100110
', dec: 39, hex: 27, oct: 47, bin: 100111
(, dec: 40, hex: 28, oct: 50, bin: 101000
...
```

(Printable View of http://www.arduino.cc/en/Tutorial/ASCIITable)

## Tutorial.Dimmer History

Hide minor edits - Show changes to markup

April 11, 2007, at 10:36 AM by David A. Mellis -
Added lines 1-78:

*Examples > Communication*

## 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.

### Circuit

An LED connected to pin 9 (with appropriate resistor).

### Code

```
int ledPin = 9;

void setup()
{
  // begin the serial communication
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  byte val;

  // check if data has been sent from the computer
  if (Serial.available()) {
    // read the most recent byte (which will be from 0 to 255)
    val = Serial.read();
    // set the brightness of the LED
    analogWrite(ledPin, val);
  }
}
```

### Processing Code

```
// Dimmer - sends bytes over a serial port
// by David A. Mellis

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);
}
```

Restore

*Examples > Communication*

# 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.

## Circuit

An LED connected to pin 9 (with appropriate resistor).

## Code

```
int ledPin = 9;

void setup()
{
  // begin the serial communication
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  byte val;

  // check if data has been sent from the computer
  if (Serial.available()) {
    // read the most recent byte (which will be from 0 to 255)
    val = Serial.read();
    // set the brightness of the LED
    analogWrite(ledPin, val);
  }
}
```

## Processing Code

```
// Dimmer - sends bytes over a serial port
// by David A. Mellis

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);
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/Dimmer)

# Tutorial.Graph History

Hide minor edits - Show changes to markup

March 25, 2007, at 06:14 AM by David A. Mellis -
Added lines 1-15:

*Examples > Communication*

## Graph

A simple example of communication from the Arduino board to the computer: the value of an analog input is printed. We call this "serial" communication because the connection appears to both the Arduino and the computer as an old-fashioned serial port, even though it may actually use a USB cable.

You can use the Arduino serial monitor to view the sent data, or it can be read by Processing (see code below), Flash, PD, Max/MSP, etc.

### Circuit

An analog input connected to analog input pin 0.

### Code

Changed line 24 from:

```
  Serial.print(analogRead(0) / 4, BYTE);
```

to:

```
  Serial.println(analogRead(0));
```

Changed lines 27-28 from:

/*

to:

@]

### Processing Code

[@ // Graph // by David A. Mellis // // Demonstrates reading data from the Arduino board by graphing the // values received. // // based on Analog In // by <a href="http://itp.jtnimoy.com">Josh Nimoy</a>.

Added lines 44-47:

String buff = ""; int NEWLINE = 10;

// Store the last 64 values received so we can graph them.

Changed lines 54-56 from:

```
  // Print a list in case COM1 doesn't work out
  //println("Available serial ports:");
  //printarr(PSerial.list());
```

to:

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

Changed lines 57-58 from:

```
//port = new Serial(this, "COM1", 9600);
// Uses the first available port
```

to:

```
// 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.
```

Added lines 63-66:

```
// 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);
```

Added line 74:

```
// Graph the stored values by drawing a lines between them.
```

Changed lines 84-87 from:

```
println(serial);

for (int i = 0; i < 63; i++)
  values[i] = values[i + 1];
```

to:

```
if (serial != NEWLINE) {
  // Store all the characters on the line.
  buff += char(serial);
} else {
  // The end of each line is marked by two characters, a carriage
  // return and a newline.  We're here because we've gotten a newline,
  // but we still need to strip off the carriage return.
  buff = buff.substring(0, buff.length()-1);
```

Changed lines 93-107 from:

```
values[63] = serial;
```

to:

```
  // Parse the String into an integer.  We divide by 4 because
  // analog inputs go from 0 to 1023 while colors in Processing
  // only go from 0 to 255.
  int val = Integer.parseInt(buff)/4;

  // Clear the value of "buff"
  buff = "";

  // Shift over the existing values to make room for the new one.
  for (int i = 0; i < 63; i++)
    values[i] = values[i + 1];

  // Add the received value to the array.
  values[63] = val;
}
```

Deleted line 108:

- /

Restore
January 14, 2007, at 08:38 AM by David A. Mellis -
Added lines 1-54:

```
void setup()
```

```
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print(analogRead(0) / 4, BYTE);
  delay(20);
}

/*
import processing.serial.*;

Serial port;
int[] values = new int[64];

void setup()
{
  size(512, 256);

  // Print a list in case COM1 doesn't work out
  //println("Available serial ports:");
  //printarr(PSerial.list());

  //port = new Serial(this, "COM1", 9600);
  // Uses the first available port
  port = new Serial(this, Serial.list()[0], 9600);
}

void draw()
{
  background(53);
  stroke(255);

  for (int i = 0; i < 63; i++)
    line(i * 8, 255 - values[i], (i + 1) * 8, 255 - values[i + 1]);

  while (port.available() > 0)
    serialEvent(port.read());
}

void serialEvent(int serial)
{
  println(serial);

  for (int i = 0; i < 63; i++)
    values[i] = values[i + 1];

  values[63] = serial;
}
*/
```

Restore

*Examples > Communication*

# Graph

A simple example of communication from the Arduino board to the computer: the value of an analog input is printed. We call this "serial" communication because the connection appears to both the Arduino and the computer as an old-fashioned serial port, even though it may actually use a USB cable.

You can use the Arduino serial monitor to view the sent data, or it can be read by Processing (see code below), Flash, PD, Max/MSP, etc.

## Circuit

An analog input connected to analog input pin 0.

## Code

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

void loop()
{
  Serial.println(analogRead(0));
  delay(20);
}
```

## Processing Code

```
// Graph
// by David A. Mellis
//
// Demonstrates reading data from the Arduino board by graphing the
// values received.
//
// based on Analog In
// by <a href="http://itp.jtnimoy.com">Josh Nimoy</a>.

import processing.serial.*;

Serial port;
String buff = "";
int NEWLINE = 10;

// Store the last 64 values received so we can graph them.
int[] values = new int[64];

void setup()
{
  size(512, 256);

  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()
{
  background(53);
  stroke(255);

  // Graph the stored values by drawing a lines between them.
  for (int i = 0; i < 63; i++)
    line(i * 8, 255 - values[i], (i + 1) * 8, 255 - values[i + 1]);

  while (port.available() > 0)
    serialEvent(port.read());
}

void serialEvent(int serial)
{
  if (serial != NEWLINE) {
    // Store all the characters on the line.
    buff += char(serial);
  } else {
    // The end of each line is marked by two characters, a carriage
    // return and a newline.  We're here because we've gotten a newline,
    // but we still need to strip off the carriage return.
    buff = buff.substring(0, buff.length()-1);

    // Parse the String into an integer.  We divide by 4 because
    // analog inputs go from 0 to 1023 while colors in Processing
    // only go from 0 to 255.
    int val = Integer.parseInt(buff)/4;

    // Clear the value of "buff"
    buff = "";

    // Shift over the existing values to make room for the new one.
    for (int i = 0; i < 63; i++)
      values[i] = values[i + 1];

    // Add the received value to the array.
    values[63] = val;
  }
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/Graph)

## Tutorial.PhysicalPixel History

Hide minor edits - Show changes to markup

April 11, 2007, at 10:12 AM by David A. Mellis -
Added lines 1-91:

*Examples > Communication*

## Physical Pixel

An example of using the Arduino board to receive data from the computer. In this case, the Arduino boards turns on an LED when it receives the character 'H', and turns off the LED when it receives the character 'L'.

The data can be sent from the Arduino serial monitor, or another program like Processing (see code below), Flash (via a serial-net proxy), PD, or Max/MSP.

### Circuit

An LED on pin 13.

### Code

```
int outputPin = 13;
int val;

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

void loop()
{
  if (Serial.available()) {
    val = Serial.read();
    if (val == 'H') {
      digitalWrite(outputPin, HIGH);
    }
    if (val == 'L') {
      digitalWrite(outputPin, LOW);
    }
  }
}
```

### Processing Code

```
// mouseover serial
// by BARRAGAN <http://people.interaction-ivrea.it/h.barragan>

// Demonstrates how to send data to the Arduino I/O board, in order to
// turn ON a light if the mouse is over a rectangle and turn it off
// if the mouse is not.

// created 13 May 2004
```

```
import processing.serial.*;

Serial port;

void setup()
{
  size(200, 200);
  noStroke();
  frameRate(10);

  // List all the available serial ports in the output pane.
  // You will need to choose the port that the Arduino board is
  // connected to from this list. The first port in the list is
  // port #0 and the third port in the list is port #2.
  println(Serial.list());

  // Open the port that the Arduino board is connected to (in this case #0)
  // Make sure to open the port at the same speed Arduino is using (9600bps)
  port = new Serial(this, Serial.list()[0], 9600);
}

// function to test if mouse is over square
boolean mouseOverRect()
{
  return ((mouseX >= 50)&&(mouseX <= 150)&&(mouseY >= 50)&(mouseY <= 150));
}

void draw()
{
  background(#222222);
  if(mouseOverRect())      // if mouse is over square
  {
    fill(#BBBBB0);         // change color
    port.write('H');       // send an 'H' to indicate mouse is over square
  } else {
    fill(#666660);         // change color
    port.write('L');       // send an 'L' otherwise
  }
  rect(50, 50, 100, 100);  // draw square
}
```

Restore

*Examples > Communication*

# Physical Pixel

An example of using the Arduino board to receive data from the computer. In this case, the Arduino boards turns on an LED when it receives the character 'H', and turns off the LED when it receives the character 'L'.

The data can be sent from the Arduino serial monitor, or another program like Processing (see code below), Flash (via a serial-net proxy), PD, or Max/MSP.

## Circuit

An LED on pin 13.

## Code

```
int outputPin = 13;
int val;

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

void loop()
{
  if (Serial.available()) {
    val = Serial.read();
    if (val == 'H') {
      digitalWrite(outputPin, HIGH);
    }
    if (val == 'L') {
      digitalWrite(outputPin, LOW);
    }
  }
}
```

## Processing Code

```
// mouseover serial
// by BARRAGAN <http://people.interaction-ivrea.it/h.barragan>

// Demonstrates how to send data to the Arduino I/O board, in order to
// turn ON a light if the mouse is over a rectangle and turn it off
// if the mouse is not.

// created 13 May 2004

import processing.serial.*;

Serial port;

void setup()
{
  size(200, 200);
  noStroke();
  frameRate(10);

  // List all the available serial ports in the output pane.
  // You will need to choose the port that the Arduino board is
  // connected to from this list. The first port in the list is
  // port #0 and the third port in the list is port #2.
  println(Serial.list());
  // Open the port that the Arduino board is connected to (in this case #0)
  // Make sure to open the port at the same speed Arduino is using (9600bps)
  port = new Serial(this, Serial.list()[0], 9600);
}

// function to test if mouse is over square
boolean mouseOverRect()
{
  return ((mouseX >= 50)&&(mouseX <= 150)&&(mouseY >= 50)&(mouseY <= 150));
}

void draw()
{
  background(#222222);
  if(mouseOverRect())      // if mouse is over square
  {
    fill(#BBBBB0);          // change color
    port.write('H');        // send an 'H' to indicate mouse is over square
  } else {
    fill(#666660);          // change color
    port.write('L');        // send an 'L' otherwise
  }
  rect(50, 50, 100, 100);  // draw square
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/PhysicalPixel)

## Tutorial.VirtualColorMixer History

Hide minor edits - Show changes to markup

March 26, 2007, at 07:39 AM by David A. Mellis -
Added lines 1-12:

*Examples > Communication*

## Virtual Color Mixer

Demonstrates one technique for sending multiple values from the Arduino board to the computer. In this case, the readings from three potentiometers are used to set the red, green, and blue components of the background color of a Processing sketch.

### Circuit

Potentiometers connected to analog input pins 0, 1, and 2.

### Code

Changed lines 33-37 from:
to:

@]

### Processing Code

[@

Deleted line 50:

/*

Deleted line 105:

- /

Restore
January 14, 2007, at 08:30 AM by David A. Mellis -
Added lines 1-92:

```
int redPin = 0;
int greenPin = 1;
int bluePin = 2;

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

void loop()
{
  Serial.print("R");
  Serial.println(analogRead(redPin));
  Serial.print("G");
  Serial.println(analogRead(greenPin));
  Serial.print("B");
  Serial.println(analogRead(bluePin));
  delay(100);
}

/**
 * Color Mixer
 * by David A. Mellis
 *
 * Created 2 December 2006
 *
 * based on Analog In
 * by <a href="http://itp.jtnimoy.com">Josh Nimoy</a>.
 *
 * Created 8 February 2003
 * Updated 2 April 2005
 */

/*
import processing.serial.*;

String buff = "";
int rval = 0, gval = 0, bval = 0;
int NEWLINE = 10;

Serial port;

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

  // Print a list in case COM1 doesn't work out
  println("Available serial ports:");
  println(Serial.list());

  //port = new Serial(this, "COM1", 9600);
  // Uses the first available port
  port = new Serial(this, Serial.list()[0], 9600);
}

void draw()
{
  while (port.available() > 0) {
    serialEvent(port.read());
  }
  background(rval, gval, bval);
}

void serialEvent(int serial)
{
  // If the variable "serial" is not equal to the value for
  // a new line, add the value to the variable "buff". If the
  // value "serial" is equal to the value for a new line,
  //  save the value of the buffer into the variable "val".
  if(serial != NEWLINE) {
    buff += char(serial);
  } else {
    // The first character tells us which color this value is for
    char c = buff.charAt(0);
    // Remove it from the string
    buff = buff.substring(1);
    // Discard the carriage return at the end of the buffer
    buff = buff.substring(0, buff.length()-1);
    // Parse the String into an integer
    if (c == 'R')
      rval = Integer.parseInt(buff);
    else if (c == 'G')
```

```
      gval = Integer.parseInt(buff);
    else if (c == 'B')
      bval = Integer.parseInt(buff);
    // Clear the value of "buff"
    buff = "";
  }
}
*/
```

Restore

**Arduino : Tutorial / Virtual Color Mixer**

**Learning** Examples | Foundations | Hacking | Links

*Examples > Communication*

# Virtual Color Mixer

Demonstrates one technique for sending multiple values from the Arduino board to the computer. In this case, the readings from three potentiometers are used to set the red, green, and blue components of the background color of a Processing sketch.

## Circuit

Potentiometers connected to analog input pins 0, 1, and 2.

## Code

```
int redPin = 0;
int greenPin = 1;
int bluePin = 2;

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

void loop()
{
  Serial.print("R");
  Serial.println(analogRead(redPin));
  Serial.print("G");
  Serial.println(analogRead(greenPin));
  Serial.print("B");
  Serial.println(analogRead(bluePin));
  delay(100);
}
```

## Processing Code

```
/**
 * Color Mixer
 * by David A. Mellis
 *
 * Created 2 December 2006
 *
 * based on Analog In
 * by <a href="http://itp.jtnimoy.com">Josh Nimoy</a>.
 *
 * Created 8 February 2003
 * Updated 2 April 2005
 */

import processing.serial.*;

String buff = "";
int rval = 0, gval = 0, bval = 0;
int NEWLINE = 10;

Serial port;

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

  // Print a list in case COM1 doesn't work out
  println("Available serial ports:");
  println(Serial.list());
  //port = new Serial(this, "COM1", 9600);
  // Uses the first available port
  port = new Serial(this, Serial.list()[0], 9600);
}

void draw()
{
  while (port.available() > 0) {
    serialEvent(port.read());
  }
  background(rval, gval, bval);
}

void serialEvent(int serial)
{
  // If the variable "serial" is not equal to the value for
  // a new line, add the value to the variable "buff". If the
  // value "serial" is equal to the value for a new line,
  //  save the value of the buffer into the variable "val".
  if(serial != NEWLINE) {
    buff += char(serial);
  } else {
    // The first character tells us which color this value is for
    char c = buff.charAt(0);
    // Remove it from the string
    buff = buff.substring(1);
    // Discard the carriage return at the end of the buffer
    buff = buff.substring(0, buff.length()-1);
    // Parse the String into an integer
    if (c == 'R')
      rval = Integer.parseInt(buff);
    else if (c == 'G')
      gval = Integer.parseInt(buff);
    else if (c == 'B')
      bval = Integer.parseInt(buff);
    // Clear the value of "buff"
    buff = "";
  }
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/VirtualColorMixer)

## Tutorial.TwoSwitchesOnePin History

Hide minor edits - Show changes to markup

April 08, 2008, at 08:24 PM by David A. Mellis -
Changed line 17 from:

```
 * Read two pushbutton switches or one center-off toggle switch with one Freeduino pin
```

to:

```
 * Read two pushbutton switches or one center-off toggle switch with one Arduino pin
```

Restore
April 08, 2008, at 08:23 PM by David A. Mellis - freeduino -> arduino
Changed lines 3-4 from:

There are handy 20K pullup resistors (resistors connected internally between Freeduino I/O pins and VCC - +5 volts in the Freeduino's case) built into the Atmega chip upon which Freeduino's are based. They are accessible from software by using the digitalWrite() function, when the pin is set to an input.

to:

There are handy 20K pullup resistors (resistors connected internally between Arduino I/O pins and VCC - +5 volts in the Arduino's case) built into the Atmega chip upon which Freeduino's are based. They are accessible from software by using the digitalWrite() function, when the pin is set to an input.

Restore
April 08, 2008, at 07:59 PM by Paul Badger -
Changed lines 9-12 from:

if both buttons are pushed at the same time. In this case the scheme just reports that sw2 is pushed. The job of the 10K series resistor incidentally is to prevent a short circuit if a pesky user pushes both buttons at once. It can be omitted on a center-off slide or toggle switch where the states are mutually exclusive.

to:

if both buttons are pushed at the same time. In this case the scheme just reports that sw2 is pushed. The job of the 10K series resistor, incidentally, is to prevent a short circuit if a pesky user pushes both buttons at once. It can be omitted on a center-off slide or toggle switch where the states are mutually exclusive.

Restore
April 08, 2008, at 07:57 PM by Paul Badger -
Deleted lines 25-30:

```
 * One downside of the scheme (there always has to be a downside doesn't there?) is that you can't tell
 * if both buttons are pushed at the same time. In this case the scheme just reports sw2 is pushed. Swap the 10k
 * resistor to the bottom of the schematic if you want it to favor sw1. The job of the 10K series resitor is to prevent
 * a short circuit if pesky user pushes both buttons at once. It can be ommitted on a center-off slide or toggle
 * where states are mutually exclusive.
 *
```

Restore
April 08, 2008, at 07:56 PM by Paul Badger -
Changed line 16 from:

```
 * Read_Two_Switches_ON_One_Pin
```

to:

```
 * Read_Two_Switches_On_One_Pin
```

Restore
April 08, 2008, at 07:56 PM by Paul Badger -
Changed lines 9-12 from:

if both buttons are pushed at the same time. In this case the scheme just reports that sw2 is pushed. The job of the 10K series resitor incidentally is to prevent a short circuit if a pesky user pushes both buttons at once. It can be ommitted on a center-off slide or toggle swithc where the states are mutually exclusive.

to:

if both buttons are pushed at the same time. In this case the scheme just reports that sw2 is pushed. The job of the 10K series resistor incidentally is to prevent a short circuit if a pesky user pushes both buttons at once. It can be omitted on a center-off slide or toggle switch where the states are mutually exclusive.

Restore
April 08, 2008, at 07:55 PM by Paul Badger -
Changed lines 5-7 from:

This sketch exploits the pullup resistors under software control. The idea is that an external 200K resistor to ground will cause the input pin to report LOW when the internal (20K) pullup resistor is turned off. When the internal pullup resistor is turned on, it will overwhelm the external 200K resistor and the pin will report HIGH.

to:

This sketch exploits the pullup resistors under software control. The idea is that an external 200K resistor to ground will cause an input pin to report LOW when the internal (20K) pullup resistor is turned off. When the internal pullup resistor is turned on however, it will overwhelm the external 200K resistor and the pin will report HIGH.

Restore
April 08, 2008, at 07:54 PM by Paul Badger -
Changed lines 3-4 from:

There are handy 20K pullup resistors (resistors connected internally between Freeduino I/O pins and VCC - +5 volts in the Freeduino's case). They are accessible from software by using the digitalWrite() function, when the pin is set to an input.

to:

There are handy 20K pullup resistors (resistors connected internally between Freeduino I/O pins and VCC - +5 volts in the Freeduino's case) built into the Atmega chip upon which Freeduino's are based. They are accessible from software by using the digitalWrite() function, when the pin is set to an input.

Restore
April 08, 2008, at 07:53 PM by Paul Badger -
Added lines 3-12:

There are handy 20K pullup resistors (resistors connected internally between Freeduino I/O pins and VCC - +5 volts in the Freeduino's case). They are accessible from software by using the digitalWrite() function, when the pin is set to an input.

This sketch exploits the pullup resistors under software control. The idea is that an external 200K resistor to ground will cause the input pin to report LOW when the internal (20K) pullup resistor is turned off. When the internal pullup resistor is turned on, it will overwhelm the external 200K resistor and the pin will report HIGH.

One downside of the scheme (there always has to be a downside doesn't there?) is that one can't tell if both buttons are pushed at the same time. In this case the scheme just reports that sw2 is pushed. The job of the 10K series resitor incidentally is to prevent a short circuit if a pesky user pushes both buttons at once. It can be ommitted on a center-off slide or toggle swithc where the states are mutually exclusive.

Restore
April 08, 2008, at 07:43 PM by Paul Badger -
Added lines 1-86:

## Read Two Switches With One I/O Pin

```
/*
```

```
 * Read_Two_Switches_ON_One_Pin
 * Read two pushbutton switches or one center-off toggle switch with one Freeduino pin
 * Paul Badger 2008
 * From an idea in EDN (Electronic Design News)
 *
 * Exploits the pullup resistors available on each I/O and analog pin
 * The idea is that the 200K resistor to ground will cause the input pin to report LOW when the
 * (20K) pullup resistor is turned off, but when the pullup resistor is turned on,
 * it will overwhelm the 200K resistor and the pin will report HIGH.
 *
 * One downside of the scheme (there always has to be a downside doesn't there?) is that you can't tell
 * if both buttons are pushed at the same time. In this case the scheme just reports sw2 is pushed. Swap
the 10k
 * resistor to the bottom of the schematic if you want it to favor sw1. The job of the 10K series resitor
is to prevent
 * a short circuit if pesky user pushes both buttons at once. It can be ommitted on a center-off slide or
toggle
 * where states are mutually exclusive.
 *
 * Schematic Diagram    ( can't belive I drew this funky ascii schematic )
 *
 *
 *                             +5 V
 *                                |
 *                                \
 *                                /
 *                                \    10K
 *                                /
 *                                \
 *                                |
 *                               /    switch 1  or 1/2 of center-off toggle or slide switch
 *                              /
 *                                |
 *            digital pin ________+_____________/\/\/\____________   ground
 *                                |
 *                                |               200K to 1M  (not critical)
 *                               /
 *                              /        switch 2 or 1/2 of center-off toggle or slide switch
 *                                |
 *                                |
 *                              _____
 *                               ___     ground
 *                                _
 *
 */


#define swPin 2                 // pin for input  - note: no semicolon after #define
int stateA, stateB;             // variables to store pin states
int sw1, sw2;                   // variables to represent switch states

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

void loop()
{
   digitalWrite(swPin, LOW);                  // make sure the puillup resistors are off
   stateA = digitalRead(swPin);
   digitalWrite(swPin, HIGH);                 // turn on the puillup resistors
   stateB = digitalRead(swPin);

   if ( stateA == 1 && stateB == 1 ){          // both states HIGH - switch 1 must be pushed
      sw1 = 1;
      sw2 = 0;
   }
   else if ( stateA == 0 && stateB == 0 ){     // both states LOW - switch 2 must be pushed
      sw1 = 0;
      sw2 = 1;
   }
   else{                                       // stateA HIGH and stateB LOW
      sw1 = 0;                                  // no switches pushed - or center-off toggle in middle position
      sw2 = 0;
   }

   Serial.print(sw1);
   Serial.print("    ");     // pad some spaces to format print output
   Serial.println(sw2);

   delay(100);
}
```

Restore

Edit Page | Page History | Printable View | All Recent Site Changes

## Arduino : Tutorial / Two Switches One Pin

# Read Two Switches With One I/O Pin

There are handy 20K pullup resistors (resistors connected internally between Arduino I/O pins and VCC - +5 volts in the Arduino's case) built into the Atmega chip upon which Freeduino's are based. They are accessible from software by using the digitalWrite() function, when the pin is set to an input.

This sketch exploits the pullup resistors under software control. The idea is that an external 200K resistor to ground will cause an input pin to report LOW when the internal (20K) pullup resistor is turned off. When the internal pullup resistor is turned on however, it will overwhelm the external 200K resistor and the pin will report HIGH.

One downside of the scheme (there always has to be a downside doesn't there?) is that one can't tell if both buttons are pushed at the same time. In this case the scheme just reports that sw2 is pushed. The job of the 10K series resistor, incidentally, is to prevent a short circuit if a pesky user pushes both buttons at once. It can be omitted on a center-off slide or toggle switch where the states are mutually exclusive.

```
/*
 * Read_Two_Switches_On_One_Pin
 * Read two pushbutton switches or one center-off toggle switch with one Arduino pin
 * Paul Badger 2008
 * From an idea in EDN (Electronic Design News)
 *
 * Exploits the pullup resistors available on each I/O and analog pin
 * The idea is that the 200K resistor to ground will cause the input pin to report LOW when the
 * (20K) pullup resistor is turned off, but when the pullup resistor is turned on,
 * it will overwhelm the 200K resistor and the pin will report HIGH.
 *
 * Schematic Diagram    ( can't belive I drew this funky ascii schematic )
 *
 *
 *                              +5 V
 *                                 |
 *                                 \
 *                                 /
 *                                 \    10K
 *                                 /
 *                                 \
 *                                 |
 *                                /    switch 1  or 1/2 of center-off toggle or slide switch
 *                               /
 *                                 |
 *            digital pin _________+____________/\/\/\____________   ground
 *                                 |
 *                                 |               200K to 1M  (not critical)
 *                                /
 *                               /        switch 2 or 1/2 of center-off toggle or slide switch
 *                                 |
 *                                 |
 *                               _____
 *                                ___     ground
 *                                 _
 *
 */


#define swPin 2                  // pin for input  - note: no semicolon after #define
int stateA, stateB;              // variables to store pin states
int sw1, sw2;                    // variables to represent switch states

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

void loop()
{
   digitalWrite(swPin, LOW);                    // make sure the puillup resistors are off
   stateA = digitalRead(swPin);
   digitalWrite(swPin, HIGH);                  // turn on the puillup resistors
   stateB = digitalRead(swPin);

   if ( stateA == 1 && stateB == 1 ){          // both states HIGH - switch 1 must be pushed
      sw1 = 1;
      sw2 = 0;
   }
   else if ( stateA == 0 && stateB == 0 ){     // both states LOW - switch 2 must be pushed
      sw1 = 0;
      sw2 = 1;
   }
   else{                                       // stateA HIGH and stateB LOW
      sw1 = 0;                                 // no switches pushed - or center-off toggle in
middle position
      sw2 = 0;
   }

   Serial.print(sw1);
   Serial.print("    ");     // pad some spaces to format print output
   Serial.println(sw2);

   delay(100);
}
```

## Tutorial.TiltSensor History

Hide minor edits - Show changes to markup

June 12, 2007, at 08:53 AM by David A. Mellis -
Changed line 17 from:

Use the *'Digital > Button* example to read the tilt-sensor, but you'll need to make sure that the inputPin variable in the code matches the digital pin you're using on the Arduino board.

to:

Use the **Digital > Button** example to read the tilt-sensor, but you'll need to make sure that the inputPin variable in the code matches the digital pin you're using on the Arduino board.

Restore
June 12, 2007, at 08:53 AM by David A. Mellis -
Added lines 9-10:

### Circuit

Changed lines 15-44 from:

```
/* Tilt Sensor
 * -----------
 *
 * Detects if the sensor has been tilted or not and
 * lights up the LED if so. Note that due to the
 * use of active low inputs (through a pull-up resistor)
 * the input is at low when the sensor is active.
 *
 * (cleft) David Cuartielles for DojoCorp and K3
 * @author: D. Cuartielles
 *
 */

int ledPin = 13;
int inPin = 7;
int value = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);              // initializes digital pin 13 as output
  pinMode(inPin, INPUT);                // initializes digital pin 7 as input
}

void loop()
{
  value = digitalRead(inPin);   // reads the value at a digital input
  digitalWrite(ledPin, value);
}
```

to:

### Code

Use the *'Digital > Button* example to read the tilt-sensor, but you'll need to make sure that the inputPin variable in the code matches the digital pin you're using on the Arduino board.

Restore
December 01, 2006, at 04:57 AM by David Cuartielles -
Changed lines 7-8 from:

The prototyping board has been populated with a 1K resitor to make the pull-up and the sensor itself. We have chosen the tilt sensor from Assemtech, which datasheet can be found here. The hardware was mounted and photographed by Anders Gran, the software comes from the basic Arduino examples.

to:

The prototyping board has been populated with a 1K resitor to make the pull-up and the sensor itself. We have chosen the tilt sensor from Assemtech, which datasheet can be found here. The hardware was mounted and photographed by Anders Gran, the software comes from the basic Arduino examples.

Changed lines 11-12 from:

*Picture of a protoboard supporting the tilt sensor, by Anders Gran*

to:

*Picture of a protoboard supporting the tilt sensor, by Anders Gran*

Restore
January 04, 2006, at 06:27 AM by 193.222.246.39 -
Added lines 1-42:

## Tilt Sensor

The tilt sensor is a component that can detect the tilting of an object. However it is only the equivalent to a pushbutton activated through a different physical mechanism. This type of sensor is the environmental-friendly version of a mercury-switch. It contains a metallic ball inside that will commute the two pins of the device from on to off and viceversa if the sensor reaches a certain angle.

The code example is exactly as the one we would use for a pushbutton but substituting this one with the tilt sensor. We use a pull-up resistor (thus use active-low to activate the pins) and connect the sensor to a digital input pin that we will read when needed.

The prototyping board has been populated with a 1K resitor to make the pull-up and the sensor itself. We have chosen the tilt sensor from Assemtech, which datasheet can be found here. The hardware was mounted and photographed by Anders Gran, the software comes from the basic Arduino examples.

http://static.flickr.com/30/65458903_d9a89442a9.jpg

*Picture of a protoboard supporting the tilt sensor, by Anders Gran*

```
/* Tilt Sensor
 * -----------
 *
 * Detects if the sensor has been tilted or not and
 * lights up the LED if so. Note that due to the
 * use of active low inputs (through a pull-up resistor)
 * the input is at low when the sensor is active.
 *
 * (cleft) David Cuartielles for DojoCorp and K3
 * @author: D. Cuartielles
 *
 */

int ledPin = 13;
int inPin = 7;
int value = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);              // initializes digital pin 13 as output
  pinMode(inPin, INPUT);                // initializes digital pin 7 as input
}

void loop()
{
  value = digitalRead(inPin);   // reads the value at a digital input
  digitalWrite(ledPin, value);
}
```

Restore

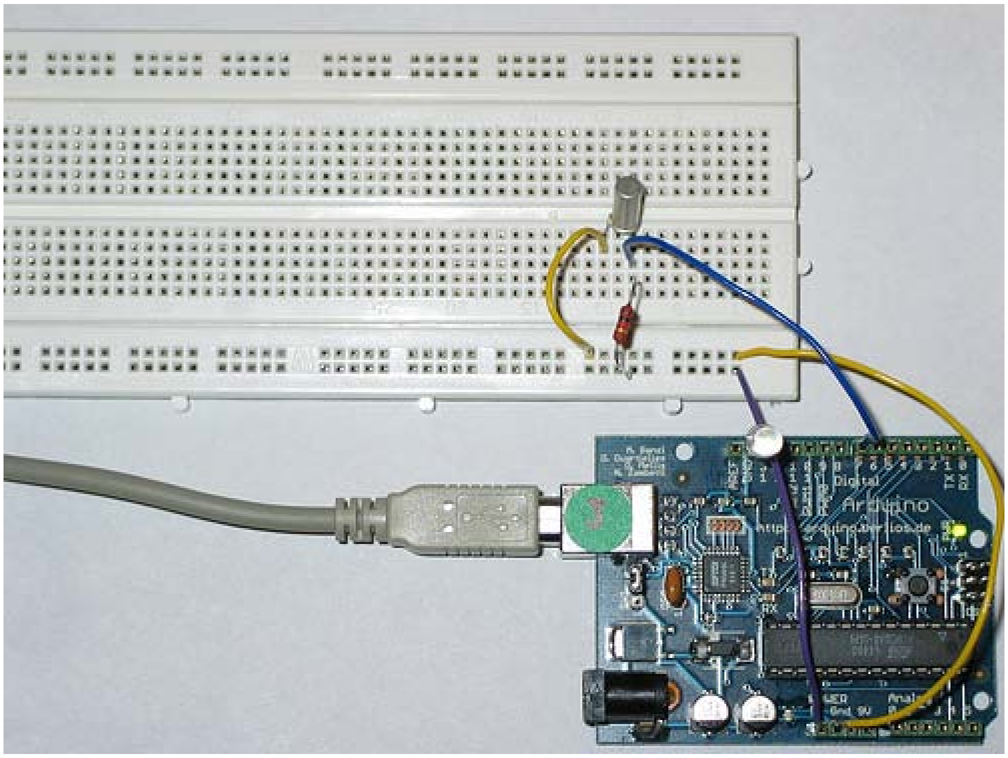
# Tilt Sensor

The tilt sensor is a component that can detect the tilting of an object. However it is only the equivalent to a pushbutton activated through a different physical mechanism. This type of sensor is the environmental-friendly version of a mercury-switch. It contains a metallic ball inside that will commute the two pins of the device from on to off and viceversa if the sensor reaches a certain angle.

The code example is exactly as the one we would use for a pushbutton but substituting this one with the tilt sensor. We use a pull-up resistor (thus use active-low to activate the pins) and connect the sensor to a digital input pin that we will read when needed.

The prototyping board has been populated with a 1K resitor to make the pull-up and the sensor itself. We have chosen the tilt sensor from Assemtech, which datasheet can be found here. The hardware was mounted and photographed by Anders Gran, the software comes from the basic Arduino examples.

## Circuit



*Picture of a protoboard supporting the tilt sensor, by Anders Gran*

## Code

Use the **Digital > Button** example to read the tilt-sensor, but you'll need to make sure that the inputPin variable in the code matches the digital pin you're using on the Arduino board.

# Tutorial.ControleLEDcircleWithJoystick History

Hide minor edits - Show changes to markup

February 03, 2006, at 10:31 AM by 193.49.124.107 -
Added lines 1-163:

## Controlling a circle of LEDs with a Joystick

### The whole circuit:

http://static.flickr.com/35/94946013_ba47fe116e.jpg

### Detail of the LED wiring

http://static.flickr.com/39/94946015_aaab0281e8.jpg

### Detail of the arduino wiring

http://static.flickr.com/42/94946020_2a1dd30b97.jpg

### How this works

As you know from the **Interfacing a Joystick** tutorial, the joystick gives a coordinate (x,y) back to arduino. As you can see looking to the joystick is that the space in which he moves is a circle. This circle will be from now on our 'Pie' (see bottom right of the first image).

The only thing we need now to understand is that we have divided our Pie in 8 pieces. To each piece will correspond an LED. (See figure below). This way, when the joystick gives us a coordinate, it will necesarilly belong to one of the pies. Then, the program always lights up the LED corresponding to the pie in which the joystick is.

http://static.flickr.com/19/94946024_f7fd4b55ec.jpg

### Code

```
/* Controle_LEDcirle_with_joystik
 * ------------
 * This program controles a cirle of 8 LEDs through a joystick
 *
 * First it reads two analog pins that are connected
 * to a joystick made of two potentiometers
 *
 * This input is interpreted as a coordinate (x,y)
 *
 * The program then calculates to which of the 8
 * possible zones belogns the coordinate (x,y)
 *
 * Finally it ligths up the LED which is placed in the
 * detected zone
 *
 * @authors: Cristina Hoffmann and Gustavo Jose Valera
 * @hardware: Cristina Hofmann and Gustavo Jose Valera
 * @context: Arduino Workshop at medialamadrid
 */

 // Declaration of Variables

 int ledPins [] = { 2,3,4,5,6,7,8,9 };    // Array of 8 leds mounted in a circle
 int ledVerde = 13;
```

```
  int espera = 40;                // Time you should wait for turning on the leds
  int joyPin1 = 0;                // slider variable connecetd to analog pin 0
  int joyPin2 = 1;                // slider variable connecetd to analog pin 1
  int coordX = 0;                 // variable to read the value from the analog pin 0
  int coordY = 0;                 // variable to read the value from the analog pin 1
  int centerX = 500;              // we measured the value for the center of the joystick
  int centerY = 500;
  int actualZone = 0;
  int previousZone = 0;

  // Asignment of the pins
  void setup()
  {
    int i;
    beginSerial(9600);
    pinMode (ledVerde, OUTPUT);
    for (i=0; i< 8; i++)
    {
      pinMode(ledPins[i], OUTPUT);
    }
  }

  // function that calculates the slope of the line that passes through the points
  // x1, y1 and x2, y2
  int calculateSlope(int x1, int y1, int x2, int y2)
  {
    return ((y1-y2) / (x1-x2));
  }

  // function that calculates in which of the 8 possible zones is the coordinate x y, given the center cx,
cy
  int calculateZone (int x, int y, int cx, int cy)
  {
    int alpha = calculateSlope(x,y, cx,cy); // slope of the segment betweent the point and the center

    if (x > cx)
    {
      if (y > cy) // first cuadrant
      {
        if (alpha > 1) // The slope is > 1, thus higher part of the first quadrant
          return 0;
        else
          return 1;    // Otherwise the point is in the lower part of the first quadrant
      }
      else // second cuadrant
      {
        if (alpha > -1)
          return 2;
        else
          return 3;
      }
    }

    else
    {
      if (y < cy) // third cuadrant
      {
        if (alpha > 1)
          return 4;
        else
          return 5;
      }
      else // fourth cuadrant
      {
        if (alpha > -1)
          return 6;
        else
          return 7;
      }
    }
  }

   void loop() {
    digitalWrite(ledVerde, HIGH); // flag to know we entered the loop, you can erase this if you want

    // reads the value of the variable resistors
    coordX = analogRead(joyPin1);
    coordY = analogRead(joyPin2);

    // We calculate in which x
    actualZone = calculateZone(coordX, coordY, centerX, centerY);

    digitalWrite (ledPins[actualZone], HIGH);

    if (actualZone != previousZone)
      digitalWrite (ledPins[previousZone], LOW);

   // we print int the terminal, the cartesian value of the coordinate, and the zone where it belongs.
  //This is not necesary for a standalone version
    serialWrite('C');
    serialWrite(32); // print space
    printInteger(coordX);
    serialWrite(32); // print space
    printInteger(coordY);
    serialWrite(10);
    serialWrite(13);

    serialWrite('Z');
    serialWrite(32); // print space
    printInteger(actualZone);
    serialWrite(10);
    serialWrite(13);

  // But this is necesary so, don't delete it!
    previousZone = actualZone;
   // delay (500);

 }
```

@idea: Cristina Hoffmann and Gustavo Jose Valera

@code: Cristina Hoffmann and Gustavo Jose Valera

@pictures and graphics: Cristina Hoffmann

@date: 20051008 - Madrid - Spain

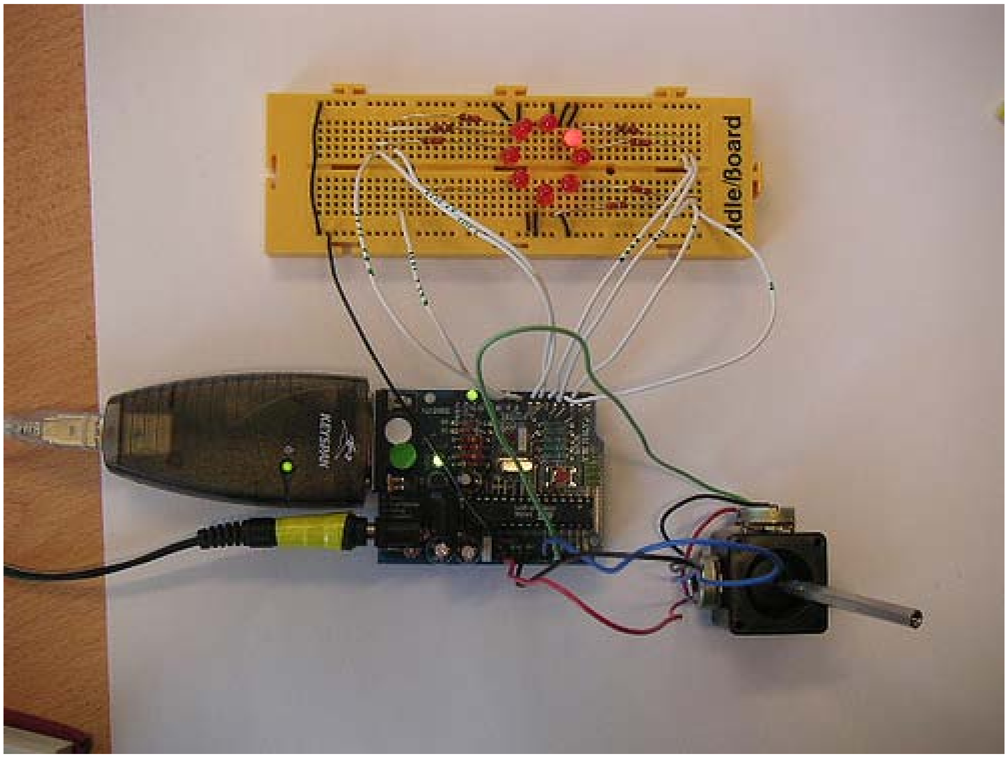
Restore

## Arduino : Tutorial / Controle LE Dcircle With Joystick

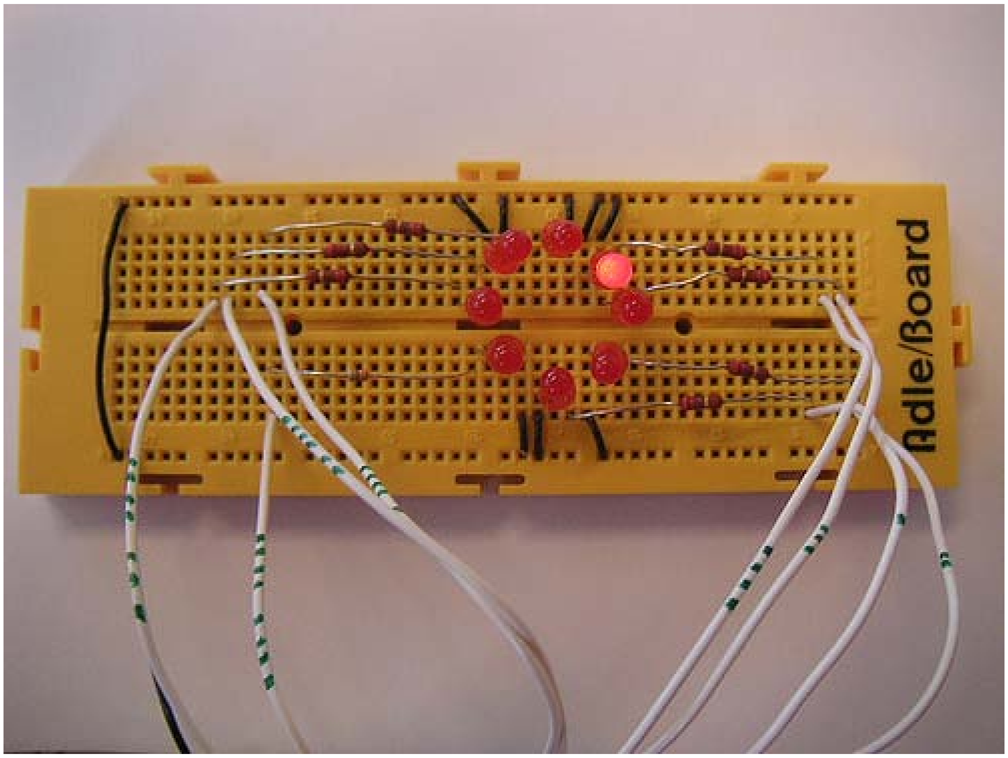
**Learning** Examples | Foundations | Hacking | Links

# Controlling a circle of LEDs with a Joystick

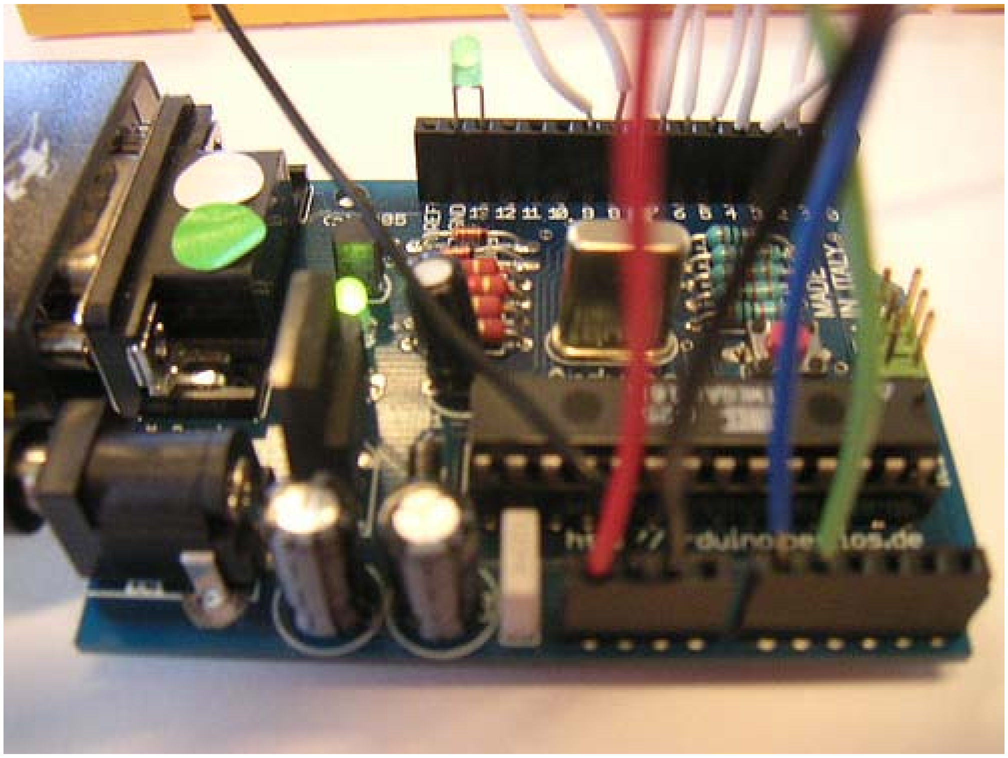
### The whole circuit:



### Detail of the LED wiring



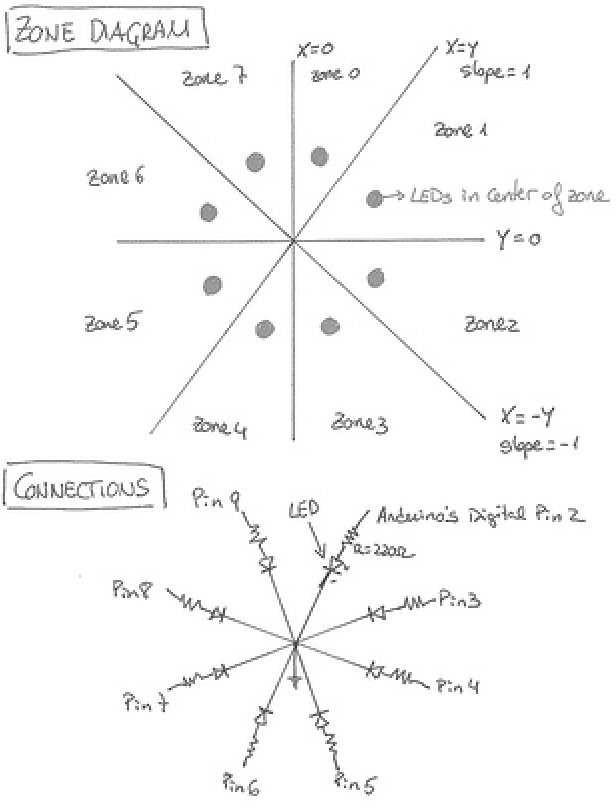
## Detail of the arduino wiring



## How this works

As you know from the **Interfacing a Joystick** tutorial, the joystick gives a coordinate (x,y) back to arduino. As you can see looking to the joystick is that the space in which he moves is a circle. This circle will be from now on our 'Pie' (see bottom right of the first image).

The only thing we need now to understand is that we have divided our Pie in 8 pieces. To each piece will correspond an LED. (See figure below). This way, when the joystick gives us a coordinate, it will necesarilly belong to one of the pies. Then, the program always lights up the LED corresponding to the pie in which the joystick is.



DRAWING BY CRISTINA HOFFMANN

## Code

```
/* Controle_LEDcirle_with_joystik
 * ------------
 * This program controles a cirle of 8 LEDs through a joystick
 *
 * First it reads two analog pins that are connected
 * to a joystick made of two potentiometers
 *
 * This input is interpreted as a coordinate (x,y)
 *
 * The program then calculates to which of the 8
 * possible zones belogns the coordinate (x,y)
 *
 * Finally it ligths up the LED which is placed in the
 * detected zone
 *
 * @authors: Cristina Hoffmann and Gustavo Jose Valera
 * @hardware: Cristina Hofmann and Gustavo Jose Valera
 * @context: Arduino Workshop at medialamadrid
 */

 // Declaration of Variables

 int ledPins [] = { 2,3,4,5,6,7,8,9 };    // Array of 8 leds mounted in a circle
 int ledVerde = 13;
 int espera = 40;                 // Time you should wait for turning on the leds
 int joyPin1 = 0;                 // slider variable connecetd to analog pin 0
 int joyPin2 = 1;                 // slider variable connecetd to analog pin 1
 int coordX = 0;                  // variable to read the value from the analog pin 0
 int coordY = 0;                  // variable to read the value from the analog pin 1
 int centerX = 500;               // we measured the value for the center of the joystick
 int centerY = 500;
 int actualZone = 0;
 int previousZone = 0;
```

```
  // Asignment of the pins
  void setup()
  {
    int i;
    beginSerial(9600);
    pinMode (ledVerde, OUTPUT);
    for (i=0; i< 8; i++)
    {
      pinMode(ledPins[i], OUTPUT);
    }
  }

  // function that calculates the slope of the line that passes through the points
  // x1, y1 and x2, y2
  int calculateSlope(int x1, int y1, int x2, int y2)
  {
    return ((y1-y2) / (x1-x2));
  }

  // function that calculates in which of the 8 possible zones is the coordinate x y, given the
center cx, cy
  int calculateZone (int x, int y, int cx, int cy)
  {
    int alpha = calculateSlope(x,y, cx,cy); // slope of the segment betweent the point and the
center

    if (x > cx)
    {
      if (y > cy) // first cuadrant
      {
        if (alpha > 1) // The slope is > 1, thus higher part of the first quadrant
          return 0;
        else
          return 1;    // Otherwise the point is in the lower part of the first quadrant
      }
      else // second cuadrant
      {
        if (alpha > -1)
          return 2;
        else
          return 3;
      }
    }

    else
    {
      if (y < cy) // third cuadrant
      {
        if (alpha > 1)
          return 4;
        else
          return 5;
      }
      else // fourth cuadrant
      {
        if (alpha > -1)
          return 6;
        else
          return 7;
      }
    }
  }

   void loop() {
    digitalWrite(ledVerde, HIGH); // flag to know we entered the loop, you can erase this if you
want

    // reads the value of the variable resistors
    coordX = analogRead(joyPin1);
    coordY = analogRead(joyPin2);

    // We calculate in which x
    actualZone = calculateZone(coordX, coordY, centerX, centerY);

    digitalWrite (ledPins[actualZone], HIGH);
    if (actualZone != previousZone)
      digitalWrite (ledPins[previousZone], LOW);

   // we print int the terminal, the cartesian value of the coordinate, and the zone where it
belongs.
  //This is not necesary for a standalone version
    serialWrite('C');
    serialWrite(32); // print space
    printInteger(coordX);
    serialWrite(32); // print space
    printInteger(coordY);
    serialWrite(10);
    serialWrite(13);

    serialWrite('Z');
    serialWrite(32); // print space
    printInteger(actualZone);
    serialWrite(10);
    serialWrite(13);

  // But this is necesary so, don't delete it!
    previousZone = actualZone;
   // delay (500);

 }
```

@idea: Cristina Hoffmann and Gustavo Jose Valera

@code: Cristina Hoffmann and Gustavo Jose Valera

@pictures and graphics: Cristina Hoffmann

@date: 20051008 - Madrid - Spain

## Tutorial.LEDColorMixerWith3Potentiometers History

Hide minor edits - Show changes to markup

September 07, 2006, at 07:57 AM by Clay Shirky - Fixed sensitivity bug
Changed lines 73-74 from:

```
    if ( abs(checkSum - prevCheckSum) <= sens )   // If old and new values are
                                                 // within "sens" points...
```

to:

```
    if ( abs(checkSum - prevCheckSum) > sens )   // If old and new values differ
                                                 // above sensitivity threshold
```

Restore
September 06, 2006, at 08:02 PM by Clay Shirky -
Changed line 74 from:

```
                                                 // within sens points...
```

to:

```
                                                 // within "sens" points...
```

Restore
September 06, 2006, at 08:00 PM by Clay Shirky -
Changed lines 8-10 from:

- If the LEDs are different colors, and are directed at a diffusing surface (stuck in a
- Ping-Pong ball, or placed in a paper coffee cup with a cut-out bottom and a white plastic lid),
- the colors will mix together.

to:

- If the LEDs are different colors, and are directed at diffusing surface (stuck in a
- a Ping-Pong ball, or placed in a paper coffee cup with a cut-out bottom and
- a white plastic lid), the colors will mix together.

Restore
September 06, 2006, at 07:59 PM by Clay Shirky -
Changed lines 8-9 from:

- If the LEDs are different colors, and are behind a diffusing surface (anything
- from a Ping-Pong ball to a coffee cup with a cut-out bottom and a white plastic lid),

to:

- If the LEDs are different colors, and are directed at a diffusing surface (stuck in a
- Ping-Pong ball, or placed in a paper coffee cup with a cut-out bottom and a white plastic lid),

Restore
September 06, 2006, at 07:58 PM by Clay Shirky -
Changed line 1 from:

[=

to:

```
 [=
```

Added line 105:

=]

Restore
September 06, 2006, at 07:56 PM by Clay Shirky -
Added line 1:

[=

Changed line 3 from:

- Espresso-cup Color Mixer:

to:

- "Coffee-cup" Color Mixer:

Changed lines 9-10 from:

- from a Ping-Pong ball to a coffee cup with a white plastic lid), the colors
- will mix together.

to:

- from a Ping-Pong ball to a coffee cup with a cut-out bottom and a white plastic lid),
- the colors will mix together.

Restore
September 06, 2006, at 07:53 PM by Clay Shirky -
Added lines 1-103:

/*

- Espresso-cup Color Mixer:
- Code for mixing and reporting PWM-mediated color
- Assumes Arduino 0004 or higher, as it uses Serial.begin()-style communication
-
- Control 3 LEDs with 3 potentiometers
- If the LEDs are different colors, and are behind a diffusing surface (anything
- from a Ping-Pong ball to a coffee cup with a white plastic lid), the colors
- will mix together.
-
- When you mix a color you like, stop adjusting the pots.
- The mix values that create that color will be reported via serial out.
-
- Standard colors for light mixing are Red, Green, and Blue, though you can mix
- with any three colors; Red + Blue + White would let you mix shades of red,
- blue, and purple (though no yellow, orange, green, or blue-green.)
-
- Put 220 Ohm resistors in line with pots, to prevent circuit from
- grounding out when the pots are at zero
- /

// Analog pin settings int aIn = 0; // Potentiometers connected to analog pins 0, 1, and 2 int bIn = 1; // (Connect power to 5V and ground to analog ground) int cIn = 2;

// Digital pin settings int aOut = 9; // LEDs connected to digital pins 9, 10 and 11 int bOut = 10; // (Connect cathodes to digital ground) int cOut = 11;

// Values int aVal = 0; // Variables to store the input from the potentiometers int bVal = 0; int cVal = 0;

// Variables for comparing values between loops int i = 0; // Loop counter int wait = (1000); // Delay between most recent pot adjustment and output

int checkSum = 0; // Aggregate pot values int prevCheckSum = 0; int sens = 3; // Sensitivity theshold, to prevent small changes in

```
                         // pot values from triggering false reporting
```

// FLAGS int PRINT = 1; // Set to 1 to output values int DEBUG = 1; // Set to 1 to turn on debugging output

```
void setup() {

  pinMode(aOut, OUTPUT);   // sets the digital pins as output
  pinMode(bOut, OUTPUT);
  pinMode(cOut, OUTPUT);
  Serial.begin(9600);     // Open serial communication for reporting

}

void loop() {

  i += 1; // Count loop

  aVal = analogRead(aIn) / 4;  // read input pins, convert to 0-255 scale
  bVal = analogRead(bIn) / 4;
  cVal = analogRead(cIn) / 4;

  analogWrite(aOut, aVal);    // Send new values to LEDs
  analogWrite(bOut, bVal);
  analogWrite(cOut, cVal);

  if (i % wait == 0)                 // If enough time has passed...
  {
    checkSum = aVal+bVal+cVal;      // ...add up the 3 values.
    if ( abs(checkSum - prevCheckSum) <= sens )   // If old and new values are
                                                 // within sens points...
    {
      if (PRINT)                    // ...and if the PRINT flag is set...
      {
        Serial.print("A: ");        // ...then print the values.
        Serial.print(aVal);
        Serial.print("\t");
        Serial.print("B: ");
        Serial.print(bVal);
        Serial.print("\t");
        Serial.print("C: ");
        Serial.println(cVal);
        PRINT = 0;
      }
    }
    else
    {
      PRINT = 1;  // Re-set the flag
    }
    prevCheckSum = checkSum;  // Update the values

    if (DEBUG)   // If we want debugging output as well...
    {
      Serial.print(checkSum);
      Serial.print("<=>");
      Serial.print(prevCheckSum);
      Serial.print("\tPrint: ");
      Serial.println(PRINT);
    }
  }

}
```

Restore

# Arduino : Tutorial / LED Color Mixer With 3 Potentiometers

```
/*
* "Coffee-cup" Color Mixer:
* Code for mixing and reporting PWM-mediated color
* Assumes Arduino 0004 or higher, as it uses Serial.begin()-style communication
*
* Control 3 LEDs with 3 potentiometers
* If the LEDs are different colors, and are directed at diffusing surface (stuck in a
*   a Ping-Pong ball, or placed in a paper coffee cup with a cut-out bottom and
*   a white plastic lid), the colors will mix together.
*
* When you mix a color you like, stop adjusting the pots.
* The mix values that create that color will be reported via serial out.
*
* Standard colors for light mixing are Red, Green, and Blue, though you can mix
*   with any three colors; Red + Blue + White would let you mix shades of red,
*   blue, and purple (though no yellow, orange, green, or blue-green.)
*
* Put 220 Ohm resistors in line with pots, to prevent circuit from
*   grounding out when the pots are at zero
*/

// Analog pin settings
int aIn = 0;    // Potentiometers connected to analog pins 0, 1, and 2
int bIn = 1;    //   (Connect power to 5V and ground to analog ground)
int cIn = 2;

// Digital pin settings
int aOut = 9;   // LEDs connected to digital pins 9, 10 and 11
int bOut = 10;  //   (Connect cathodes to digital ground)
int cOut = 11;

// Values
int aVal = 0;   // Variables to store the input from the potentiometers
int bVal = 0;
int cVal = 0;

// Variables for comparing values between loops
int i = 0;            // Loop counter
int wait = (1000);    // Delay between most recent pot adjustment and output

int checkSum     = 0; // Aggregate pot values
int prevCheckSum = 0;
int sens         = 3; // Sensitivity theshold, to prevent small changes in
                      // pot values from triggering false reporting
// FLAGS
int PRINT = 1; // Set to 1 to output values
int DEBUG = 1; // Set to 1 to turn on debugging output

void setup()
{
  pinMode(aOut, OUTPUT);   // sets the digital pins as output
  pinMode(bOut, OUTPUT);
  pinMode(cOut, OUTPUT);
  Serial.begin(9600);     // Open serial communication for reporting
}

void loop()
{
  i += 1; // Count loop

  aVal = analogRead(aIn) / 4;  // read input pins, convert to 0-255 scale
  bVal = analogRead(bIn) / 4;
  cVal = analogRead(cIn) / 4;

  analogWrite(aOut, aVal);    // Send new values to LEDs
  analogWrite(bOut, bVal);
  analogWrite(cOut, cVal);

  if (i % wait == 0)                // If enough time has passed...
  {
    checkSum = aVal+bVal+cVal;      // ...add up the 3 values.
    if ( abs(checkSum - prevCheckSum) > sens )   // If old and new values differ
                                                  // above sensitivity threshold
    {
      if (PRINT)                    // ...and if the PRINT flag is set...
      {
        Serial.print("A: ");        // ...then print the values.
        Serial.print(aVal);
        Serial.print("\t");
        Serial.print("B: ");
        Serial.print(bVal);
        Serial.print("\t");
        Serial.print("C: ");
        Serial.println(cVal);
        PRINT = 0;
      }
    }
    else
    {
      PRINT = 1;  // Re-set the flag
    }
    prevCheckSum = checkSum;  // Update the values

    if (DEBUG)   // If we want debugging output as well...
    {
      Serial.print(checkSum);
      Serial.print("<=>");
      Serial.print(prevCheckSum);
      Serial.print("\tPrint: ");
      Serial.println(PRINT);
    }
  }
}
```

# Tutorial.Stopwatch History

Hide minor edits - Show changes to markup

April 21, 2008, at 09:49 PM by Paul Badger -
Changed lines 47-48 from:

```
   buttonState = digitalRead(buttonPin);                        // read the button state and store
```

to:

```
   buttonState = digitalRead(buttonPin);                 // read the button state and store
```

Restore
April 21, 2008, at 09:48 PM by Paul Badger -
Changed lines 46-48 from:

```
   // here is where you'd put code that needs to be running all the time.

   // check for button press
```

to:

```
    // check for button press
```

Restore
April 21, 2008, at 09:47 PM by Paul Badger -
Changed line 64 from:

```
      elapsedTime =   (millis() - startTime) - 5;            // store elapsed time (-5 to make up for
switch debounce time)
```

to:

```
      elapsedTime =   millis() - startTime;                // store elapsed time
```

Restore
April 21, 2008, at 09:45 PM by Paul Badger -
Changed lines 70-71 from:

```
         // routine to report elapsed time
```

to:

```
         // routine to report elapsed time - this breaks when delays are in single or double digits. Fix
this as a coding exercise.
```

Restore
April 18, 2008, at 07:25 AM by Paul Badger -
Changed lines 10-11 from:

```
 * Demonstrates using millis(), pullup resistors, making two things happen at once, printing fractions
```

to:

```
 * Demonstrates using millis(), pullup resistors,
 * making two things happen at once, printing fractions
```

Restore
April 18, 2008, at 07:24 AM by Paul Badger -
Added lines 1-105:

## Stopwatch

A sketch that demonstrates how to do two (or more) things at once by using millis().

```
/* StopWatch
 * Paul Badger 2008
 * Demonstrates using millis(), pullup resistors, making two things happen at once, printing fractions
 *
 * Physical setup: momentary switch connected to pin 4, other side connected to ground
 * LED with series resistor between pin 13 and ground
 */


#define ledPin  13                  // LED connected to digital pin 13
#define buttonPin 4                 // button on pin 4

int value = LOW;                    // previous value of the LED
int buttonState;                    // variable to store button state
int lastButtonState;                // variable to store last button state
int blinking;                       // condition for blinking - timer is timing
long interval = 100;                // blink interval - change to suit
long previousMillis = 0;            // variable to store last time LED was updated
long startTime ;                    // start time for stop watch
long elapsedTime ;                  // elapsed time for stop watch
int fractional;                     // variable used to store fractional part of time



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

   pinMode(ledPin, OUTPUT);         // sets the digital pin as output

   pinMode(buttonPin, INPUT);       // not really necessary, pins default to INPUT anyway
   digitalWrite(buttonPin, HIGH);   // turn on pullup resistors. Wire button so that press shorts pin to
ground.

}

void loop()
{
   // here is where you'd put code that needs to be running all the time.

   // check for button press
   buttonState = digitalRead(buttonPin);                        // read the button state and store

   if (buttonState == LOW && lastButtonState == HIGH  &&  blinking == false){     // check for a high to
low transition
      // if true then found a new button press while clock is not running - start the clock

      startTime = millis();                                  // store the start time
      blinking = true;                                       // turn on blinking while timing
      delay(5);                                              // short delay to debounce switch
      lastButtonState = buttonState;                         // store buttonState in lastButtonState, to
compare next time

   }

   else if (buttonState == LOW && lastButtonState == HIGH && blinking == true){     // check for a high to
low transition
      // if true then found a new button press while clock is running - stop the clock and report

      elapsedTime =   (millis() - startTime) - 5;            // store elapsed time (-5 to make up for
switch debounce time)
      blinking = false;                                       // turn off blinking, all done timing
      lastButtonState = buttonState;                          // store buttonState in lastButtonState, to
compare next time



         // routine to report elapsed time

         Serial.print( (int)(elapsedTime / 1000L) );         // divide by 1000 to convert to seconds - then
cast to an int to print
      Serial.print(".");                                     // print decimal point
      fractional = (int)(elapsedTime % 1000L);               // use modulo operator to get fractional part
of time
      Serial.println(fractional);                            // print fractional part of time

   }

   else{
      lastButtonState = buttonState;                         // store buttonState in lastButtonState, to
compare next time
   }

   // blink routine - blink the LED while timing
   // check to see if it's time to blink the LED; that is, is the difference
   // between the current time and last time we blinked the LED bigger than
   // the interval at which we want to blink the LED.

   if ( (millis() - previousMillis > interval) ) {

      if (blinking == true){
         previousMillis = millis();                          // remember the last time we blinked the LED

         // if the LED is off turn it on and vice-versa.
         if (value == LOW)
            value = HIGH;
         else
            value = LOW;
         digitalWrite(ledPin, value);
      }
      else{
         digitalWrite(ledPin, LOW);                          // turn off LED when not blinking
      }
   }

}
```

Restore

## Arduino : Tutorial / Stopwatch

**Learning** Examples | Foundations | Hacking | Links

# Stopwatch

A sketch that demonstrates how to do two (or more) things at once by using millis().

```
/* StopWatch
 * Paul Badger 2008
 * Demonstrates using millis(), pullup resistors,
 * making two things happen at once, printing fractions
 *
 * Physical setup: momentary switch connected to pin 4, other side connected to ground
 * LED with series resistor between pin 13 and ground
 */


#define ledPin  13                  // LED connected to digital pin 13
#define buttonPin 4                 // button on pin 4

int value = LOW;                    // previous value of the LED
int buttonState;                    // variable to store button state
int lastButtonState;                // variable to store last button state
int blinking;                       // condition for blinking - timer is timing
long interval = 100;                // blink interval - change to suit
long previousMillis = 0;            // variable to store last time LED was updated
long startTime ;                    // start time for stop watch
long elapsedTime ;                  // elapsed time for stop watch
int fractional;                     // variable used to store fractional part of time



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

   pinMode(ledPin, OUTPUT);         // sets the digital pin as output

   pinMode(buttonPin, INPUT);       // not really necessary, pins default to INPUT anyway
   digitalWrite(buttonPin, HIGH);   // turn on pullup resistors. Wire button so that press shorts
pin to ground.

}

void loop()
{
    // check for button press
   buttonState = digitalRead(buttonPin);                  // read the button state and store

   if (buttonState == LOW && lastButtonState == HIGH  &&  blinking == false){     // check for a
high to low transition
      // if true then found a new button press while clock is not running - start the clock

      startTime = millis();                                   // store the start time
      blinking = true;                                        // turn on blinking while timing
      delay(5);                                               // short delay to debounce switch
      lastButtonState = buttonState;                          // store buttonState in
lastButtonState, to compare next time

   }

   else if (buttonState == LOW && lastButtonState == HIGH && blinking == true){     // check for
a high to low transition
      // if true then found a new button press while clock is running - stop the clock and report

      elapsedTime =   millis() - startTime;                // store elapsed time
      blinking = false;                                       // turn off blinking, all done
timing
      lastButtonState = buttonState;                          // store buttonState in
lastButtonState, to compare next time



         // routine to report elapsed time - this breaks when delays are in single or double
digits. Fix this as a coding exercise.
```

```
         Serial.print( (int)(elapsedTime / 1000L) );         // divide by 1000 to convert to
seconds - then cast to an int to print
      Serial.print(".");                                     // print decimal point
      fractional = (int)(elapsedTime % 1000L);               // use modulo operator to get
fractional part of time
      Serial.println(fractional);                            // print fractional part of time

   }

   else{
      lastButtonState = buttonState;                         // store buttonState in
lastButtonState, to compare next time
   }

   // blink routine - blink the LED while timing
   // check to see if it's time to blink the LED; that is, is the difference
   // between the current time and last time we blinked the LED bigger than
   // the interval at which we want to blink the LED.

   if ( (millis() - previousMillis > interval) ) {

      if (blinking == true){
         previousMillis = millis();                          // remember the last time we blinked
the LED

         // if the LED is off turn it on and vice-versa.
         if (value == LOW)
            value = HIGH;
         else
            value = LOW;
         digitalWrite(ledPin, value);
      }
      else{
         digitalWrite(ledPin, LOW);                          // turn off LED when not blinking
      }
   }

}
```

## Tutorial.ADXL3xx History

Hide minor edits - Show changes to markup

July 02, 2008, at 02:07 PM by David A. Mellis -
Changed lines 1-2 from:

*Examples > Devices*

to:

*Examples > Analog I/O*

Restore
July 02, 2008, at 01:58 PM by David A. Mellis -
Changed lines 5-6 from:

Reads an Analog Devices ADXL3xx series (e.g. ADXL320, ADXL321, ADXL322, ADXL330) accelerometer and communicates the acceleration to the computer. The pins used are designed to be easily compatible with the breakout boards from Sparkfun. The acceleration on each axis is output as an analog voltage between 0 and 5 volts, which is read by an analog input on the Arduino.

to:

Reads an Analog Devices ADXL3xx series (e.g. ADXL320, ADXL321, ADXL322, ADXL330) accelerometer and communicates the acceleration to the computer. The pins used are designed to be easily compatible with the breakout boards from Sparkfun. The ADXL3xx outputs the acceleration on each axis as an analog voltage between 0 and 5 volts, which is read by an analog input on the Arduino.

Restore
July 02, 2008, at 01:57 PM by David A. Mellis -
Changed lines 5-6 from:

Reads an Analog Devices ADXL3xx series (e.g. ADXL320, ADXL321, ADXL322, ADXL330) accelerometer and communicates the acceleration to the computer. The pins used are designed to be easily compatible with the breakout boards from Sparkfun.

to:

Reads an Analog Devices ADXL3xx series (e.g. ADXL320, ADXL321, ADXL322, ADXL330) accelerometer and communicates the acceleration to the computer. The pins used are designed to be easily compatible with the breakout boards from Sparkfun. The acceleration on each axis is output as an analog voltage between 0 and 5 volts, which is read by an analog input on the Arduino.

Restore
July 02, 2008, at 01:52 PM by David A. Mellis -
Changed lines 62-63 from:

Here are some accelerometer readings from the y-axis of an ADXL322 2g accelerometer. Values should be the same for the other axes, but will vary based on the sensitivity of the device.

to:

Here are some accelerometer readings collected by the positioning the y-axis of an ADXL322 2g accelerometer at various angles from ground. Values should be the same for the other axes, but will vary based on the sensitivity of the device. With the axis horizontal (i.e. parallel to ground or 0°), the accelerometer reading should be around 512, but values at other angles will be different for a different accelerometer (e.g. the ADXL302 5g one).

Restore
July 02, 2008, at 01:49 PM by David A. Mellis -

Changed line 64 from:
to:
Restore
July 02, 2008, at 01:49 PM by David A. Mellis -
Changed lines 58-66 from:

@]

to:

@]

## Data

Here are some accelerometer readings from the y-axis of an ADXL322 2g accelerometer. Values should be the same for the other axes, but will vary based on the sensitivity of the device.

| **Angle** | -90 | -80 | -70 | -60 | -50 | -40 | -30 | -20 | -10 | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Acceleration** | 662 | 660 | 654 | 642 | 628 | 610 | 589 | 563 | 537 | 510 | 485 | 455 | 433 | 408 | 390 | 374 | 363 | 357 | 355 |

Restore
July 02, 2008, at 01:43 PM by David A. Mellis -
Changed line 21 from:
to:
Restore
July 02, 2008, at 01:43 PM by David A. Mellis -
Changed line 15 from:
to:
Restore
July 02, 2008, at 01:42 PM by David A. Mellis -
Added lines 13-14:

Pinout for the above configuration:

Added lines 19-20:

Or, if you're using just the accelerometer:

Restore
July 02, 2008, at 01:41 PM by David A. Mellis -
Changed line 13 from:
to:
Changed line 17 from:
to:
Restore
July 02, 2008, at 01:40 PM by David A. Mellis -
Changed line 13 from:
to:
Changed line 17 from:
to:
Restore
July 02, 2008, at 01:40 PM by David A. Mellis -
Changed line 17 from:
to:
Restore
July 02, 2008, at 01:40 PM by David A. Mellis -
Changed line 13 from:
to:
Restore
July 02, 2008, at 01:40 PM by David A. Mellis -
Added lines 11-12:

*An ADXL322 on a Sparkfun breakout board inserted into the analog input pins of an Arduino.*

Changed lines 14-21 from:

| **Accelerometer Pin** | **Arduino Pin** |
|---|---|
| Self-Test | Analog Input 0 |
| Z-Axis | Analog Input 1 |

| | |
|---|---|
| Y-Axis | Analog Input 2 |
| X-Axis | Analog Input 3 |
| Ground | Analog Input 4 |
| VDD | Analog Input 5 |

to:

| **Breakout Board Pin** | Self-Test | Z-Axis | Y-Axis | X-Axis | Ground | VDD |
|---|---|---|---|---|---|---|
| **Arduino Analog Input Pin** | 0 | 1 | 2 | 3 | 4 | 5 |

| **ADXL3xx Pin** | Self-Test | ZOut | YOut | XOut | Ground | VDD |
|---|---|---|---|---|---|---|
| **Arduino Pin** | *None* (unconnected) | Analog Input 1 | Analog Input 2 | Analog Input 3 | GND | 5V |

Restore

July 02, 2008, at 01:33 PM by David A. Mellis -
Added line 11:

Restore

July 02, 2008, at 01:33 PM by David A. Mellis -
Changed lines 11-18 from:

| **Accelerometer Pin** | **Arduino Pin** |
|---|---|
| Self-Test | Analog Input 0 |
| Z-Axis | Analog Input 1 |
| Y-Axis | Analog Input 2 |
| X-Axis | Analog Input 3 |
| Ground | Analog Input 4 |
| VDD | Analog Input 5 |

to:

| **Accelerometer Pin** | **Arduino Pin** |
|---|---|
| Self-Test | Analog Input 0 |
| Z-Axis | Analog Input 1 |
| Y-Axis | Analog Input 2 |
| X-Axis | Analog Input 3 |
| Ground | Analog Input 4 |
| VDD | Analog Input 5 |

Restore

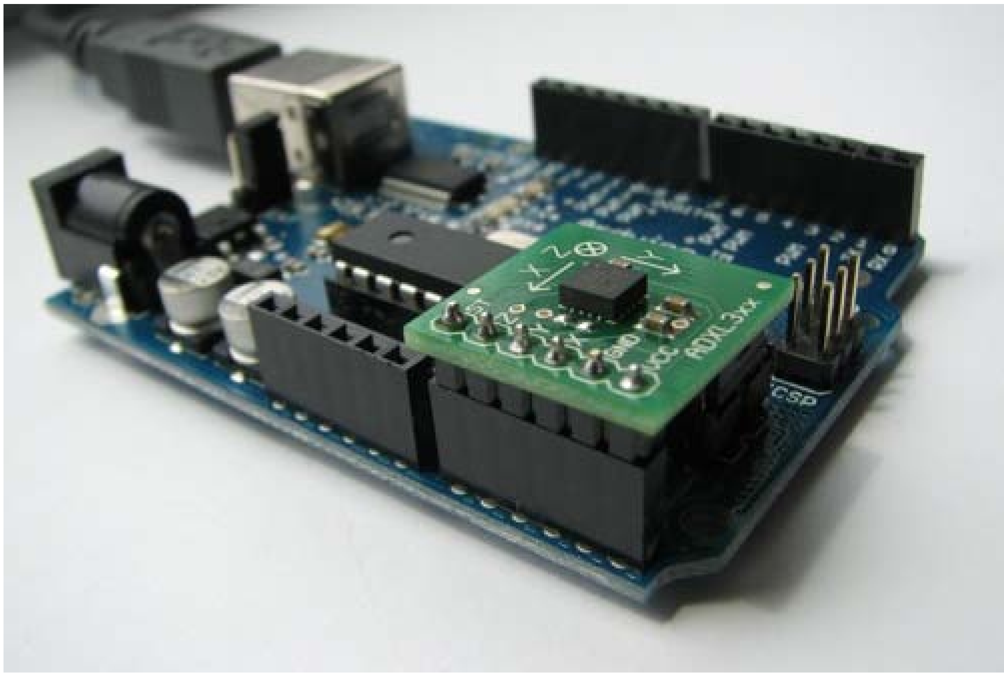
July 02, 2008, at 01:32 PM by David A. Mellis -
Added lines 1-52:

*Examples > Devices*

## ADXL3xx Accelerometer

Reads an Analog Devices ADXL3xx series (e.g. ADXL320, ADXL321, ADXL322, ADXL330) accelerometer and communicates the acceleration to the computer. The pins used are designed to be easily compatible with the breakout boards from Sparkfun.

### Circuit



| Accelerometer Pin | Arduino Pin |
|---|---|
| Self-Test | Analog Input 0 |
| Z-Axis | Analog Input 1 |
| Y-Axis | Analog Input 2 |
| X-Axis | Analog Input 3 |
| Ground | Analog Input 4 |
| VDD | Analog Input 5 |

## Code

```
int groundpin = 18;             // analog input pin 4
int powerpin = 19;              // analog input pin 5
int xpin = 3;                   // x-axis of the accelerometer
int ypin = 2;                   // y-axis
int zpin = 1;                   // z-axis (only on 3-axis models)

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

  // Provide ground and power by using the analog inputs as normal
  // digital pins.  This makes it possible to directly connect the
  // breakout board to the Arduino.  If you use the normal 5V and
  // GND pins on the Arduino, you can remove these lines.
  pinMode(groundPin, OUTPUT);
  pinMode(powerPin, OUTPUT);
  digitalWrite(groundPin, LOW);
  digitalWrite(powerPin, HIGH);
}

void loop()
{
  Serial.print(analogRead(xpin));
  Serial.print(" ");
  Serial.print(analogRead(ypin));
  Serial.print(" ");
  Serial.print(analogRead(zpin));
  Serial.println();
  delay(1000);
}
```

Restore

Edit Page | Page History | Printable View | All Recent Site Changes

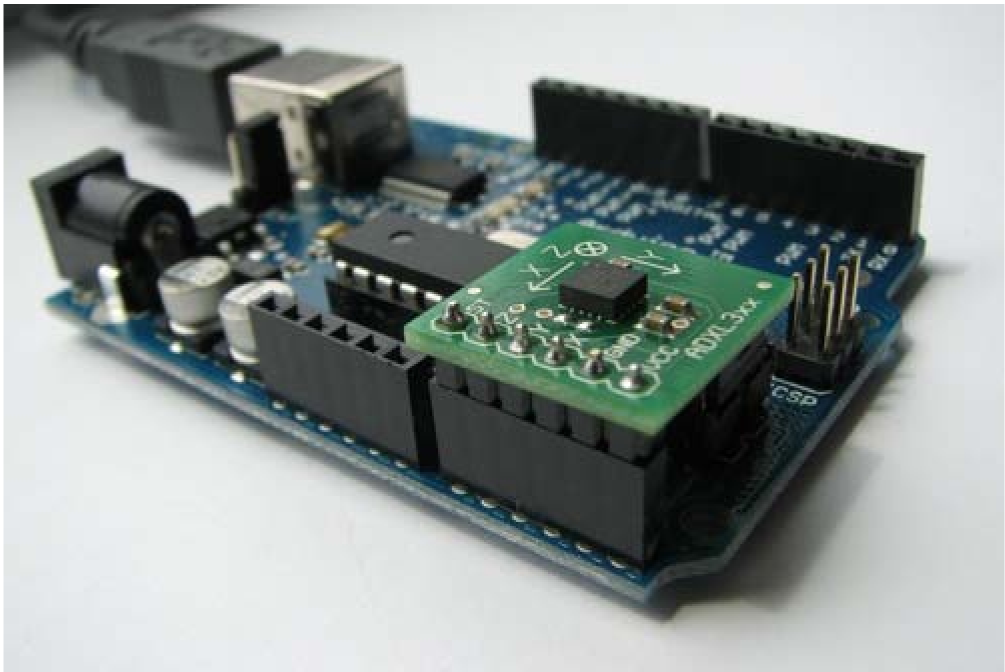
**Learning** Examples | Foundations | Hacking | Links

*Examples > Analog I/O*

# ADXL3xx Accelerometer

Reads an Analog Devices ADXL3xx series (e.g. ADXL320, ADXL321, ADXL322, ADXL330) accelerometer and communicates the acceleration to the computer. The pins used are designed to be easily compatible with the breakout boards from Sparkfun. The ADXL3xx outputs the acceleration on each axis as an analog voltage between 0 and 5 volts, which is read by an analog input on the Arduino.

## Circuit



*An ADXL322 on a Sparkfun breakout board inserted into the analog input pins of an Arduino.*

Pinout for the above configuration:

| **Breakout Board Pin** | Self-Test | Z-Axis | Y-Axis | X-Axis | Ground | VDD |
|---|---|---|---|---|---|---|
| **Arduino Analog Input Pin** | 0 | 1 | 2 | 3 | 4 | 5 |

Or, if you're using just the accelerometer:

| **ADXL3xx Pin** | Self-Test | ZOut | YOut | XOut | Ground | VDD |
|---|---|---|---|---|---|---|
| **Arduino Pin** | *None* (unconnected) | Analog Input 1 | Analog Input 2 | Analog Input 3 | GND | 5V |

## Code

```
int groundpin = 18;             // analog input pin 4
int powerpin = 19;              // analog input pin 5
int xpin = 3;                   // x-axis of the accelerometer
int ypin = 2;                   // y-axis
int zpin = 1;                   // z-axis (only on 3-axis models)

void setup()
```

```
{
  Serial.begin(9600);

  // Provide ground and power by using the analog inputs as normal
  // digital pins.  This makes it possible to directly connect the
  // breakout board to the Arduino.  If you use the normal 5V and
  // GND pins on the Arduino, you can remove these lines.
  pinMode(groundPin, OUTPUT);
  pinMode(powerPin, OUTPUT);
  digitalWrite(groundPin, LOW);
  digitalWrite(powerPin, HIGH);
}

void loop()
{
  Serial.print(analogRead(xpin));
  Serial.print(" ");
  Serial.print(analogRead(ypin));
  Serial.print(" ");
  Serial.print(analogRead(zpin));
  Serial.println();
  delay(1000);
}
```

## Data

Here are some accelerometer readings collected by the positioning the y-axis of an ADXL322 2g accelerometer at various angles from ground. Values should be the same for the other axes, but will vary based on the sensitivity of the device. With the axis horizontal (i.e. parallel to ground or 0°), the accelerometer reading should be around 512, but values at other angles will be different for a different accelerometer (e.g. the ADXL302 5g one).

| **Angle** | -90 | -80 | -70 | -60 | -50 | -40 | -30 | -20 | -10 | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Acceleration** | 662 | 660 | 654 | 642 | 628 | 610 | 589 | 563 | 537 | 510 | 485 | 455 | 433 | 408 | 390 | 374 | 363 | 357 | 355 |

## Tutorial.AccelerometerMemsic2125 History

Hide minor edits - Show changes to markup

December 01, 2006, at 04:58 AM by David Cuartielles -
Changed lines 11-12 from:

*Protoboard with an Accelerometer, picture by Anders Gran*

to:

*Protoboard with an Accelerometer, picture by Anders Gran*

Restore
November 21, 2005, at 10:26 AM by 195.178.229.112 -
Changed lines 3-4 from:

The Memsic 2125 is a dual axis accelerometer sensor from Parallax able of measuring up to acceleration in the range of 2g. When making very accurate measurements, the sensor counts with a temperature pin that can be used to compensate possible errors.

to:

The Memsic 2125 is a dual axis accelerometer sensor from Parallax able of measuring up to a 2g acceleration. When making very accurate measurements, the sensor counts with a temperature pin that can be used to compensate possible errors.

Restore
November 21, 2005, at 09:57 AM by 195.178.229.112 -
Deleted lines 12-13:

The following example is an adaptation of the previous one. Marcos Yarza added two 220Ohm resistors to the pins coming out of the accelerometer. The board chosen for this small circuit is just a piece of prototyping board.

Restore
November 21, 2005, at 09:52 AM by 195.178.229.112 -
Changed line 115 from:

Here the code is exactly the same as before, but the installation on the board allows to embed the whole circutry in a much smaller housing.

to:

Here the code is exactly the same as before (changing the input pins to be 2 and 3), but the installation on the board allows to embed the whole circutry in a much smaller housing.

Restore
November 21, 2005, at 09:52 AM by 195.178.229.112 -
Added line 115:

Here the code is exactly the same as before, but the installation on the board allows to embed the whole circutry in a much smaller housing.

Restore
November 21, 2005, at 09:50 AM by 195.178.229.112 -
Added lines 13-14:

The following example is an adaptation of the previous one. Marcos Yarza added two 220Ohm resistors to the pins coming out of the accelerometer. The board chosen for this small circuit is just a piece of prototyping board.

Changed lines 108-114 from:

=]

to:

=]

http://static.flickr.com/28/65531406_d7150f954a.jpg

*Accelerometer mounted on prototyping board, by M. Yarza*

The following example is an adaptation of the previous one. Marcos Yarza added two 220Ohm resistors to the pins coming out of the accelerometer. The board chosen for this small circuit is just a piece of prototyping board.

Restore
November 21, 2005, at 07:32 AM by 195.178.229.112 -
Changed lines 7-8 from:

The example shown here was mounted by Anders Grant, while the software was created by Marcos Yarza, who is Arduino's accelerometer technology researcher, at the University of Zaragoza, Spain. The board is connected minimally, only the two axis pins are plugged to the board, leaving the temperature pin open.

to:

The example shown here was mounted by Anders Gran, while the software was created by Marcos Yarza, who is Arduino's accelerometer technology researcher, at the University of Zaragoza, Spain. The board is connected minimally, only the two axis pins are plugged to the board, leaving the temperature pin open.

Changed lines 11-12 from:

*Protoboard with an Accelerometer, picture by Anders Grant*

to:

*Protoboard with an Accelerometer, picture by Anders Gran*

Restore
November 21, 2005, at 07:31 AM by 195.178.229.112 -
Changed lines 7-8 from:

The example shown here was mounted by Anders Grant, while the software was created by Marcos Yarza, who is Arduino's accelerometer technology researcher, at the University of Zaragoza, Spain. The board is connected minimally, only the two axis are plugged to the board, leaving the temperature pin open.

to:

The example shown here was mounted by Anders Grant, while the software was created by Marcos Yarza, who is Arduino's accelerometer technology researcher, at the University of Zaragoza, Spain. The board is connected minimally, only the two axis pins are plugged to the board, leaving the temperature pin open.

Restore
November 21, 2005, at 07:30 AM by 195.178.229.112 -
Changed line 89 from:

```
serialWrite('X');
```

to:

```
printByte('X');
```

Changed line 92 from:

```
serialWrite('Y');
```

to:

```
printByte('Y');
```

Changed line 94 from:

```
serialWrite(sign);
```

to:

```
printByte(sign);
```

Changed lines 96-97 from:

serialWrite(' ');

to:

printByte(' ');

Restore
November 21, 2005, at 07:29 AM by 195.178.229.112 -
Added lines 1-106:

## Memsic 2125 Accelerometer

The Memsic 2125 is a dual axis accelerometer sensor from Parallax able of measuring up to acceleration in the range of 2g. When making very accurate measurements, the sensor counts with a temperature pin that can be used to compensate possible errors.

The pins dedicated to measure acceleration can be connected directly to digital inputs to the Arduino board, while the the temperature should be taken as an analog input. The acceleration pins send the signals back to the computer in the form of pulses which width represents the acceleration.

The example shown here was mounted by Anders Grant, while the software was created by Marcos Yarza, who is Arduino's accelerometer technology researcher, at the University of Zaragoza, Spain. The board is connected minimally, only the two axis are plugged to the board, leaving the temperature pin open.

http://static.flickr.com/30/65458902_f4f2898ed9.jpg

*Protoboard with an Accelerometer, picture by Anders Grant*

```
 /* Accelerometer Sensor
* --------------------
*
* Reads an 2-D accelerometer
* attached to a couple of digital inputs and
* sends their values over the serial port; makes
* the monitor LED blink once sent
*
*
* http://www.0j0.org
* copyleft 2005 K3 - Malmo University - Sweden
* @author: Marcos Yarza
* @hardware: Marcos Yarza
* @project: SMEE - Experiential Vehicles
* @sponsor: Experiments in Art and Technology Sweden, 1:1 Scale
*/

int ledPin = 13;
int xaccPin = 7;
int yaccPin = 6;
int value = 0;
int accel = 0;
char sign = ' ';

int timer = 0;
int count = 0;

void setup() {
beginSerial(9600); // Sets the baud rate to 9600
pinMode(ledPin, OUTPUT);
pinMode(xaccPin, INPUT);
pinMode(yaccPin, INPUT);
}

/* (int) Operate Acceleration
* function to calculate acceleration
* returns an integer
```

```
*/
int operateAcceleration(int time1) {
return abs(8 * (time1 / 10 - 500));
}

/* (void) readAccelerometer
* procedure to read the sensor, calculate
* acceleration and represent the value
*/
void readAcceleration(int axe){
timer = 0;
count = 0;
value = digitalRead(axe);
while(value == HIGH) { // Loop until pin reads a low
value = digitalRead(axe);
}
while(value == LOW) { // Loop until pin reads a high
value = digitalRead(axe);
}
while(value == HIGH) { // Loop until pin reads a low and count
value = digitalRead(axe);
count = count + 1;
}
timer = count * 18; //calculate the teme in miliseconds

//operate sign
if (timer > 5000){
sign = '+';
}
if (timer < 5000){
sign = '-';
}

//determine the value
accel = operateAcceleration(timer);

//Represent acceleration over serial port
if (axe == 7){
serialWrite('X');
}
else {
serialWrite('Y');
}
serialWrite(sign);
printInteger(accel);
serialWrite(' ');

}
void loop() {
readAcceleration(xaccPin); //reads and represents acceleration X
readAcceleration(yaccPin); //reads and represents acceleration Y
digitalWrite(ledPin, HIGH);
delay(300);
digitalWrite(ledPin, LOW);
}
```

Restore

## Arduino : Tutorial / Accelerometer Memsic 2125

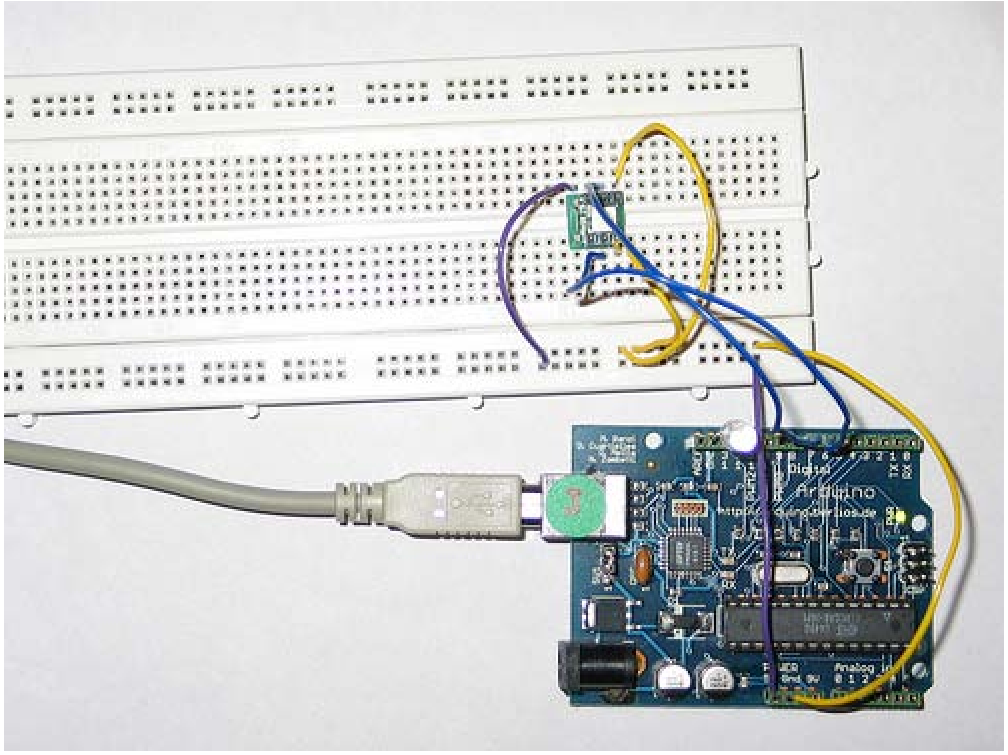
**Learning** Examples | Foundations | Hacking | Links

# Memsic 2125 Accelerometer

The Memsic 2125 is a dual axis accelerometer sensor from Parallax able of measuring up to a 2g acceleration. When making very accurate measurements, the sensor counts with a temperature pin that can be used to compensate possible errors.

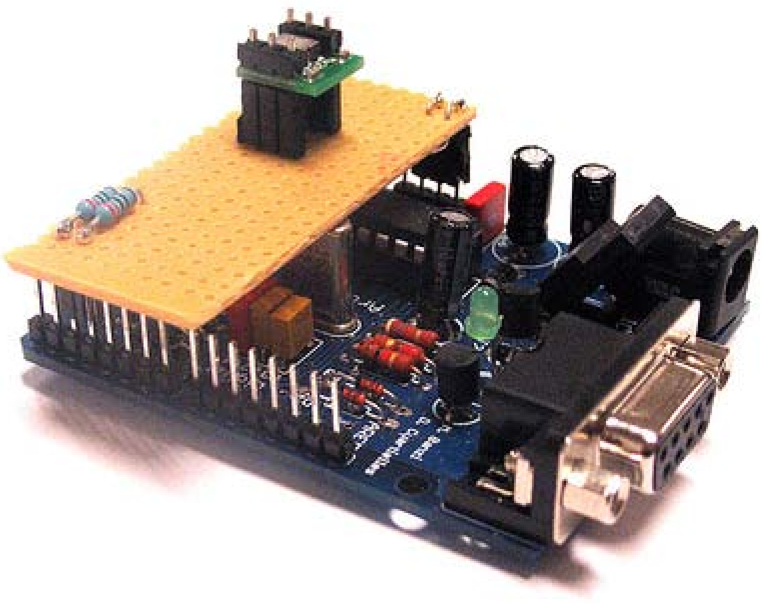
The pins dedicated to measure acceleration can be connected directly to digital inputs to the Arduino board, while the the temperature should be taken as an analog input. The acceleration pins send the signals back to the computer in the form of pulses which width represents the acceleration.

The example shown here was mounted by Anders Gran, while the software was created by Marcos Yarza, who is Arduino's accelerometer technology researcher, at the University of Zaragoza, Spain. The board is connected minimally, only the two axis pins are plugged to the board, leaving the temperature pin open.



*Protoboard with an Accelerometer, picture by Anders Gran*

```
 /* Accelerometer Sensor
* --------------------
*
* Reads an 2-D accelerometer
* attached to a couple of digital inputs and
* sends their values over the serial port; makes
* the monitor LED blink once sent
*
*
* http://www.0j0.org
* copyleft 2005 K3 - Malmo University - Sweden
* @author: Marcos Yarza
* @hardware: Marcos Yarza
* @project: SMEE - Experiential Vehicles
* @sponsor: Experiments in Art and Technology Sweden, 1:1 Scale
*/

int ledPin = 13;
int xaccPin = 7;
int yaccPin = 6;
int value = 0;
int accel = 0;
char sign = ' ';

int timer = 0;
int count = 0;

void setup() {
beginSerial(9600); // Sets the baud rate to 9600
pinMode(ledPin, OUTPUT);
pinMode(xaccPin, INPUT);
pinMode(yaccPin, INPUT);
}

/* (int) Operate Acceleration
* function to calculate acceleration
* returns an integer
*/
int operateAcceleration(int time1) {
return abs(8 * (time1 / 10 - 500));
}

/* (void) readAccelerometer
* procedure to read the sensor, calculate
* acceleration and represent the value
*/
void readAcceleration(int axe){
timer = 0;
count = 0;
value = digitalRead(axe);
while(value == HIGH) { // Loop until pin reads a low
value = digitalRead(axe);
}
while(value == LOW) { // Loop until pin reads a high
value = digitalRead(axe);
}
while(value == HIGH) { // Loop until pin reads a low and count
value = digitalRead(axe);
count = count + 1;
}
timer = count * 18; //calculate the teme in miliseconds

//operate sign
if (timer > 5000){
sign = '+';
}
if (timer < 5000){
sign = '-';
}

//determine the value
accel = operateAcceleration(timer);

//Represent acceleration over serial port
if (axe == 7){
printByte('X');
}
else {
printByte('Y');
}
printByte(sign);
printInteger(accel);
printByte(' ');

}
void loop() {
readAcceleration(xaccPin); //reads and represents acceleration X
readAcceleration(yaccPin); //reads and represents acceleration Y
digitalWrite(ledPin, HIGH);
delay(300);
digitalWrite(ledPin, LOW);
}
```



*Accelerometer mounted on prototyping board, by M. Yarza*

The following example is an adaptation of the previous one. Marcos Yarza added two 220Ohm resistors to the pins coming out of the accelerometer. The board chosen for this small circuit is just a piece of prototyping board. Here the code is exactly the same as before (changing the input pins to be 2 and 3), but the installation on the board allows to embed the whole circutry in a much smaller housing.

## Tutorial.UltrasoundSensor History

Hide minor edits - Show changes to markup

November 21, 2005, at 10:24 AM by 195.178.229.112 -
Added lines 1-12:

## PING range finder

The PING range finder is an ultrasound sensor from Parallax able of detecting objects up to a 3 mts distance. The sensor counts with 3 pins, two are dedicated to power and ground, while the third one is used both as input and output.

The pin dedicated to make the readings has to be shifting configuration from input to output according to the PING specification sheet. First we have to send a pulse that will make the sensor send an ultrasound tone and wait for an echo. Once the tone is received back, the sensor will send a pulse over the same pin as earlier. The width of that pulse will determine the distance to the object.

The example shown here was mounted by Marcus Hannerstig, while the software was created by David Cuartielles. The board is connected as explained using only wires coming from an old computer.

http://static.flickr.com/31/65531405_fa57b9ff66.jpg

*Ultrasound sensor connected to an Arduino USB v1.0*

Restore
October 10, 2005, at 12:31 PM by 81.33.141.194 -
Added line 1:

```
 [=
```

Changed lines 12-13 from:

```
int ultraSoundSignal = 7; // Ultrasound signal pin
```

to:

```
int ultraSoundSignal = 7; // Ultrasound signal pin
```

Changed lines 15-21 from:

```
int ultrasoundValue = 0;
```

```
int timecount = 0; // Echo counter
```

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

to:

```
int ultrasoundValue = 0; int timecount = 0; // Echo counter int ledPin = 13; // LED connected to digital pin 13
```

Deleted line 19:
Deleted line 20:
Deleted line 21:
Changed lines 25-31 from:

```
timecount = 0;
```

```
val = 0;
```

```
pinMode(ultraSoundSignal, OUTPUT); // Switch signalpin to output
```

to:

```
 timecount = 0;
 val = 0;
 pinMode(ultraSoundSignal, OUTPUT); // Switch signalpin to output
```

Deleted line 29:
Deleted line 30:
Changed lines 33-42 from:

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

to:

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

Deleted line 39:
Deleted line 40:
Changed lines 43-48 from:

pinMode(ultraSoundSignal, INPUT); // Switch signalpin to input

val = digitalRead(ultraSoundSignal); // Append signal value to val

while(val == LOW) { // Loop until pin reads a high value

to:

pinMode(ultraSoundSignal, INPUT); // Switch signalpin to input val = digitalRead(ultraSoundSignal); // Append signal value to val while(val == LOW) { // Loop until pin reads a high value

Deleted line 46:
Changed lines 49-50 from:

while(val == HIGH) { // Loop until pin reads a high value

to:

while(val == HIGH) { // Loop until pin reads a high value

Deleted line 50:
Deleted line 51:
Deleted line 54:
Deleted line 55:
Changed lines 58-61 from:

ultrasoundValue = timecount; // Append echo pulse time to ultrasoundValue

serialWrite('A'); // Example identifier for the sensor

to:

ultrasoundValue = timecount; // Append echo pulse time to ultrasoundValue

serialWrite('A'); // Example identifier for the sensor

Deleted line 61:
Deleted line 62:
Deleted line 65:
Deleted line 66:
Deleted line 69:
Changed lines 71-74 from:

}

to:

}

Deleted line 73:
Deleted line 74:
Changed lines 78-79 from:

}

to:

}

 =]

Restore
October 08, 2005, at 03:58 AM by 62.97.121.93 -
Added lines 1-114:

```
/* Ultrasound Sensor

 *------------------
 *
 * Reads values (00014-01199) from an ultrasound sensor (3m sensor)
 * and writes the values to the serialport.
 *
 * http://www.xlab.se | http://www.0j0.org
 * copyleft 2005 Mackie for XLAB | DojoDave for DojoCorp
 *
 */

int ultraSoundSignal = 7; // Ultrasound signal pin

int val = 0;

int ultrasoundValue = 0;

int timecount = 0; // Echo counter

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

void setup() {

  beginSerial(9600);                  // Sets the baud rate to 9600

  pinMode(ledPin, OUTPUT);            // Sets the digital pin as output

}

void loop() {

timecount = 0;

val = 0;

pinMode(ultraSoundSignal, OUTPUT); // Switch signalpin to output

/* Send low-high-low pulse to activate the trigger pulse of the sensor

 * -------------------------------------------------------------------

 */

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

/* Listening for echo pulse
```

```
 * ----------------------------------------------------------------
 */
pinMode(ultraSoundSignal, INPUT); // Switch signalpin to input
val = digitalRead(ultraSoundSignal); // Append signal value to val
while(val == LOW) { // Loop until pin reads a high value
  val = digitalRead(ultraSoundSignal);
}
while(val == HIGH) { // Loop until pin reads a high value
  val = digitalRead(ultraSoundSignal);
  timecount = timecount +1;            // Count echo pulse time
}
/* Writing out values to the serial port
 * ----------------------------------------------------------------
 */
ultrasoundValue = timecount; // Append echo pulse time to ultrasoundValue
serialWrite('A'); // Example identifier for the sensor
printInteger(ultrasoundValue);
serialWrite(10);
serialWrite(13);
/* Lite up LED if any value is passed by the echo pulse
 * ----------------------------------------------------------------
 */
if(timecount > 0){
  digitalWrite(ledPin, HIGH);
}
/* Delay of program
 * ----------------------------------------------------------------
 */
delay(100);
}
```

Restore

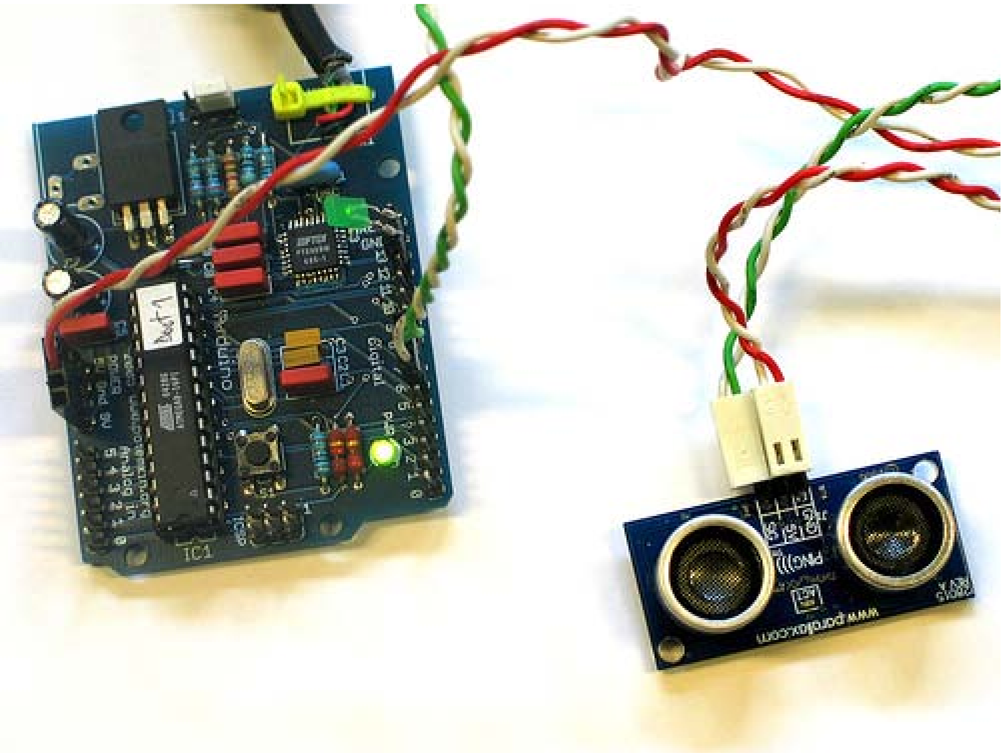
## Arduino : Tutorial / Ultrasound Sensor

# PING range finder

The PING range finder is an ultrasound sensor from Parallax able of detecting objects up to a 3 mts distance. The sensor counts with 3 pins, two are dedicated to power and ground, while the third one is used both as input and output.

The pin dedicated to make the readings has to be shifting configuration from input to output according to the PING specification sheet. First we have to send a pulse that will make the sensor send an ultrasound tone and wait for an echo. Once the tone is received back, the sensor will send a pulse over the same pin as earlier. The width of that pulse will determine the distance to the object.

The example shown here was mounted by Marcus Hannerstig, while the software was created by David Cuartielles. The board is connected as explained using only wires coming from an old computer.



*Ultrasound sensor connected to an Arduino USB v1.0*

```
/* Ultrasound Sensor
 *------------------
 *
 * Reads values (00014-01199) from an ultrasound sensor (3m sensor)
 * and writes the values to the serialport.
 *
 * http://www.xlab.se | http://www.0j0.org
 * copyleft 2005 Mackie for XLAB | DojoDave for DojoCorp
 *
 */

int ultraSoundSignal = 7; // Ultrasound signal pin
int val = 0;
int ultrasoundValue = 0;
int timecount = 0; // Echo counter
int ledPin = 13; // LED connected to digital pin 13

void setup() {
  beginSerial(9600);                  // Sets the baud rate to 9600
  pinMode(ledPin, OUTPUT);            // Sets the digital pin as output
}

void loop() {
 timecount = 0;
 val = 0;
 pinMode(ultraSoundSignal, OUTPUT); // Switch signalpin to output

/* Send low-high-low pulse to activate the trigger pulse of the sensor
 * -------------------------------------------------------------------
 */

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

/* Listening for echo pulse
 * -------------------------------------------------------------------
 */

pinMode(ultraSoundSignal, INPUT); // Switch signalpin to input
val = digitalRead(ultraSoundSignal); // Append signal value to val
while(val == LOW) { // Loop until pin reads a high value
  val = digitalRead(ultraSoundSignal);
}

while(val == HIGH) { // Loop until pin reads a high value
  val = digitalRead(ultraSoundSignal);
  timecount = timecount +1;            // Count echo pulse time
}

/* Writing out values to the serial port
 * -------------------------------------------------------------------
 */

ultrasoundValue = timecount; // Append echo pulse time to ultrasoundValue

serialWrite('A'); // Example identifier for the sensor
printInteger(ultrasoundValue);
serialWrite(10);
serialWrite(13);

/* Lite up LED if any value is passed by the echo pulse
 * -------------------------------------------------------------------
 */

if(timecount > 0){
  digitalWrite(ledPin, HIGH);
}

/* Delay of program
 * -------------------------------------------------------------------
 */

delay(100);
}
```

# Tutorial.Qt401 History

Hide minor edits - Show changes to markup

January 19, 2006, at 06:19 AM by 85.18.81.162 -
Added lines 1-5:

## qt401 sensor

full tutorial coming soon

Restore
January 19, 2006, at 06:19 AM by 85.18.81.162 -
Added lines 1-195:

```
/* qt401 demo
 * ------------
 *
 * the qt401 from qprox http://www.qprox.com is a linear capacitive sensor
 * that is able to read the position of a finger touching the sensor
 * the surface of the sensor is divided in 128 positions
 * the pin qt401_prx detects when a hand is near the sensor while
 * qt401_det determines when somebody is actually touching the sensor
 * these can be left unconnected if you are short of pins
 *
 * read the datasheet to understand the parametres passed to initialise the sensor
 *
 * Created January 2006
 * Massimo Banzi http://www.potemkin.org
 *
 * based on C code written by Nicholas Zambetti
 */


// define pin mapping
int qt401_drd = 2; // data ready
int qt401_di  = 3; // data in (from sensor)
int qt401_ss  = 4; // slave select
int qt401_clk = 5; // clock
int qt401_do  = 6; // data out (to sensor)
int qt401_det = 7; // detect
int qt401_prx = 8; // proximity


byte result;

void qt401_init() {
  // define pin directions
  pinMode(qt401_drd, INPUT);
  pinMode(qt401_di,  INPUT);
  pinMode(qt401_ss,  OUTPUT);
  pinMode(qt401_clk, OUTPUT);
  pinMode(qt401_do,  OUTPUT);
  pinMode(qt401_det, INPUT);
  pinMode(qt401 prx, INPUT);
  // initialise pins
  digitalWrite(qt401_clk,HIGH);
  digitalWrite(qt401_ss, HIGH);
}

//
//  wait for the qt401 to be ready
//
void qt401_waitForReady(void)
{
  while(!digitalRead(qt401_drd)){
    continue;
  }
}



//
//  exchange a byte with the sensor
//

byte qt401_transfer(byte data_out)
{
  byte i = 8;

  byte mask = 0;
  byte data_in = 0;

  digitalWrite(qt401_ss,LOW); // select slave by lowering ss pin
  delayMicroseconds(75); //wait for 75 microseconds

  while(0 < i) {

    mask = 0x01 << --i; // generate bitmask for the appropriate bit MSB first

    // set out byte
    if(data_out & mask){ // choose bit

      digitalWrite(qt401_do,HIGH); // send 1

    }
    else{

      digitalWrite(qt401_do,LOW); // send 0

    }

    // lower clock pin, this tells the sensor to read the bit we just put out
    digitalWrite(qt401_clk,LOW); // tick

    // give the sensor time to read the data

    delayMicroseconds(75);

    // bring clock back up

    digitalWrite(qt401_clk,HIGH); // tock

    // give the sensor some time to think

    delayMicroseconds(20);
    // now read a bit coming from the sensor
    if(digitalRead(qt401_di)){

      data_in |= mask;

    }

    //  give the sensor some time to think

      delayMicroseconds(20);

  }

  delayMicroseconds(75); //  give the sensor some time to think
  digitalWrite(qt401_ss,HIGH); // do acquisition burst

  return data_in;
}

void qt401_calibrate(void)
{
  // calibrate
  qt401_waitForReady();
  qt401_transfer(0x01);
  delay(600);

  // calibrate ends
  qt401_waitForReady();
  qt401_transfer(0x02);
  delay(600);
}


void qt401_setProxThreshold(byte amount)
{
  qt401_waitForReady();
  qt401_transfer(0x40 & (amount & 0x3F));
}


void qt401_setTouchThreshold(byte amount)
{
  qt401_waitForReady();
  qt401_transfer(0x80 & (amount & 0x3F));
}


byte qt401_driftCompensate(void)
{
  qt401_waitForReady();
  return qt401_transfer(0x03);
}


byte qt401_readSensor(void)
{

  qt401_waitForReady();
  return qt401_transfer(0x00);
}


void setup() {

  //setup the sensor
  qt401_init();
  qt401_calibrate();
  qt401_setProxThreshold(10);
  qt401_setTouchThreshold(10);

  beginSerial(9600);

}


void loop() {

  if(digitalRead(qt401_det)){
    result = qt401_readSensor();
    if(0x80 & result){
      result = result & 0x7f;

      printInteger(result);
      printNewline();

    }
  }

}
```

Restore

## Arduino : Tutorial / Qt 401

**Learning** Examples | Foundations | Hacking | Links

# qt401 sensor

full tutorial coming soon

```
/* qt401 demo
 * ------------
 *
 * the qt401 from qprox http://www.qprox.com is a linear capacitive sensor
 * that is able to read the position of a finger touching the sensor
 * the surface of the sensor is divided in 128 positions
 * the pin qt401_prx detects when a hand is near the sensor while
 * qt401_det determines when somebody is actually touching the sensor
 * these can be left unconnected if you are short of pins
 *
 * read the datasheet to understand the parametres passed to initialise the sensor
 *
 * Created January 2006
 * Massimo Banzi http://www.potemkin.org
 *
 * based on C code written by Nicholas Zambetti
 */


// define pin mapping
int qt401_drd = 2; // data ready
int qt401_di  = 3; // data in (from sensor)
int qt401_ss  = 4; // slave select
int qt401_clk = 5; // clock
int qt401_do  = 6; // data out (to sensor)
int qt401_det = 7; // detect
int qt401_prx = 8; // proximity


byte result;

void qt401_init() {
  // define pin directions
  pinMode(qt401_drd, INPUT);
  pinMode(qt401_di,  INPUT);
  pinMode(qt401_ss,  OUTPUT);
  pinMode(qt401_clk, OUTPUT);
  pinMode(qt401_do,  OUTPUT);
  pinMode(qt401_det, INPUT);
  pinMode(qt401_prx, INPUT);


  // initialise pins
  digitalWrite(qt401_clk,HIGH);
  digitalWrite(qt401_ss, HIGH);
}

//
//  wait for the qt401 to be ready
//
void qt401_waitForReady(void)
{
  while(!digitalRead(qt401_drd)){
    continue;
  }
}



//
//  exchange a byte with the sensor
//

byte qt401_transfer(byte data_out)
{
  byte i = 8;

  byte mask = 0;
  byte data_in = 0;

  digitalWrite(qt401_ss,LOW); // select slave by lowering ss pin
  delayMicroseconds(75); //wait for 75 microseconds

  while(0 < i) {

    mask = 0x01 << --i; // generate bitmask for the appropriate bit MSB first

    // set out byte
    if(data_out & mask){ // choose bit

      digitalWrite(qt401_do,HIGH); // send 1

    }
    else{

      digitalWrite(qt401_do,LOW); // send 0

    }

    // lower clock pin, this tells the sensor to read the bit we just put out
    digitalWrite(qt401_clk,LOW); // tick

    // give the sensor time to read the data

    delayMicroseconds(75);

    // bring clock back up

    digitalWrite(qt401_clk,HIGH); // tock

    // give the sensor some time to think

    delayMicroseconds(20);

    // now read a bit coming from the sensor
    if(digitalRead(qt401_di)){

      data_in |= mask;

    }

    //  give the sensor some time to think

      delayMicroseconds(20);

  }

  delayMicroseconds(75); //  give the sensor some time to think
  digitalWrite(qt401_ss,HIGH); // do acquisition burst

  return data_in;
}

void qt401_calibrate(void)
{
  // calibrate
  qt401_waitForReady();
  qt401_transfer(0x01);
  delay(600);

  // calibrate ends
  qt401_waitForReady();
  qt401_transfer(0x02);
  delay(600);
}


void qt401_setProxThreshold(byte amount)
{
  qt401_waitForReady();
  qt401_transfer(0x40 & (amount & 0x3F));
}


void qt401_setTouchThreshold(byte amount)
{
  qt401_waitForReady();
  qt401_transfer(0x80 & (amount & 0x3F));
}
```

```
byte qt401_driftCompensate(void)
{
  qt401_waitForReady();
  return qt401_transfer(0x03);
}


byte qt401_readSensor(void)
{

  qt401_waitForReady();
  return qt401_transfer(0x00);
}


void setup() {

  //setup the sensor
  qt401_init();
  qt401_calibrate();
  qt401_setProxThreshold(10);
  qt401_setTouchThreshold(10);

  beginSerial(9600);

}


void loop() {

  if(digitalRead(qt401_det)){
    result = qt401_readSensor();
    if(0x80 & result){
      result = result & 0x7f;

      printInteger(result);
      printNewline();

    }
  }

}
```

## Tutorial.PlayMelody History

Hide minor edits - Show changes to markup

October 02, 2006, at 04:15 PM by Clay Shirky -
Changed line 169 from:

[=

to:

```
 [=
```

Restore
October 02, 2006, at 04:15 PM by Clay Shirky -
Added lines 165-234:

=]

Second version, with volume control set using analogWrite()

[= /* Play Melody

```
 * -----------
 *
 * Program to play melodies stored in an array, it requires to know
 * about timing issues and about how to play tones.
 *
 * The calculation of the tones is made following the mathematical
 * operation:
 *
 *       timeHigh = 1/(2 * toneFrequency) = period / 2
 *
 * where the different tones are described as in the table:
 *
 * note         frequency       period  PW (timeHigh)
 * c            261 Hz          3830    1915
 * d            294 Hz          3400    1700
 * e            329 Hz          3038    1519
 * f            349 Hz          2864    1432
 * g            392 Hz          2550    1275
 * a            440 Hz          2272    1136
 * b            493 Hz          2028    1014
 * C            523 Hz          1912    956
 *
 * (cleft) 2005 D. Cuartielles for K3
 */
```

int ledPin = 13; int speakerOut = 9; byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'}; int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956}; byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p"; // count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 // 10 20 30 int count = 0; int count2 = 0; int count3 = 0; int MAX_COUNT = 24; int statePin = LOW;

void setup() {

```
 pinMode(ledPin, OUTPUT);
```

}

```
void loop() {

  analogWrite(speakerOut, 0);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          analogWrite(speakerOut,500);
          delayMicroseconds(tones[count2]);
          analogWrite(speakerOut, 0);
          delayMicroseconds(tones[count2]);
        }
        if (melody[count*2 + 1] == 'p') {
          // make a pause of a certain size
          analogWrite(speakerOut, 0);
          delayMicroseconds(500);
        }
      }
    }
  }

}
```

Restore
October 02, 2006, at 04:10 PM by Clay Shirky - Re-wrote Example #1
Changed lines 21-22 from:

```
 * Program to play melodies stored in an array, it requires to know
 * about timing issues and about how to play tones.
```

to:

```
 * Program to play a simple melody
```

Changed lines 23-24 from:

```
 * The calculation of the tones is made following the mathematical
 * operation:
```

to:

```
 * Tones are created by quickly pulsing a speaker on and off
 *   using PWM, to create signature frequencies.
```

Changed lines 26-33 from:

```
 *       timeHigh = 1/(2 * toneFrequency) = period / 2
```

to:

```
 * Each note has a frequency, created by varying the period of
 *  vibration, measured in microseconds. We'll use pulse-width
 *  modulation (PWM) to create that vibration.

 * We calculate the pulse-width to be half the period; we pulse
 *  the speaker HIGH for 'pulse-width' microseconds, then LOW
 *  for 'pulse-width' microseconds.
 *  This pulsing creates a vibration of the desired frequency.
```

Deleted lines 34-45:

```
 * where the different tones are described as in the table:
 *
 * note         frequency       period  PW (timeHigh)
 * c            261 Hz          3830    1915
 * d            294 Hz          3400    1700
 * e            329 Hz          3038    1519
 * f            349 Hz          2864    1432
 * g            392 Hz          2550    1275
 * a            440 Hz          2272    1136
 * b            493 Hz          2028    1014
 * C            523 Hz          1912    956
 *
```

Added lines 36-37:

```
 * Refactoring and comments 2006 clay.shirky@nyu.edu
 * See NOTES in comments at end for possible improvements
```

Changed lines 40-55 from:

int ledPin = 13; int speakerOut = 9; byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'}; int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956}; byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p"; // count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 // 10 20 30 int count = 0; int count2 = 0; int count3 = 0; int MAX_COUNT = 24; int statePin = LOW;

void setup() {

```
 pinMode(ledPin, OUTPUT);
 pinMode(speakerOut, OUTPUT);
```

to:

// TONES ========================================== // Start by defining the relationship between // note, period, & frequency.

1. define c 3830 // 261 Hz
2. define d 3400 // 294 Hz
3. define e 3038 // 329 Hz
4. define f 2864 // 349 Hz
5. define g 2550 // 392 Hz
6. define a 2272 // 440 Hz
7. define b 2028 // 493 Hz
8. define C 1912 // 523 Hz

// Define a special note, 'R', to represent a rest

1. define R 0

// SETUP ============================================ // Set up speaker on a PWM pin (digital 9, 10 or 11) int speakerOut = 9; // Do we want debugging on serial out? 1 for yes, 0 for no int DEBUG = 1;

void setup() {

```
  pinMode(speakerOut, OUTPUT);
  if (DEBUG) {
    Serial.begin(9600); // Set serial out if we want debugging
  }
```

Added lines 67-112:

// MELODY and TIMING ======================================= // melody[] is an array of notes, accompanied by beats[], // which sets each note's relative length (higher #, longer note) int melody[] = { C, b, g, C, b, e, R, C, c, g, a, C }; int beats[] = { 16, 16, 16, 8, 8, 16, 32, 16, 16, 16, 8, 8 }; int MAX_COUNT = sizeof(melody) / 2; // Melody length, for looping.

// Set overall tempo long tempo = 10000; // Set length of pause between notes int pause = 1000; // Loop variable to increase Rest length int rest_count = 100; //<-BLETCHEROUS HACK; See NOTES

// Initialize core variables int tone = 0; int beat = 0; long duration = 0;

// PLAY TONE ============================================== // Pulse the speaker to play a tone for a particular duration void playTone() {

```
  long elapsed_time = 0;
  if (tone > 0) { // if this isn't a Rest beat, while the tone has
    //  played less long than 'duration', pulse speaker HIGH and LOW
    while (elapsed_time < duration) {

      digitalWrite(speakerOut,HIGH);
      delayMicroseconds(tone / 2);

      // DOWN
      digitalWrite(speakerOut, LOW);
      delayMicroseconds(tone / 2);

      // Keep track of how long we pulsed
      elapsed_time += (tone);
    }
  }
  else { // Rest beat; loop times delay
    for (int j = 0; j < rest_count; j++) { // See NOTE on rest_count
      delayMicroseconds(duration);
    }
  }

}
```

// LET THE WILD RUMPUS BEGIN =============================

Changed lines 114-131 from:

```
  digitalWrite(speakerOut, LOW);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          digitalWrite(speakerOut,HIGH);
          delayMicroseconds(tones[count2]);
          digitalWrite(speakerOut, LOW);
          delayMicroseconds(tones[count2]);
        }
        if (melody[count*2 + 1] == 'p') {
          // make a pause of a certain size
          digitalWrite(speakerOut, 0);
          delayMicroseconds(500);
        }
      }
```

to:

```
  // Set up a counter to pull from melody[] and beats[]
  for (int i=0; i<MAX_COUNT; i++) {
    tone = melody[i];
    beat = beats[i];

    duration = beat * tempo; // Set up timing

    playTone();
    // A pause between notes...
    delayMicroseconds(pause);

    if (DEBUG) { // If debugging, report loop, tone, beat, and duration
      Serial.print(i);
      Serial.print(":");
      Serial.print(beat);
      Serial.print(" ");
      Serial.print(tone);
      Serial.print(" ");
      Serial.println(duration);
```

Changed lines 137-166 from:

=]

### Example 2: Play Melody _ faded volume

[=

/* Play Melody - FV

```
 * ----------------
 *
 * Program to play melodies stored in an array, it requires to know
 * about timing issues and about how to play tones.
 *
 * The calculation of the tones is made following the mathematical
 * operation:
 *
 *       timeHigh = 1/(2 * toneFrequency) = period / 2
 *
 * where the different tones are described as in the table:
 *
 * note         frequency       period  PW (timeHigh)
 * c            261 Hz          3830    1915
 * d            294 Hz          3400    1700
 * e            329 Hz          3038    1519
 * f            349 Hz          2864    1432
 * g            392 Hz          2550    1275
 * a            440 Hz          2272    1136
 * b            493 Hz          2028    1014
 * C            523 Hz          1912    956
 *
 * We use the Pulse Width feature with analogWrite to change volume
 *
 * (cleft) 2005 D. Cuartielles for K3
```

to:

/*

```
 * NOTES
 * The program purports to hold a tone for 'duration' microseconds.
 *  Lies lies lies! It holds for at least 'duration' microseconds, _plus_
 *  any overhead created by incremeting elapsed_time (could be in excess of
 *  3K microseconds) _plus_ overhead of looping and two digitalWrites()
 *
 * As a result, a tone of 'duration' plays much more slowly than a rest
 *  of 'duration.' rest_count creates a loop variable to bring 'rest' beats
 *  in line with 'tone' beats of the same length.
 *
 * rest_count will be affected by chip architecture and speed, as well as
 *  overhead from any program mods. Past behavior is no guarantee of future
 *  performance. Your mileage may vary. Light fuse and get away.
 *
 * This could use a number of enhancements:
 * ADD code to let the programmer specify how many times the melody should
 *     loop before stopping
 * ADD another octave
 * MOVE tempo, pause, and rest_count to #define statements
 * RE-WRITE to include volume, using analogWrite, as with the second program at
 *          http://www.arduino.cc/en/Tutorial/PlayMelody
 * ADD code to make the tempo settable by pot or other input device
 * ADD code to take tempo or volume settable by serial communication
 *          (Requires 0005 or higher.)
 * ADD code to create a tone offset (higer or lower) through pot etc
 * REPLACE random melody with opening bars to 'Smoke on the Water'
```

Deleted lines 164-205:

int ledPin = 13; int speakerOut = 9; int volume = 300; // maximum volume is 1000 byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'}; int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956}; byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p"; // count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 // 10 20 30 int count = 0; int count2 = 0; int count3 = 0; int MAX_COUNT = 24; int statePin = LOW;

void setup() {

```
 pinMode(ledPin, OUTPUT);
```

```
}
```

void loop() {

```
  analogWrite(speakerOut, 0);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          analogWrite(speakerOut,volume);
          delayMicroseconds(tones[count2]);
          analogWrite(speakerOut, 0);
          delayMicroseconds(tones[count2]);
        }
        if (melody[count*2 + 1] == 'p') {
          // make a pause of a certain size
          analogWrite(speakerOut, 0);
          delayMicroseconds(500);
        }
      }
    }
  }
```

```
}
```

Restore
October 18, 2005, at 01:50 AM by 195.178.229.25 -
Changed lines 5-6 from:

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the pin number 10, that supports the functionality of writing a PWM signal to it, and not just a plain HIGH or LOW value.

to:

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the pin number 9, that supports the functionality of writing a PWM signal to it, and not just a plain HIGH or LOW value.

The first example of the code will just send a square wave to the piezo, while the second one will make use of the PWM functionality to control the volume through changing the Pulse Width.

Changed lines 13-14 from:

*Example of connection of a Piezo to pin 10*

to:

*Example of connection of a Piezo to pin 9*

### Example 1: Play Melody

Added line 59:

```
 pinMode(speakerOut, OUTPUT);
```

Changed line 63 from:

```
  analogWrite(speakerOut, 0);
```

to:

```
  digitalWrite(speakerOut, LOW);
```

Changed line 70 from:

```
          analogWrite(speakerOut,500);
```

to:

```
          digitalWrite(speakerOut,HIGH);
```

Changed line 72 from:

```
          analogWrite(speakerOut, 0);
```

to:

```
          digitalWrite(speakerOut, LOW);
```

Changed line 77 from:

```
          analogWrite(speakerOut, 0);
```

to:

```
          digitalWrite(speakerOut, 0);
```

Added lines 85-157:

=]

### Example 2: Play Melody _ faded volume

[=

/* Play Melody - FV

```
 * ----------------
 *
 * Program to play melodies stored in an array, it requires to know
 * about timing issues and about how to play tones.
 *
 * The calculation of the tones is made following the mathematical
 * operation:
 *
 *       timeHigh = 1/(2 * toneFrequency) = period / 2
 *
 * where the different tones are described as in the table:
 *
 * note         frequency       period  PW (timeHigh)
 * c            261 Hz          3830    1915
 * d            294 Hz          3400    1700
 * e            329 Hz          3038    1519
 * f            349 Hz          2864    1432
 * g            392 Hz          2550    1275
 * a            440 Hz          2272    1136
 * b            493 Hz          2028    1014
 * C            523 Hz          1912    956
 *
 * We use the Pulse Width feature with analogWrite to change volume
 *
 * (cleft) 2005 D. Cuartielles for K3
 */
```

```
int ledPin = 13; int speakerOut = 9; int volume = 300; // maximum volume is 1000 byte names[] = {'c', 'd', 'e', 'f', 'g', 'a',
'b', 'C'}; int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956}; byte melody[] =
"2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p"; // count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
5 6 7 8 9 0 // 10 20 30 int count = 0; int count2 = 0; int count3 = 0; int MAX_COUNT = 24; int statePin = LOW;
```

```
void setup() {

 pinMode(ledPin, OUTPUT);

}

void loop() {
  analogWrite(speakerOut, 0);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          analogWrite(speakerOut,volume);
          delayMicroseconds(tones[count2]);
          analogWrite(speakerOut, 0);
          delayMicroseconds(tones[count2]);
        }
        if (melody[count*2 + 1] == 'p') {
          // make a pause of a certain size
          analogWrite(speakerOut, 0);
          delayMicroseconds(500);
        }
      }
    }
  }

}
```

Restore
October 17, 2005, at 05:32 PM by 195.178.229.25 -
Added lines 7-8:

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the output. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc.

Restore
October 17, 2005, at 05:29 PM by 195.178.229.25 -
Changed lines 7-8 from:
to:

http://static.flickr.com/31/53523608_3d4268ba68.jpg

*Example of connection of a Piezo to pin 10*

Restore
October 17, 2005, at 05:21 PM by 195.178.229.25 -
Added lines 1-76:

## Play Melody

This example makes use of a Piezo Speaker in order to play melodies. We are taking advantage of the processors capability to produde PWM signals in order to play music. There is more information about how PWM works written by David Cuartielles here and even at K3's old course guide

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the pin number 10, that supports the functionality of writing a PWM signal to it, and not just a plain HIGH or LOW value.

```
/* Play Melody
 * -----------
 *
 * Program to play melodies stored in an array, it requires to know
 * about timing issues and about how to play tones.
 *
 * The calculation of the tones is made following the mathematical
 * operation:
 *
 *       timeHigh = 1/(2 * toneFrequency) = period / 2
 *
 * where the different tones are described as in the table:
 *
 * note         frequency       period  PW (timeHigh)
 * c            261 Hz          3830    1915
 * d            294 Hz          3400    1700
 * e            329 Hz          3038    1519
 * f            349 Hz          2864    1432
 * g            392 Hz          2550    1275
 * a            440 Hz          2272    1136
 * b            493 Hz          2028    1014
 * C            523 Hz          1912    956
 *
 * (cleft) 2005 D. Cuartielles for K3
 */

int ledPin = 13;
int speakerOut = 9;
byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};
byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p";
// count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//                                10                  20                  30
int count = 0;
int count2 = 0;
int count3 = 0;
int MAX_COUNT = 24;
int statePin = LOW;

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

void loop() {
  analogWrite(speakerOut, 0);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          analogWrite(speakerOut,500);
          delayMicroseconds(tones[count2]);
          analogWrite(speakerOut, 0);
          delayMicroseconds(tones[count2]);
        }
        if (melody[count*2 + 1] == 'p') {
          // make a pause of a certain size
          analogWrite(speakerOut, 0);
          delayMicroseconds(500);
        }
      }
    }
  }
}
```

Restore

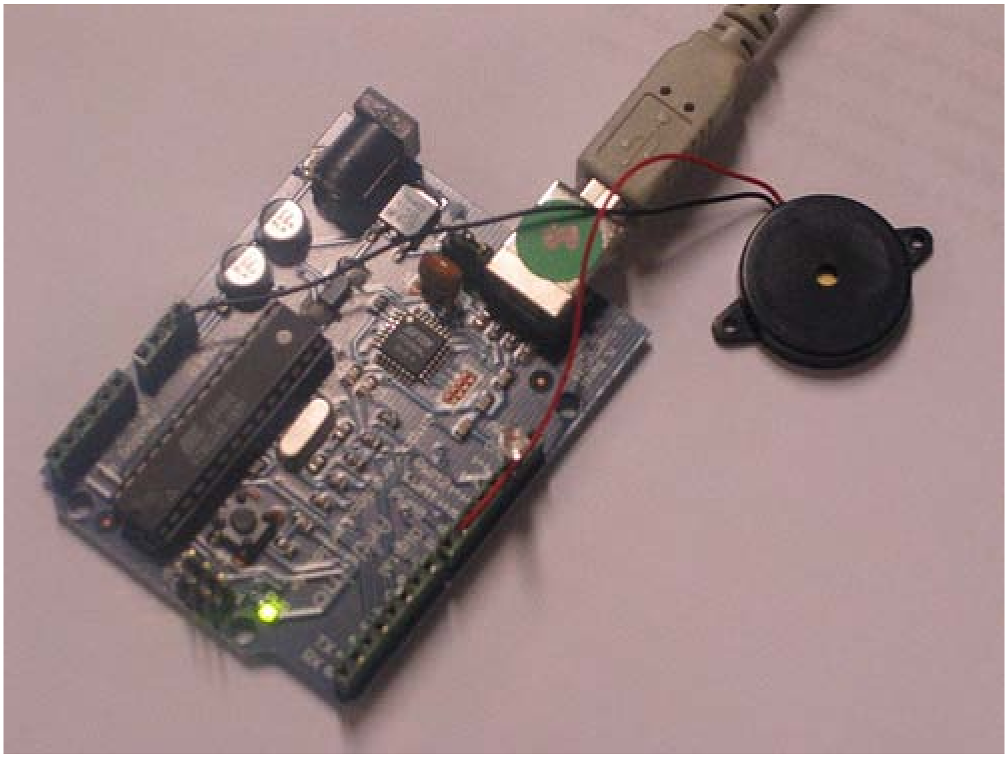
# Play Melody

This example makes use of a Piezo Speaker in order to play melodies. We are taking advantage of the processors capability to produde PWM signals in order to play music. There is more information about how PWM works written by David Cuartielles here and even at K3's old course guide

A Piezo is nothing but an electronic device that can both be used to play tones and to detect tones. In our example we are plugging the Piezo on the pin number 9, that supports the functionality of writing a PWM signal to it, and not just a plain HIGH or LOW value.

The first example of the code will just send a square wave to the piezo, while the second one will make use of the PWM functionality to control the volume through changing the Pulse Width.

The other thing to remember is that Piezos have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the output. Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc.



*Example of connection of a Piezo to pin 9*

## Example 1: Play Melody

```
/* Play Melody
 * -----------
 *
 * Program to play a simple melody
 *
 * Tones are created by quickly pulsing a speaker on and off
 *   using PWM, to create signature frequencies.
 *
 * Each note has a frequency, created by varying the period of
 *  vibration, measured in microseconds. We'll use pulse-width
 *  modulation (PWM) to create that vibration.

 * We calculate the pulse-width to be half the period; we pulse
 *  the speaker HIGH for 'pulse-width' microseconds, then LOW
 *  for 'pulse-width' microseconds.
 *  This pulsing creates a vibration of the desired frequency.
 *
 * (cleft) 2005 D. Cuartielles for K3
 * Refactoring and comments 2006 clay.shirky@nyu.edu
 * See NOTES in comments at end for possible improvements
 */

// TONES  ==========================================
// Start by defining the relationship between
//       note, period, &  frequency.
#define  c     3830    // 261 Hz
#define  d     3400    // 294 Hz
#define  e     3038    // 329 Hz
#define  f     2864    // 349 Hz
#define  g     2550    // 392 Hz
#define  a     2272    // 440 Hz
#define  b     2028    // 493 Hz
#define  C     1912    // 523 Hz
// Define a special note, 'R', to represent a rest
#define  R     0

// SETUP ============================================
// Set up speaker on a PWM pin (digital 9, 10 or 11)
int speakerOut = 9;
// Do we want debugging on serial out? 1 for yes, 0 for no
int DEBUG = 1;

void setup() {
  pinMode(speakerOut, OUTPUT);
  if (DEBUG) {
    Serial.begin(9600); // Set serial out if we want debugging
  }
}

// MELODY and TIMING  =======================================
//  melody[] is an array of notes, accompanied by beats[],
//  which sets each note's relative length (higher #, longer note)
int melody[] = {  C,  b,  g,  C,  b,   e,  R,  C,  c,  g, a, C };
int beats[]  = { 16, 16, 16,  8,  8,  16, 32, 16, 16, 16, 8, 8 };
int MAX_COUNT = sizeof(melody) / 2; // Melody length, for looping.

// Set overall tempo
long tempo = 10000;
// Set length of pause between notes
int pause = 1000;
// Loop variable to increase Rest length
int rest_count = 100; //<-BLETCHEROUS HACK; See NOTES

// Initialize core variables
int tone = 0;
int beat = 0;
long duration  = 0;

// PLAY TONE  ==============================================
// Pulse the speaker to play a tone for a particular duration
void playTone() {
  long elapsed_time = 0;
  if (tone > 0) { // if this isn't a Rest beat, while the tone has
    //  played less long than 'duration', pulse speaker HIGH and LOW
    while (elapsed_time < duration) {

      digitalWrite(speakerOut,HIGH);
      delayMicroseconds(tone / 2);

      // DOWN
      digitalWrite(speakerOut, LOW);
      delayMicroseconds(tone / 2);

      // Keep track of how long we pulsed
      elapsed_time += (tone);
    }
  }
  else { // Rest beat; loop times delay
    for (int j = 0; j < rest_count; j++) { // See NOTE on rest_count
      delayMicroseconds(duration);
    }
  }
}

// LET THE WILD RUMPUS BEGIN =============================
void loop() {
  // Set up a counter to pull from melody[] and beats[]
  for (int i=0; i<MAX_COUNT; i++) {
    tone = melody[i];
    beat = beats[i];

    duration = beat * tempo; // Set up timing

    playTone();
    // A pause between notes...
    delayMicroseconds(pause);

    if (DEBUG) { // If debugging, report loop, tone, beat, and duration
      Serial.print(i);
      Serial.print(":");
      Serial.print(beat);
      Serial.print(" ");
      Serial.print(tone);
      Serial.print(" ");
      Serial.println(duration);
    }
  }
}

/*
 * NOTES
 * The program purports to hold a tone for 'duration' microseconds.
 *  Lies lies lies! It holds for at least 'duration' microseconds, _plus_
 *  any overhead created by incremeting elapsed_time (could be in excess of
 *  3K microseconds) _plus_ overhead of looping and two digitalWrites()
 *
 * As a result, a tone of 'duration' plays much more slowly than a rest
 *  of 'duration.' rest_count creates a loop variable to bring 'rest' beats
 *  in line with 'tone' beats of the same length.
 *
 * rest_count will be affected by chip architecture and speed, as well as
 *  overhead from any program mods. Past behavior is no guarantee of future
 *  performance. Your mileage may vary. Light fuse and get away.
 *
 * This could use a number of enhancements:
 * ADD code to let the programmer specify how many times the melody should
 *     loop before stopping
 * ADD another octave
 * MOVE tempo, pause, and rest_count to #define statements
 * RE-WRITE to include volume, using analogWrite, as with the second program at
 *          http://www.arduino.cc/en/Tutorial/PlayMelody
 * ADD code to make the tempo settable by pot or other input device
 * ADD code to take tempo or volume settable by serial communication
 *          (Requires 0005 or higher.)
 * ADD code to create a tone offset (higer or lower) through pot etc
 * REPLACE random melody with opening bars to 'Smoke on the Water'
 */
```

Second version, with volume control set using analogWrite()

```
/* Play Melody
 * -----------
 *
 * Program to play melodies stored in an array, it requires to know
 * about timing issues and about how to play tones.
 *
 * The calculation of the tones is made following the mathematical
 * operation:
 *
 *       timeHigh = 1/(2 * toneFrequency) = period / 2
 *
 * where the different tones are described as in the table:
 *
 * note         frequency       period  PW (timeHigh)
 * c            261 Hz          3830    1915
 * d            294 Hz          3400    1700
 * e            329 Hz          3038    1519
 * f            349 Hz          2864    1432
 * g             392 Hz          2550    1275
 * a             440 Hz          2272    1136
 * b             493 Hz          2028    1014
 * C             523 Hz          1912    956
 *
 * (cleft) 2005 D. Cuartielles for K3
 */

int ledPin = 13;
int speakerOut = 9;
byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};
byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p";
// count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//                                  10                  20                  30
int count = 0;
int count2 = 0;
int count3 = 0;
int MAX_COUNT = 24;
int statePin = LOW;

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

void loop() {
  analogWrite(speakerOut, 0);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          analogWrite(speakerOut,500);
          delayMicroseconds(tones[count2]);
          analogWrite(speakerOut, 0);
          delayMicroseconds(tones[count2]);
        }
        if (melody[count*2 + 1] == 'p') {
          // make a pause of a certain size
          analogWrite(speakerOut, 0);
          delayMicroseconds(500);
        }
      }
    }
  }
}
```

(Printable View of http://www.arduino.cc/en/Tutorial/PlayMelody)

## Tutorial.LEDDriver History

Hide minor edits - Show changes to markup

November 11, 2005, at 01:45 AM by 81.236.128.105 -
Changed line 73 from:

```
 delay(100);                  // waits for a second
```

to:

```
 delay(100);                  // waits for a moment
```

Restore
November 10, 2005, at 02:00 PM by 195.178.229.25 -
Added lines 1-77:

## LED Driver

This example makes use of an LED Driver in order to control an almost endless amount of LEDs with only 4 pins. We use the 4794 from Philips. There is more information about this microchip that you will find in its datasheet.

An LED Driver has a shift register embedded that will take data in serial format and transfer it to parallel. It is possible to daisy chain this chip increasing the total amount of LEDs by 8 each time.

The code example you will see here is taking a value stored in the variable **dato** and showing it as a decoded binary number. E.g. if dato is 1, only the first LED will light up; if dato is 255 all the LEDs will light up.

http://static.flickr.com/30/61941877_d74eae045b.jpg

*Example of connection of a 4794*

```
/* Shift Out Data
 * --------------
 *
 * Shows a byte, stored in "dato" on a set of 8 LEDs
 *
 * (copyleft) 2005 K3, Malmo University
 * @author: David Cuartielles, Marcus Hannerstig
 * @hardware: David Cuartielles, Marcos Yarza
 * @project: made for SMEE - Experiential Vehicles
 */


int data = 9;
int strob = 8;
int clock = 10;
int oe = 11;
int count = 0;
int dato = 0;

void setup()
{
  beginSerial(9600);
  pinMode(data, OUTPUT);
  pinMode(clock, OUTPUT);
  pinMode(strob, OUTPUT);
  pinMode(oe, OUTPUT);
}

void PulseClock(void) {
    digitalWrite(clock, LOW);
    delayMicroseconds(20);
    digitalWrite(clock, HIGH);
    delayMicroseconds(50);
    digitalWrite(clock, LOW);
}

void loop()
{
   dato = 5;
   for (count = 0; count < 8; count++) {
    digitalWrite(data, dato & 01);
    //serialWrite((dato & 01) + 48);
    dato>>=1;
    if (count == 7){
    digitalWrite(oe, LOW);
    digitalWrite(strob, HIGH);

    }
    PulseClock();
     digitalWrite(oe, HIGH);
 }

  delayMicroseconds(20);
  digitalWrite(strob, LOW);
  delay(100);


  serialWrite(10);
  serialWrite(13);
 delay(100);                  // waits for a second
}
```

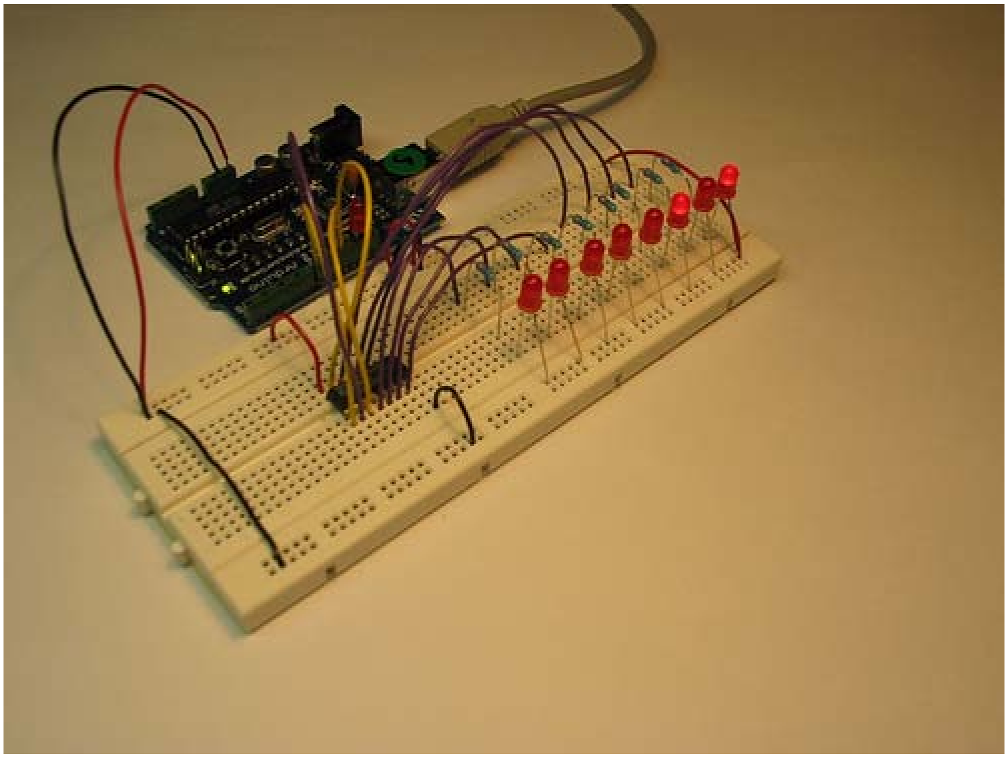
Restore

# LED Driver

This example makes use of an LED Driver in order to control an almost endless amount of LEDs with only 4 pins. We use the 4794 from Philips. There is more information about this microchip that you will find in its datasheet.

An LED Driver has a shift register embedded that will take data in serial format and transfer it to parallel. It is possible to daisy chain this chip increasing the total amount of LEDs by 8 each time.

The code example you will see here is taking a value stored in the variable **dato** and showing it as a decoded binary number. E.g. if dato is 1, only the first LED will light up; if dato is 255 all the LEDs will light up.



*Example of connection of a 4794*

```
/* Shift Out Data
 * --------------
 *
 * Shows a byte, stored in "dato" on a set of 8 LEDs
 *
 * (copyleft) 2005 K3, Malmo University
 * @author: David Cuartielles, Marcus Hannerstig
 * @hardware: David Cuartielles, Marcos Yarza
 * @project: made for SMEE - Experiential Vehicles
 */


int data = 9;
int strob = 8;
int clock = 10;
int oe = 11;
int count = 0;
int dato = 0;

void setup()
{
  beginSerial(9600);
  pinMode(data, OUTPUT);
  pinMode(clock, OUTPUT);
  pinMode(strob, OUTPUT);
  pinMode(oe, OUTPUT);
}

void PulseClock(void) {
    digitalWrite(clock, LOW);
    delayMicroseconds(20);
    digitalWrite(clock, HIGH);
    delayMicroseconds(50);
    digitalWrite(clock, LOW);
}

void loop()
{
   dato = 5;
   for (count = 0; count < 8; count++) {
    digitalWrite(data, dato & 01);
    //serialWrite((dato & 01) + 48);
    dato>>=1;
    if (count == 7){
    digitalWrite(oe, LOW);
    digitalWrite(strob, HIGH);

    }
    PulseClock();
     digitalWrite(oe, HIGH);
 }

  delayMicroseconds(20);
  digitalWrite(strob, LOW);
  delay(100);


  serialWrite(10);
  serialWrite(13);
 delay(100);                  // waits for a moment
}
```

# Tutorial.LCD8Bits History

Hide minor edits - Show changes to markup

October 17, 2005, at 07:00 PM by 195.178.229.25 -
Changed lines 1-121 from:

http://static.flickr.com/25/53544993_3611fce234_o.jpg

to:

## LCD Display - 8 bits

This example shows the most basic action to be done with a LCD display: to show a welcome message. In our case we have an LCD display with backlight and contrast control. Therefore we will use a potentiometer to regulate the contrast.

LCD displays are most of the times driven using an industrial standard established by Hitachi. According to it there is a group of pins dedicated to sending data and locations of that data on the screen, the user can choose to use 4 or 8 pins to send data. On top of that three more pins are needed to synchronize the communication towards the display.

The backdrop of this example is that we are using almost all the available pins on Arduino board in order to drive the display, but we have decided to show it this way for simplicity.

http://static.flickr.com/25/53544993_3611fce234.jpg

*Picture of a protoboard supporting the display and a potentiometer*

```
/* LCD Hola
 * --------
 *
 * This is the first example in how to use an LCD screen
 * configured with data transfers over 8 bits. The example
 * uses all the digital pins on the Arduino board, but can
 * easily display data on the display
 *
 * There are the following pins to be considered:
 *
 * - DI, RW, DB0..DB7, Enable (11 in total)
 *
 * the pinout for LCD displays is standard and there is plenty
 * of documentation to be found on the internet.
 *
 * (cleft) 2005 DojoDave for K3
 *
 */

int DI = 12;
int RW = 11;
int DB[] = {3, 4, 5, 6, 7, 8, 9, 10};
int Enable = 2;

void LcdCommandWrite(int value) {
 // poll all the pins
 int i = 0;
 for (i=DB[0]; i <= DI; i++) {
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);
 // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
}

void LcdDataWrite(int value) {
 // poll all the pins
 int i = 0;
 digitalWrite(DI, HIGH);
 digitalWrite(RW, LOW);
 for (i=DB[0]; i <= DB[7]; i++) {
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);
 // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
}

void setup (void) {
 int i = 0;
 for (i=Enable; i <= DI; i++) {
   pinMode(i,OUTPUT);
 }
 delay(100);
 // initiatize lcd after a short pause
 // needed by the LCDs controller
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(64);
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(50);
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(20);
 LcdCommandWrite(0x06);  // entry mode set:
                         // increment automatically, no display shift
 delay(20);
 LcdCommandWrite(0x0E);  // display control:
                         // turn display on, cursor on, no blinking
 delay(20);
 LcdCommandWrite(0x01);  // clear display, set cursor position to zero
 delay(100);
 LcdCommandWrite(0x80);  // display control:
                         // turn display on, cursor on, no blinking
 delay(20);
}

void loop (void) {
  LcdCommandWrite(0x02);  // set cursor position to zero
  delay(10);
  // Write the welcome message
  LcdDataWrite('H');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  LcdDataWrite(' ');
  LcdDataWrite('C');
  LcdDataWrite('a');
  LcdDataWrite('r');
  LcdDataWrite('a');
  LcdDataWrite('c');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  delay(500);
}
```

Restore

October 17, 2005, at 06:53 PM by 195.178.229.25 -
Added line 1:

http://static.flickr.com/25/53544993_3611fce234_o.jpg

Restore

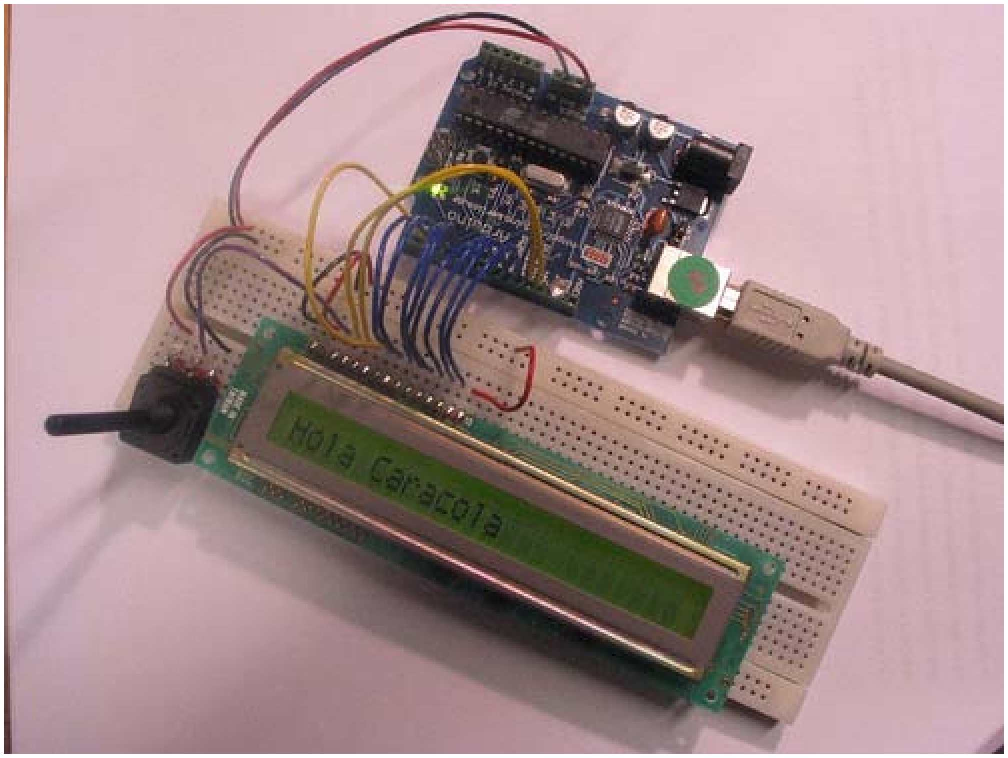
Edit Page | Page History | Printable View | All Recent Site Changes

# LCD Display - 8 bits

This example shows the most basic action to be done with a LCD display: to show a welcome message. In our case we have an LCD display with backlight and contrast control. Therefore we will use a potentiometer to regulate the contrast.

LCD displays are most of the times driven using an industrial standard established by Hitachi. According to it there is a group of pins dedicated to sending data and locations of that data on the screen, the user can choose to use 4 or 8 pins to send data. On top of that three more pins are needed to synchronize the communication towards the display.

The backdrop of this example is that we are using almost all the available pins on Arduino board in order to drive the display, but we have decided to show it this way for simplicity.



*Picture of a protoboard supporting the display and a potentiometer*

```
/* LCD Hola
 * --------
 *
 * This is the first example in how to use an LCD screen
 * configured with data transfers over 8 bits. The example
 * uses all the digital pins on the Arduino board, but can
 * easily display data on the display
 *
 * There are the following pins to be considered:
 *
 * - DI, RW, DB0..DB7, Enable (11 in total)
 *
 * the pinout for LCD displays is standard and there is plenty
 * of documentation to be found on the internet.
 *
 * (cleft) 2005 DojoDave for K3
 *
 */
```

```
int DI = 12;
int RW = 11;
int DB[] = {3, 4, 5, 6, 7, 8, 9, 10};
int Enable = 2;

void LcdCommandWrite(int value) {
 // poll all the pins
 int i = 0;
 for (i=DB[0]; i <= DI; i++) {
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);
 // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
}

void LcdDataWrite(int value) {
 // poll all the pins
 int i = 0;
 digitalWrite(DI, HIGH);
 digitalWrite(RW, LOW);
 for (i=DB[0]; i <= DB[7]; i++) {
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);
 // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);  // pause 1 ms according to datasheet
}

void setup (void) {
 int i = 0;
 for (i=Enable; i <= DI; i++) {
   pinMode(i,OUTPUT);
 }
 delay(100);
 // initiatize lcd after a short pause
 // needed by the LCDs controller
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(64);
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(50);
 LcdCommandWrite(0x30);  // function set:
                         // 8-bit interface, 1 display lines, 5x7 font
 delay(20);
 LcdCommandWrite(0x06);  // entry mode set:
                         // increment automatically, no display shift
 delay(20);
 LcdCommandWrite(0x0E);  // display control:
                         // turn display on, cursor on, no blinking
 delay(20);
 LcdCommandWrite(0x01);  // clear display, set cursor position to zero
 delay(100);
 LcdCommandWrite(0x80);  // display control:
                         // turn display on, cursor on, no blinking
 delay(20);
}

void loop (void) {
  LcdCommandWrite(0x02);  // set cursor position to zero
  delay(10);
  // Write the welcome message
  LcdDataWrite('H');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  LcdDataWrite(' ');
  LcdDataWrite('C');
  LcdDataWrite('a');
  LcdDataWrite('r');
  LcdDataWrite('a');
  LcdDataWrite('c');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  delay(500);
}
```

# Tutorial.LCDLibrary History

Hide minor edits - Show changes to markup

September 05, 2006, at 12:54 PM by Heather Dewey-Hagborg -
Changed lines 155-157 from:

To interface an LCD directly in Arduino code see this example.

to:

To interface an LCD directly in Arduino code see this example.

*LCD interface library and tutorial by Heather Dewey-Hagborg*

Restore
August 23, 2006, at 02:07 PM by Heather Dewey-Hagborg -
Restore
August 09, 2006, at 11:03 AM by Heather Dewey-Hagborg -
Changed line 17 from:

Download the LiquidCrystal library here.

to:

Download the LiquidCrystal library here.

Changed lines 66-67 from:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

Changed lines 69-70 from:

```
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

to:

```
  lcd.init(); //initialize the LCD
  digitalWrite(13,HIGH); //turn on an LED for debugging
```

Changed line 74 from:

```
   delay(1000); //repeat forever
```

to:

```
  delay(1000); //repeat forever
```

Changed lines 88-89 from:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

Changed lines 91-92 from:

```
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

to:

```
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
```

Changed line 116 from:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

Changed lines 120-121 from:

```
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

to:

```
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
```

Added line 130:
Changed line 138 from:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

Changed lines 142-143 from:

```
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

to:

```
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
```

Restore
August 09, 2006, at 10:05 AM by Heather Dewey-Hagborg -
Changed line 17 from:

Download the LiquidCrystal library here.

to:

Download the LiquidCrystal library here.

Restore
August 09, 2006, at 10:04 AM by Heather Dewey-Hagborg -
Changed line 17 from:

Download the LiquidCrystal library

to:

Download the LiquidCrystal library here.

Restore
August 09, 2006, at 10:04 AM by Heather Dewey-Hagborg -
Changed line 17 from:

Download the LiquidCrystal library here.

to:

Download the LiquidCrystal library

Restore
August 09, 2006, at 10:03 AM by Heather Dewey-Hagborg -
Changed lines 66-67 from:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

Deleted line 68:

```
   lcd.init(); //initialize the LCD
```

Added line 75:
Changed lines 87-88 from:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

Deleted line 89:

```
   lcd.init(); //initialize the LCD
```

Changed line 114 from:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

Deleted line 117:

```
   lcd.init(); //initialize the LCD
```

Added line 126:
Changed line 134 from:

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

to:

LiquidCrystal lcd = LiquidCrystal(); //create and initialize a LiquidCrystal object to control an LCD

Deleted line 137:

```
   lcd.init(); //initialize the LCD
```

Restore
August 02, 2006, at 01:25 PM by Heather Dewey-Hagborg -
Changed line 17 from:

Download the LiquidCrystal libraryhere.

to:

Download the LiquidCrystal library here.

Restore
August 02, 2006, at 01:24 PM by Heather Dewey-Hagborg -
Changed line 17 from:

Download the LiquidCrystal library Attach:LiquidCrystal.zip Δ.

to:

Download the LiquidCrystal libraryhere.

Restore
August 02, 2006, at 01:23 PM by Heather Dewey-Hagborg -
Changed line 17 from:

Download the LiquidCrystal library here.

to:

Download the LiquidCrystal library Attach:LiquidCrystal.zip Δ.

Restore
August 02, 2006, at 12:22 PM by Heather Dewey-Hagborg -
Changed lines 18-19 from:

Unzip the files and place the whole LiquidCrystal folder inside your arduino-0004\lib\targets\libraries folder. Start the Arduino program and check to make sure LiquidCrystal is now available as an option in the Sketch menu under "import library".

to:

Unzip the files and place the whole LiquidCrystal folder inside your arduino-0004\lib\targets\libraries folder. Start the Arduino program and check to make sure LiquidCrystal is now available as an option in the Sketch menu under "Import Library".

Restore
August 02, 2006, at 12:20 PM by Heather Dewey-Hagborg -
Changed lines 17-18 from:

Attach:LiquidCrystal.zip Download the LiquidCrystal library .

to:

Download the LiquidCrystal library here.

Restore
August 02, 2006, at 12:16 PM by Heather Dewey-Hagborg -
Added line 17:

Attach:LiquidCrystal.zip

Restore
August 02, 2006, at 12:14 PM by Heather Dewey-Hagborg -
Added lines 14-19:

### Install the Library

For a basic explanation of how libraries work in Arduino read the library page. Download the LiquidCrystal library . Unzip the files and place the whole LiquidCrystal folder inside your arduino-0004\lib\targets\libraries folder. Start the Arduino program and check to make sure LiquidCrystal is now available as an option in the Sketch menu under "import library".

Restore
August 02, 2006, at 12:08 PM by Heather Dewey-Hagborg -
Changed line 146 from:

To interface an LCD directly in Arduino code see .

to:

To interface an LCD directly in Arduino code see this example.

Restore
August 02, 2006, at 12:08 PM by Heather Dewey-Hagborg -
Added lines 145-146:

To interface an LCD directly in Arduino code see .

Restore
August 02, 2006, at 12:04 PM by Heather Dewey-Hagborg -
Changed lines 127-131 from:

1. include <LiquidCrystal.h>

LiquidCrystal lcd = LiquidCrystal(); char string1[] = "Hello!";

to:

1. include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD char string1[] = "Hello!"; //variable to store the string "Hello!"

Changed lines 133-134 from:

```
 lcd.init();
 digitalWrite(13,HIGH);
```

to:

```
   lcd.init(); //initialize the LCD
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

Changed lines 137-141 from:

```
   lcd.commandWrite(2);
   delay(1000);
   lcd.printIn(string1);
   delay(1000);
```

}

to:

```
   lcd.commandWrite(2); //bring the cursor to the starting position
   delay(1000); //delay 1000 ms to view change
   lcd.printIn(string1); //send the string to the LCD
   delay(1000); //delay 1000 ms to view change
```

} //repeat forever

Changed lines 144-145 from:

Using this code makes the cursor jump back and forth between the end of the message an the home position.

to:

This code makes the cursor jump back and forth between the end of the message an the home position.

Restore
August 02, 2006, at 12:02 PM by Heather Dewey-Hagborg -
Changed lines 107-111 from:

1. include <LiquidCrystal.h>

LiquidCrystal lcd = LiquidCrystal(); char string1[] = "Hello!";

to:

1. include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD char string1[] = "Hello!"; //variable to store the string "Hello!"

Changed lines 113-114 from:

```
   lcd.init();
   digitalWrite(13,HIGH);
```

to:

```
   lcd.init(); //initialize the LCD
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

Changed lines 117-121 from:

```
   lcd.clear();
   delay(1000);
   lcd.printIn(string1);
   delay(1000);
```

}

to:

```
   lcd.clear(); //clear the display
   delay(1000); //delay 1000 ms to view change
   lcd.printIn(string1); //send the string to the LCD
   delay(1000); //delay 1000 ms to view change
```

} //repeat forever

Restore

August 02, 2006, at 11:59 AM by Heather Dewey-Hagborg -
Changed lines 79-82 from:

1. include <LiquidCrystal.h>

LiquidCrystal lcd = LiquidCrystal();

to:

1. include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

Changed lines 84-85 from:

```
   lcd.init();
   digitalWrite(13,HIGH);
```

to:

```
   lcd.init(); //initialize the LCD
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

Changed lines 89-91 from:

```
   lcd.clear();
   delay(1000);
   lcd.print('a');
```

to:

```
   lcd.clear(); //clear the display
   delay(1000); //delay 1000 ms to view change
   lcd.print('a'); //send individual letters to the LCD
```

Changed lines 94-95 from:

```
   delay(1000);
```

}

to:

```
   delay(1000);//delay 1000 ms to view change
```

} //repeat forever

Restore

August 02, 2006, at 11:45 AM by Heather Dewey-Hagborg -
Changed lines 58-61 from:

1. include <LiquidCrystal.h>

LiquidCrystal lcd = LiquidCrystal();

to:

1. include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

Changed lines 63-64 from:

```
   lcd.init();
   digitalWrite(13,HIGH);
```

to:

```
   lcd.init(); //initialize the LCD
   digitalWrite(13,HIGH); //turn on an LED for debugging
```

Changed line 68 from:

```
   delay(1000);
```

to:

```
   delay(1000); //repeat forever
```

Restore
August 02, 2006, at 11:39 AM by Heather Dewey-Hagborg -
Changed lines 24-27 from:

Connect wires from the breadboard to the arduino input sockets. Look at the datasheet for your LCD board to figure out which pins are where. The pinout is as follows:

to:

Connect wires from the breadboard to the arduino input sockets. It is a lot of wires, so keep them as short and tidy as possible. Look at the datasheet for your LCD board to figure out which pins are where. Make sure to take note of whether the pin view is from the front or back side of the LCD board, you don't want to get your pins reversed!

The pinout is as follows:

Added line 56:
Changed lines 72-73 from:

If all went as planned both the LCD and the LED should turn on. Now you can use the potentiometer to adjust the contrast on the LCD until you can clearly see a cursor at the beginning of the first line.

to:

If all went as planned both the LCD and the LED should turn on. Now you can use the potentiometer to adjust the contrast on the LCD until you can clearly see a cursor at the beginning of the first line. If you turn the potentiometer too far in one direction black blocks will appear. Too far in the other direction everything will fade from the display. There should be a small spot in the middle where the cursor appears crisp and dark.

Restore
August 02, 2006, at 11:32 AM by Heather Dewey-Hagborg -
Restore
August 02, 2006, at 11:27 AM by Heather Dewey-Hagborg -
Changed lines 50-51 from:

First start by opening a new sketch in Arduino and saving it. Now go to the Sketch menu, scroll down to "import library", and choose "LiquidCrystal". The phrase #include should pop up at the top of your sketch.

to:

First start by opening a new sketch in Arduino and saving it. Now go to the Sketch menu, scroll down to "import library", and choose "LiquidCrystal". The phrase #include <LiquidCrystal.h> should pop up at the top of your sketch.

Restore
August 02, 2006, at 11:25 AM by Heather Dewey-Hagborg -
Changed lines 20-21 from:

Insert the LCD header into the breadboard and connect power and ground on the breadboard to power and ground from the microcontroller. On the Arduino module, use the 5V and any of the ground connections:

to:

Insert the LCD header into the breadboard and connect power and ground on the breadboard to power and ground from the microcontroller. On the Arduino module, use the 5V and any of the ground connections.

Changed lines 26-37 from:

Arduino LCD 2 Enable 3 Data Bit 0 (DB0) 4 (DB1) 5 (DB2) 6 (DB3) 7 (DB4) 8 (DB5) 9 (DB6) 10 (DB7) 11 Read/Write (RW) 12 Register Select (RS)

to:

Arduino LCD 2 Enable 3 Data Bit 0 (DB0) 4 (DB1) 5 (DB2) 6 (DB3) 7 (DB4) 8 (DB5) 9 (DB6) 10 (DB7) 11 Read/Write (RW) 12 Register Select (RS)

Restore
August 02, 2006, at 11:19 AM by Heather Dewey-Hagborg -
Changed lines 80-81 from:

lcd.init(); digitalWrite(13,HIGH);

to:

```
   lcd.init();
   digitalWrite(13,HIGH);
```

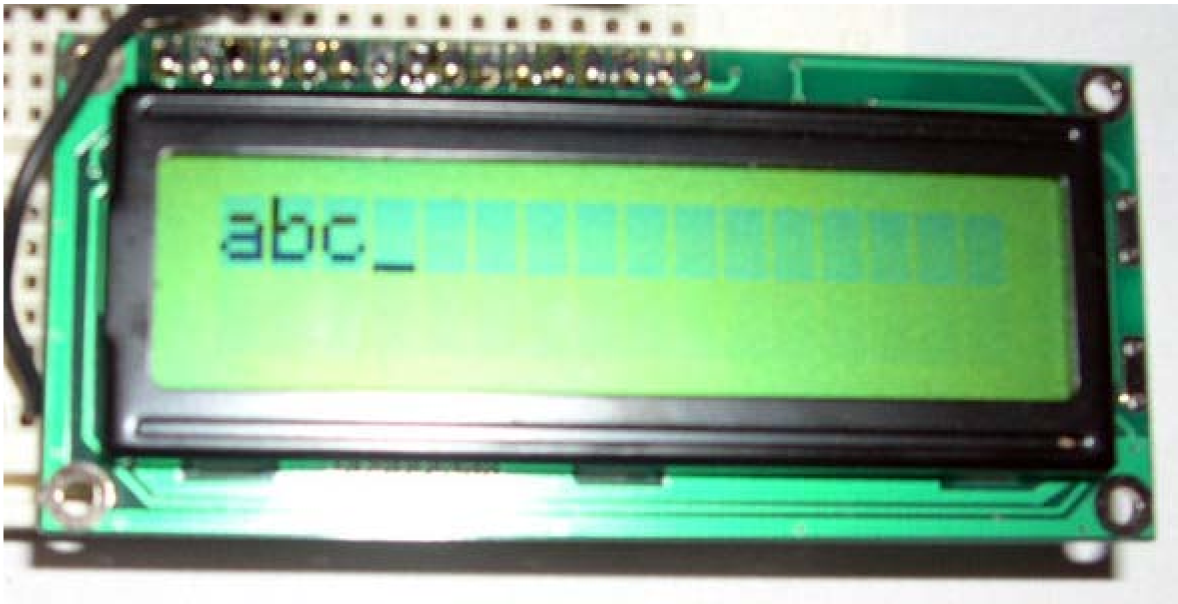
Added line 83:
Changed lines 85-90 from:

lcd.clear(); delay(1000); lcd.print('a'); lcd.print('b'); lcd.print('c'); delay(1000);

to:

```
   lcd.clear();
   delay(1000);
   lcd.print('a');
   lcd.print('b');
   lcd.print('c');
   delay(1000);
```

Added lines 96-97:



Added line 100:

[@

Added line 102:
Added line 105:
Changed lines 107-108 from:

lcd.init(); digitalWrite(13,HIGH);

to:

```
   lcd.init();
   digitalWrite(13,HIGH);
```

Changed lines 111-114 from:

lcd.clear(); delay(1000); lcd.printIn(string1); delay(1000);

to:

```
   lcd.clear();
   delay(1000);
   lcd.printIn(string1);
   delay(1000);
```

Changed lines 116-118 from:

}

Finally, you should know there is a lot of functionality in the HD44780 chip interface that is not drawn out into Arduino functions. If you are feeling ambitious glance over the datasheet and try out some of the direct commands using the commandWrite() function. For example, commandWrite(2) tells the board to move the cursor back to starting position.

to:

@]

Finally, you should know there is a lot of functionality in the HD44780 chip interface that is not drawn out into Arduino functions. If you are feeling ambitious glance over the datasheet and try out some of the direct commands using the commandWrite() function. For example, commandWrite(2) tells the board to move the cursor back to starting position. Here is an example:

Added line 122:
Added line 125:
Changed line 138 from:

Using this code makes the cursorjump back and forth between the end of the message an the home position.

to:

Using this code makes the cursor jump back and forth between the end of the message an the home position.

Restore
August 02, 2006, at 11:17 AM by Heather Dewey-Hagborg -
Changed lines 54-55 from:

1. include

to:

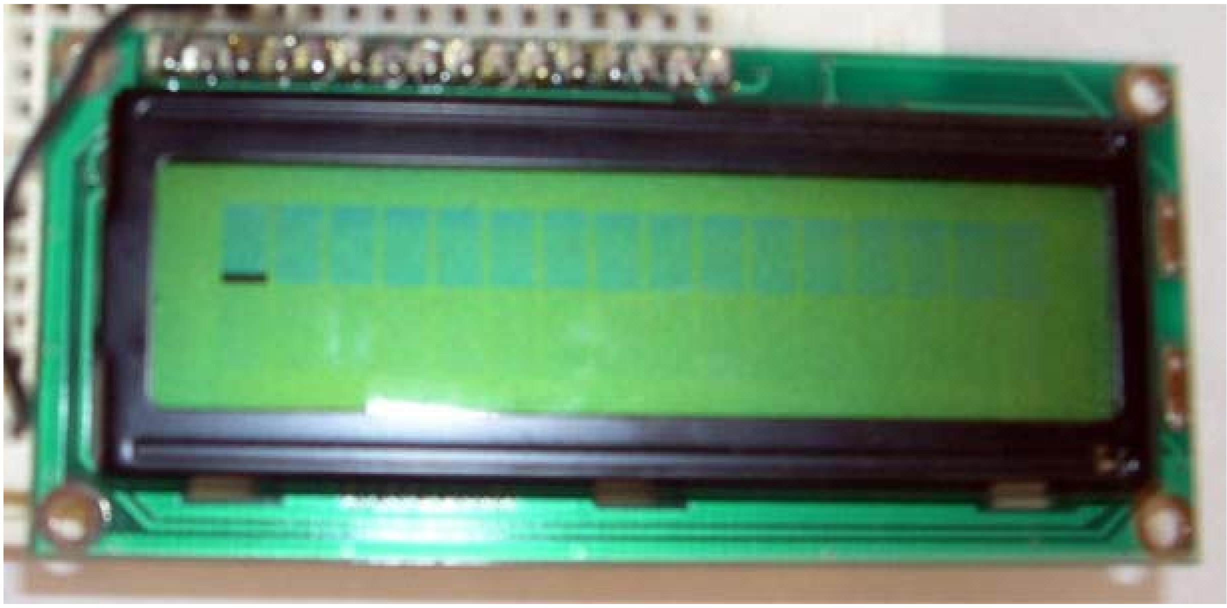
1. include <LiquidCrystal.h>

Added line 57:
Changed lines 70-72 from:

Now letÕs try something a little more interesting. Compile and upload the following code to the Arduino.

1. include

to:



Now let's try something a little more interesting. Compile and upload the following code to the Arduino.

[@

1. include <LiquidCrystal.h>

Added line 78:

Changed lines 91-92 from:

}

to:

@]

Changed line 97 from:

1. include

to:

1. include <LiquidCrystal.h>

Changed line 114 from:

1. include

to:

1. include <LiquidCrystal.h>

Restore
August 02, 2006, at 11:14 AM by Heather Dewey-Hagborg -
Changed lines 3-4 from:

In this tutorial you will control a Liquid Crystal Display (LCD) using the Arduino LiquidCrystal library. The library provides functions for accessing any LCD using the common HD44780 parallel interface chipset, such as those available from Sparkfun. It currently implements 8-bit control and one line display of 5x7 characters. Functions are provided to initialize the screen, to print characters and strings, to clear the screen, and to send commands directly to the HD44780 chip. This tutorial will walk you through the steps of wiring an LCD to an Arduino microcontroller board and implementing each of these functions.

to:

In this tutorial you will control a Liquid Crystal Display (LCD) using the Arduino LiquidCrystal library. The library provides functions for accessing any LCD using the common HD44780 parallel interface chipset, such as those available from Sparkfun. It currently implements 8-bit control and one line display of 5x7 characters. Functions are provided to initialize the screen, to print characters and strings, to clear the screen, and to send commands directly to the HD44780 chip. This tutorial will walk you through the steps of wiring an LCD to an Arduino microcontroller board and implementing each of these functions.

Added line 25:

[@

Changed lines 38-39 from:
to:

@]

Changed line 53 from:
to:

[@

Changed lines 57-58 from:

lcd.init(); digitalWrite(13,HIGH);

to:

```
lcd.init();
digitalWrite(13,HIGH);
```

Added line 60:
Changed line 62 from:

delay(1000);

to:

```
delay(1000);
```

Changed lines 64-65 from:

}

to:

@]

Restore
August 02, 2006, at 11:12 AM by Heather Dewey-Hagborg -
Changed lines 1-2 from:

Arduino Liquid Crystal Display Interface

to:

# Arduino Liquid Crystal Library LCD Interface

Changed lines 7-15 from:

```
    * Solderless breadboard
    * Hookup wire
    * Arduino Microcontoller Module
    * Potentiometer
    * Liquid Crystal Display (LCD) with HD44780 chip interface
    * Light emitting Diode (LED) - optional, for debugging
```

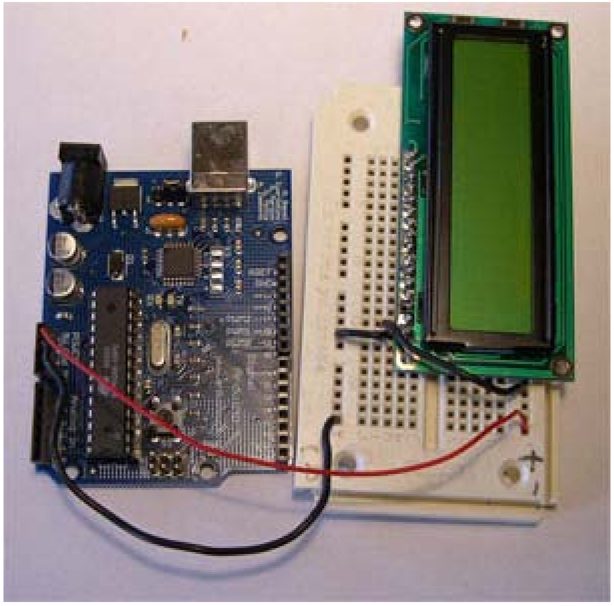
Prepare the breadboard

to:

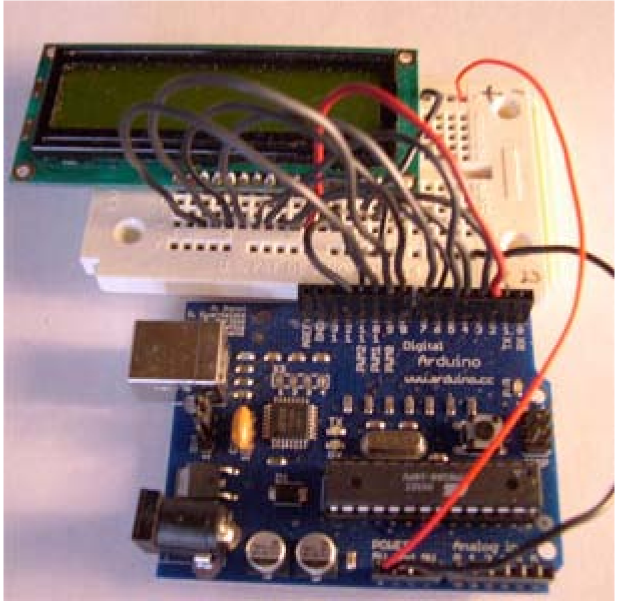
- Solderless breadboard
- Hookup wire
- Arduino Microcontoller Module
- Potentiometer
- Liquid Crystal Display (LCD) with HD44780 chip interface
- Light emitting Diode (LED) - optional, for debugging

### Prepare the breadboard

Added lines 22-23:



Added lines 38-39:

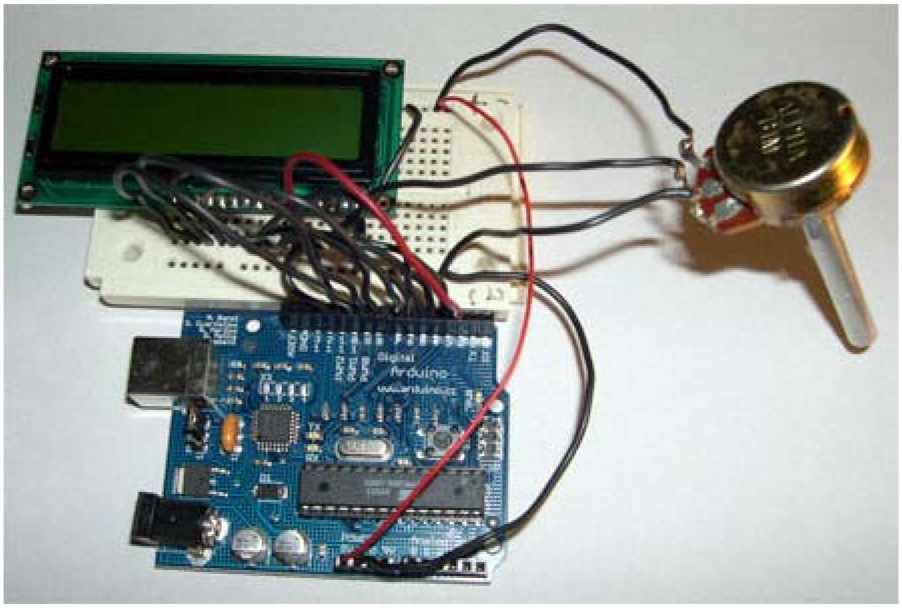


Changed lines 42-45 from:

Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground. Program the Arduino

First start by opening a new sketch in Arduino and saving it. Now go to the Sketch menu, scroll down to Òimport libraryÓ, and choose ÒLiquidCrystalÓ. The phrase #include should pop up at the top of your sketch.

to:



Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground.

### Program the Arduino

First start by opening a new sketch in Arduino and saving it. Now go to the Sketch menu, scroll down to "import library", and choose "LiquidCrystal". The phrase #include should pop up at the top of your sketch.

Restore
August 02, 2006, at 10:45 AM by Heather Dewey-Hagborg -
Changed line 95 from:
to:

[@

Changed lines 100-101 from:

lcd.init(); digitalWrite(13,HIGH);

to:

```
   lcd.init();
   digitalWrite(13,HIGH);
```

Changed lines 104-107 from:

lcd.commandWrite(2); delay(1000); lcd.printIn(string1); delay(1000);

to:

```
   lcd.commandWrite(2);
   delay(1000);
   lcd.printIn(string1);
   delay(1000);
```

Changed lines 109-110 from:

}

to:

@]

Restore
August 02, 2006, at 10:42 AM by Heather Dewey-Hagborg -
Added lines 1-111:

Arduino Liquid Crystal Display Interface

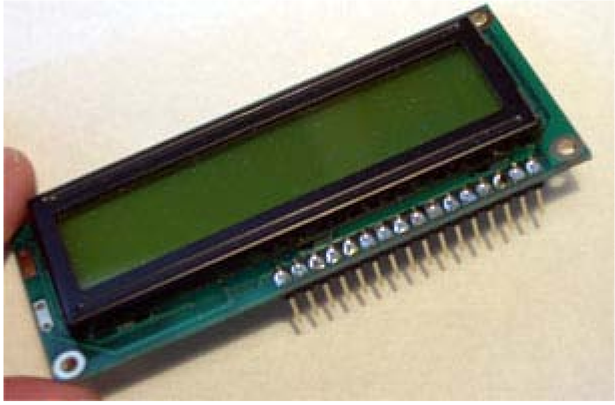
In this tutorial you will control a Liquid Crystal Display (LCD) using the Arduino LiquidCrystal library. The library provides functions for accessing any LCD using the common HD44780 parallel interface chipset, such as those available from Sparkfun. It currently implements 8-bit control and one line display of 5x7 characters. Functions are provided to initialize the screen, to print characters and strings, to clear the screen, and to send commands directly to the HD44780 chip. This tutorial will walk you through the steps of wiring an LCD to an Arduino microcontroller board and implementing each of these functions.

Materials needed:

```
    * Solderless breadboard
    * Hookup wire
    * Arduino Microcontoller Module
    * Potentiometer
    * Liquid Crystal Display (LCD) with HD44780 chip interface
    * Light emitting Diode (LED) - optional, for debugging
```

Prepare the breadboard

Solder a header to the LCD board if one is not present already.



Insert the LCD header into the breadboard and connect power and ground on the breadboard to power and ground from the microcontroller. On the Arduino module, use the 5V and any of the ground connections:

Connect wires from the breadboard to the arduino input sockets. Look at the datasheet for your LCD board to figure out which pins are where. The pinout is as follows: Arduino LCD 2 Enable 3 Data Bit 0 (DB0) 4 (DB1) 5 (DB2) 6 (DB3) 7 (DB4) 8 (DB5) 9 (DB6) 10 (DB7) 11 Read/Write (RW) 12 Register Select (RS)

Connect a potentiometer a a voltage divider between 5V, Ground, and the contrast adjustment pin on your LCD.

Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground. Program the Arduino

First start by opening a new sketch in Arduino and saving it. Now go to the Sketch menu, scroll down to Òimport libraryÓ, and choose ÒLiquidCrystalÓ. The phrase #include should pop up at the top of your sketch.

The first program we are going to try is simply for calibration and debugging. Copy the following code into your sketch, compile and upload to the Arduino.

1. include

```
LiquidCrystal lcd = LiquidCrystal(); void setup(void){ lcd.init(); digitalWrite(13,HIGH); } void loop(void){ delay(1000); } }
```

If all went as planned both the LCD and the LED should turn on. Now you can use the potentiometer to adjust the contrast on the LCD until you can clearly see a cursor at the beginning of the first line.

Now letÕs try something a little more interesting. Compile and upload the following code to the Arduino.

1. include

```
LiquidCrystal lcd = LiquidCrystal(); void setup(void){ lcd.init(); digitalWrite(13,HIGH); } void loop(void){ lcd.clear();
delay(1000); lcd.print('a'); lcd.print('b'); lcd.print('c'); delay(1000); } }
```

This time you should see the letters a b and c appear and clear from the display in an endless loop.

This is all great fun, but who really wants to type out each letter of a message indivually? Enter the printIn() function. Simply initialize a string, pass it to printIn(), and now we have ourselves a proper hello world program.

1. include

```
LiquidCrystal lcd = LiquidCrystal(); char string1[] = "Hello!"; void setup(void){ lcd.init(); digitalWrite(13,HIGH); } void
loop(void){ lcd.clear(); delay(1000); lcd.printIn(string1); delay(1000); } }
```

Finally, you should know there is a lot of functionality in the HD44780 chip interface that is not drawn out into Arduino functions. If you are feeling ambitious glance over the datasheet and try out some of the direct commands using the commandWrite() function. For example, commandWrite(2) tells the board to move the cursor back to starting position.

1. include

```
LiquidCrystal lcd = LiquidCrystal(); char string1[] = "Hello!"; void setup(void){ lcd.init(); digitalWrite(13,HIGH); } void
loop(void){ lcd.commandWrite(2); delay(1000); lcd.printIn(string1); delay(1000); } }
```

Using this code makes the cursorjump back and forth between the end of the message an the home position.

Restore

# Arduino Liquid Crystal Library LCD Interface

In this tutorial you will control a Liquid Crystal Display (LCD) using the Arduino LiquidCrystal library. The library provides functions for accessing any LCD using the common HD44780 parallel interface chipset, such as those available from Sparkfun. It currently implements 8-bit control and one line display of 5x7 characters. Functions are provided to initialize the screen, to print characters and strings, to clear the screen, and to send commands directly to the HD44780 chip. This tutorial will walk you through the steps of wiring an LCD to an Arduino microcontroller board and implementing each of these functions.

Materials needed:

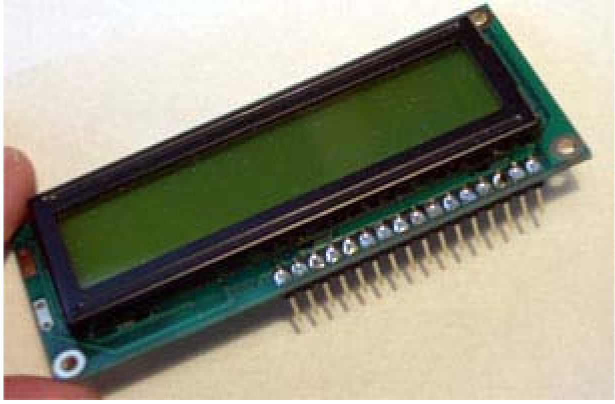
- Solderless breadboard
- Hookup wire
- Arduino Microcontoller Module
- Potentiometer
- Liquid Crystal Display (LCD) with HD44780 chip interface
- Light emitting Diode (LED) - optional, for debugging

## Install the Library

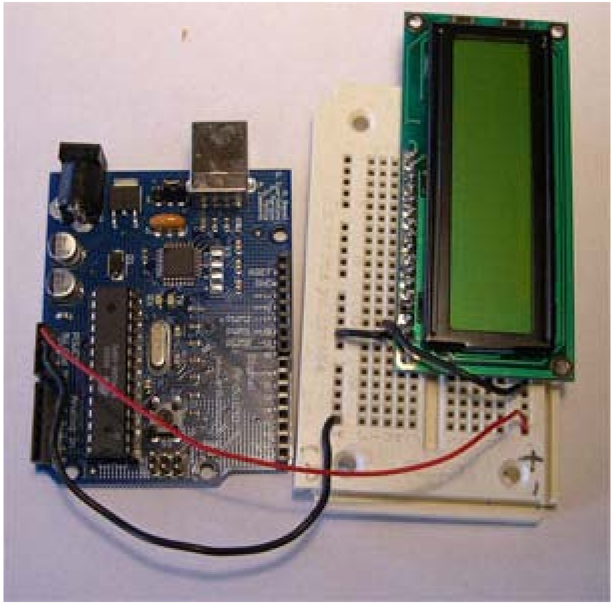
For a basic explanation of how libraries work in Arduino read the library page. Download the LiquidCrystal library here. Unzip the files and place the whole LiquidCrystal folder inside your arduino-0004\lib\targets\libraries folder. Start the Arduino program and check to make sure LiquidCrystal is now available as an option in the Sketch menu under "Import Library".

## Prepare the breadboard

Solder a header to the LCD board if one is not present already.



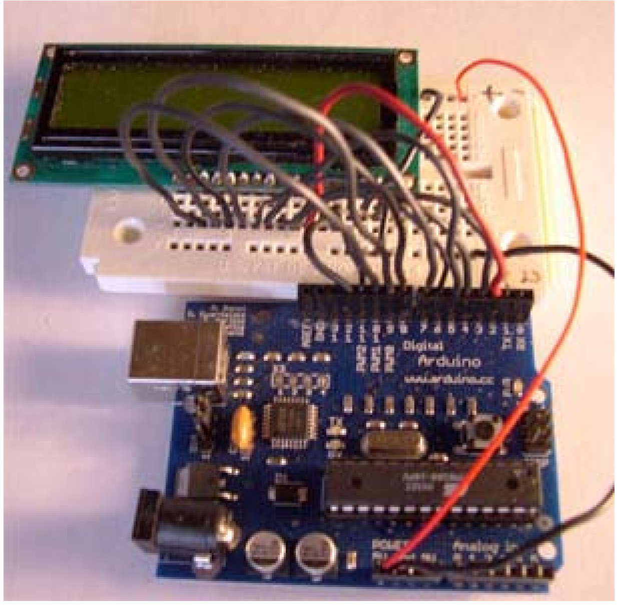
Insert the LCD header into the breadboard and connect power and ground on the breadboard to power and ground from the microcontroller. On the Arduino module, use the 5V and any of the ground connections.



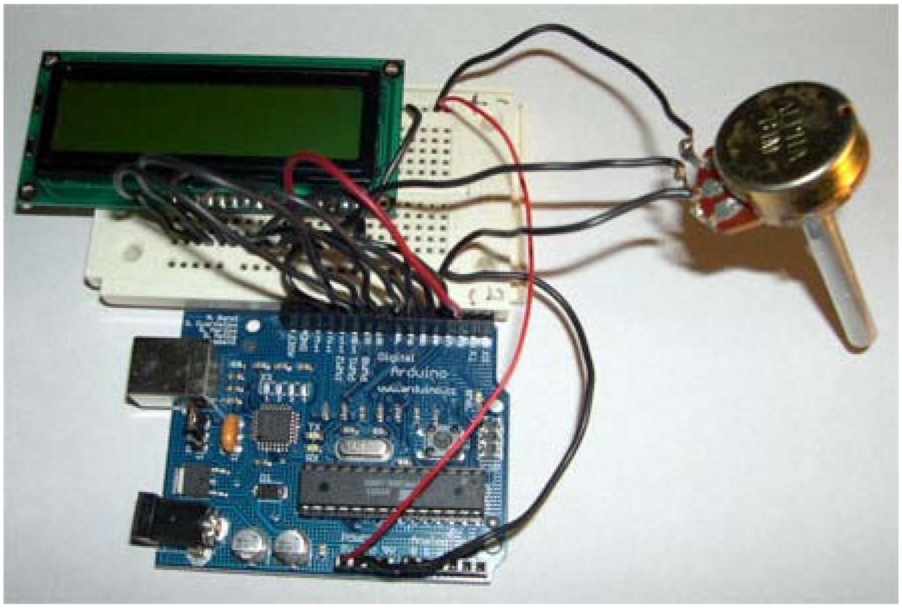
Connect wires from the breadboard to the arduino input sockets. It is a lot of wires, so keep them as short and tidy as possible. Look at the datasheet for your LCD board to figure out which pins are where. Make sure to take note of whether the pin view is from the front or back side of the LCD board, you don't want to get your pins reversed!

The pinout is as follows:

| Arduino | LCD |
|---|---|
| 2 | Enable |
| 3 | Data Bit 0 (DB0) |
| 4 | (DB1) |
| 5 | (DB2) |
| 6 | (DB3) |
| 7 | (DB4) |
| 8 | (DB5) |
| 9 | (DB6) |
| 10 | (DB7) |
| 11 | Read/Write (RW) |
| 12 | Register Select (RS) |



Connect a potentiometer a a voltage divider between 5V, Ground, and the contrast adjustment pin on your LCD.



Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground.

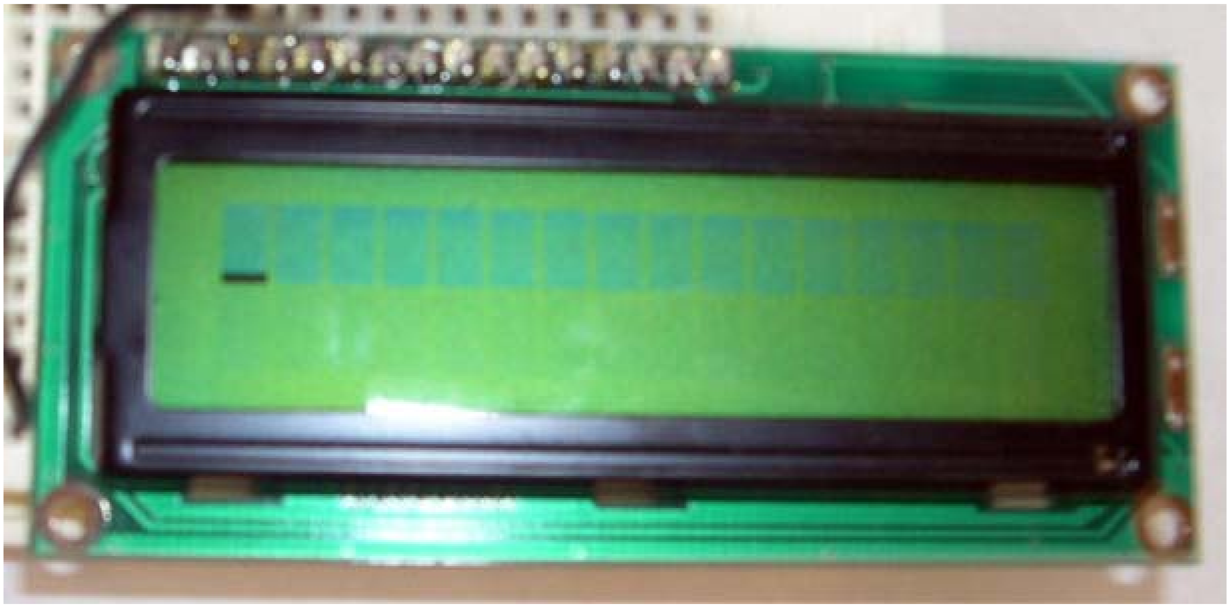
## Program the Arduino

First start by opening a new sketch in Arduino and saving it. Now go to the Sketch menu, scroll down to "import library", and choose "LiquidCrystal". The phrase #include <LiquidCrystal.h> should pop up at the top of your sketch.

The first program we are going to try is simply for calibration and debugging. Copy the following code into your sketch, compile and upload to the Arduino.

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

void setup(void){
  lcd.init(); //initialize the LCD
  digitalWrite(13,HIGH); //turn on an LED for debugging
}

void loop(void){
  delay(1000); //repeat forever
}
```

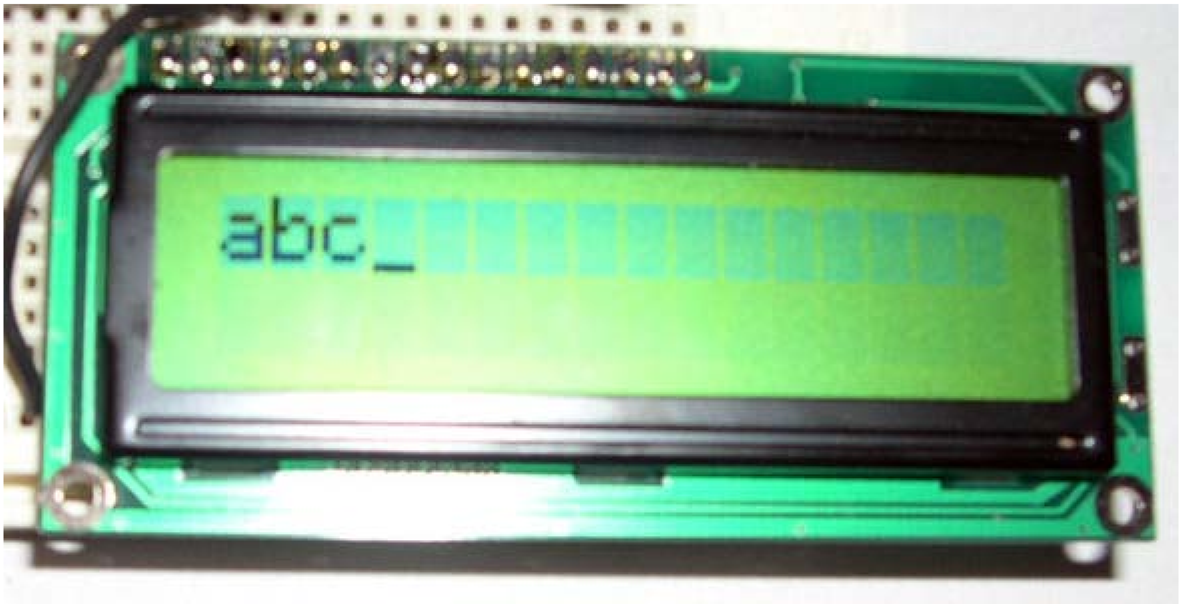
If all went as planned both the LCD and the LED should turn on. Now you can use the potentiometer to adjust the contrast on the LCD until you can clearly see a cursor at the beginning of the first line. If you turn the potentiometer too far in one direction black blocks will appear. Too far in the other direction everything will fade from the display. There should be a small spot in the middle where the cursor appears crisp and dark.



Now let's try something a little more interesting. Compile and upload the following code to the Arduino.

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD

void setup(void){
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
}

void loop(void){
   lcd.clear(); //clear the display
   delay(1000); //delay 1000 ms to view change
   lcd.print('a'); //send individual letters to the LCD
   lcd.print('b');
   lcd.print('c');
   delay(1000);//delay 1000 ms to view change

} //repeat forever
```

This time you should see the letters a b and c appear and clear from the display in an endless loop.



This is all great fun, but who really wants to type out each letter of a message indivually? Enter the println()

function. Simply initialize a string, pass it to printIn(), and now we have ourselves a proper hello world program.

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD
char string1[] = "Hello!"; //variable to store the string "Hello!"

void setup(void){
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
}
void loop(void){
   lcd.clear(); //clear the display
   delay(1000); //delay 1000 ms to view change
   lcd.printIn(string1); //send the string to the LCD
   delay(1000); //delay 1000 ms to view change
} //repeat forever
```

Finally, you should know there is a lot of functionality in the HD44780 chip interface that is not drawn out into Arduino functions. If you are feeling ambitious glance over the datasheet and try out some of the direct commands using the commandWrite() function. For example, commandWrite(2) tells the board to move the cursor back to starting position. Here is an example:

```
#include <LiquidCrystal.h> //include LiquidCrystal library

LiquidCrystal lcd = LiquidCrystal(); //create a LiquidCrystal object to control an LCD
char string1[] = "Hello!"; //variable to store the string "Hello!"

void setup(void){
        lcd.init(); //initialize the LCD
    digitalWrite(13,HIGH); //turn on an LED for debugging
}
void loop(void){
   lcd.commandWrite(2); //bring the cursor to the starting position
   delay(1000); //delay 1000 ms to view change
   lcd.printIn(string1); //send the string to the LCD
   delay(1000); //delay 1000 ms to view change
} //repeat forever
```

This code makes the cursor jump back and forth between the end of the message an the home position.

To interface an LCD directly in Arduino code see this example.

*LCD interface library and tutorial by Heather Dewey-Hagborg*

## Tutorial.StepperUnipolar History

Hide minor edits - Show changes to markup

December 16, 2005, at 12:55 PM by 195.178.229.25 -
Deleted line 107:

```
  pinMode(ledPin, OUTPUT);
```

Restore
October 21, 2005, at 05:38 AM by 195.178.229.25 -
Changed lines 3-4 from:

This page shows two examples on how to drive a bipolar stepper motor. These motors can be found in old floppy drives and are easy to control. The one we use has 6 connectors of which one is power (VCC) and the other four are used to drive the motor sending synchronous signals.

to:

This page shows two examples on how to drive a unipolar stepper motor. These motors can be found in old floppy drives and are easy to control. The one we use has 6 connectors of which one is power (VCC) and the other four are used to drive the motor sending synchronous signals.

Changed line 25 from:

```
 * It is a bipolar stepper motor with 5 wires:
```

to:

```
 * It is a unipolar stepper motor with 5 wires:
```

Changed lines 76-77 from:

### Example 2: Stepper Bipolar Advanced

to:

### Example 2: Stepper Unipolar Advanced

Changed lines 79-80 from:

/* Stepper Bipolar Advanced

```
 * ------------------------
```

to:

/* Stepper Unipolar Advanced

```
 * -------------------------
```

Changed line 88 from:

```
 * It is a bipolar stepper motor with 5 wires:
```

to:

```
 * It is a unipolar stepper motor with 5 wires:
```

Restore
October 21, 2005, at 05:37 AM by 195.178.229.25 -
Added lines 1-160:

## Unipolar Stepper Motor

This page shows two examples on how to drive a bipolar stepper motor. These motors can be found in old floppy drives and are easy to control. The one we use has 6 connectors of which one is power (VCC) and the other four are used to drive the motor sending synchronous signals.

The first example is the basic code to make the motor spin in one direction. It is aiming those that have no knowledge in how to control stepper motors. The second example is coded in a more complex way, but allows to make the motor spin at different speeds, in both directions, and controlling both from a potentiometer.

The prototyping board has been populated with a 10K potentiomenter that we connect to an analog input, and a ULN2003A driver. This chip has a bunch of transistors embedded in a single housing. It allows the connection of devices and components that need much higher current than the ones that the ATMEGA8 from our Arduino board can offer.

http://static.flickr.com/32/54357295_756c131217.jpg

*Picture of a protoboard supporting the ULN2003A and a potentiometer*

### Example 1: Simple example

```
/* Stepper Copal
 * -------------
 *
 * Program to drive a stepper motor coming from a 5'25 disk drive
 * according to the documentation I found, this stepper: "[...] motor
 * made by Copal Electronics, with 1.8 degrees per step and 96 ohms
 * per winding, with center taps brought out to separate leads [...]"
 * [http://www.cs.uiowa.edu/~jones/step/example.html]
 *
 * It is a bipolar stepper motor with 5 wires:
 *
 * - red: power connector, I have it at 5V and works fine
 * - orange and black: coil 1
 * - brown and yellow: coil 2
 *
 * (cleft) 2005 DojoDave for K3
 * http://www.0j0.org | http://arduino.berlios.de
 *
 * @author: David Cuartielles
 * @date: 20 Oct. 2005
 */

int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;
int delayTime = 500;

void setup() {
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
}

void loop() {
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  delay(delayTime);
}
```

### Example 2: Stepper Bipolar Advanced

```
/* Stepper Bipolar Advanced
 * ------------------------
 *
 * Program to drive a stepper motor coming from a 5'25 disk drive
 * according to the documentation I found, this stepper: "[...] motor
 * made by Copal Electronics, with 1.8 degrees per step and 96 ohms
 * per winding, with center taps brought out to separate leads [...]"
 * [http://www.cs.uiowa.edu/~jones/step/example.html]
 *
 * It is a bipolar stepper motor with 5 wires:
 *
 * - red: power connector, I have it at 5V and works fine
 * - orange and black: coil 1
 * - brown and yellow: coil 2
 *
 * (cleft) 2005 DojoDave for K3
 * http://www.0j0.org | http://arduino.berlios.de
 *
 * @author: David Cuartielles
 * @date: 20 Oct. 2005
 */

int motorPins[] = {8, 9, 10, 11};
int count = 0;
int count2 = 0;
int delayTime = 500;
int val = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  for (count = 0; count < 4; count++) {
    pinMode(motorPins[count], OUTPUT);
  }
}

void moveForward() {
  if ((count2 == 0) || (count2 == 1)) {
    count2 = 16;
  }
  count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[count], count2>>count&0x01);
  }
  delay(delayTime);
}

void moveBackward() {
  if ((count2 == 0) || (count2 == 1)) {
    count2 = 16;
  }
  count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[3 - count], count2>>count&0x01);
  }
  delay(delayTime);
}

void loop() {
  val = analogRead(0);
  if (val > 540) {
    // move faster the higher the value from the potentiometer
    delayTime = 2048 - 1024 * val / 512 + 1;
    moveForward();
  } else if (val < 480) {
    // move faster the lower the value from the potentiometer
    delayTime = 1024 * val / 512 + 1;
    moveBackward();
  } else {
    delayTime = 1024;
  }
}
```

## References

In order to work out this example, we have been looking into quite a lot of documentation. The following links may be useful for you to visit in order to understand the theory underlying behind stepper motors:

- information about the motor we are using - here

- basic explanation about steppers - here

- good PDF with basic information - here

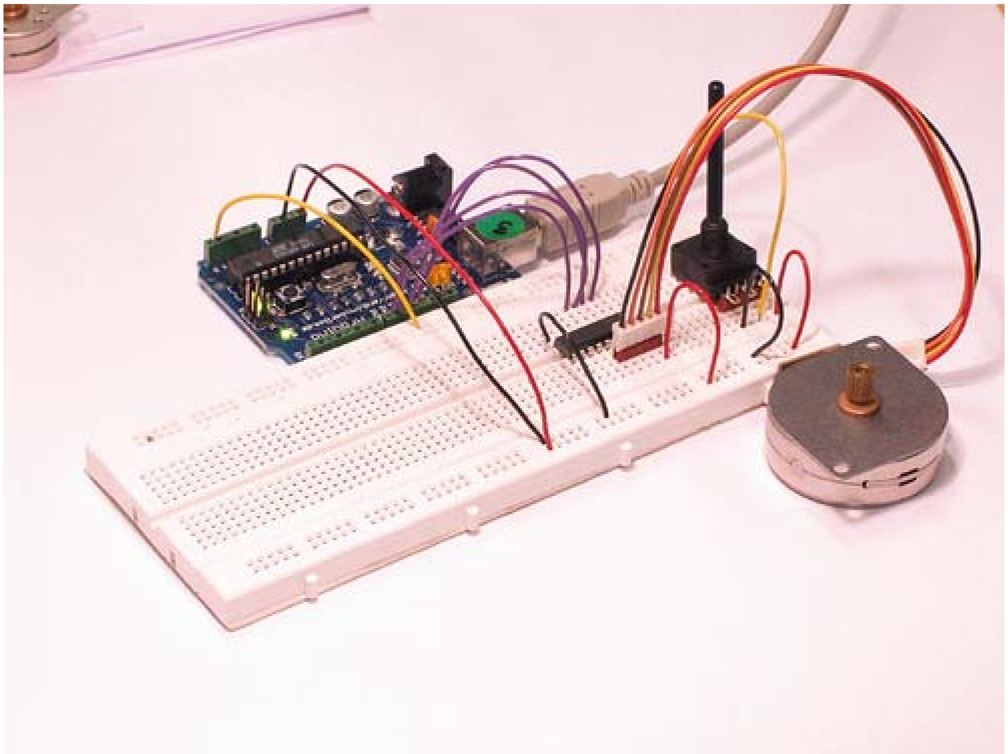
Restore

# Unipolar Stepper Motor

This page shows two examples on how to drive a unipolar stepper motor. These motors can be found in old floppy drives and are easy to control. The one we use has 6 connectors of which one is power (VCC) and the other four are used to drive the motor sending synchronous signals.

The first example is the basic code to make the motor spin in one direction. It is aiming those that have no knowledge in how to control stepper motors. The second example is coded in a more complex way, but allows to make the motor spin at different speeds, in both directions, and controlling both from a potentiometer.

The prototyping board has been populated with a 10K potentiomenter that we connect to an analog input, and a ULN2003A driver. This chip has a bunch of transistors embedded in a single housing. It allows the connection of devices and components that need much higher current than the ones that the ATMEGA8 from our Arduino board can offer.



*Picture of a protoboard supporting the ULN2003A and a potentiometer*

## Example 1: Simple example

```
/* Stepper Copal
 * -------------
 *
 * Program to drive a stepper motor coming from a 5'25 disk drive
 * according to the documentation I found, this stepper: "[...] motor
 * made by Copal Electronics, with 1.8 degrees per step and 96 ohms
 * per winding, with center taps brought out to separate leads [...]"
 * [http://www.cs.uiowa.edu/~jones/step/example.html]
 *
 * It is a unipolar stepper motor with 5 wires:
 *
 * - red: power connector, I have it at 5V and works fine
 * - orange and black: coil 1
 * - brown and yellow: coil 2
 *
 * (cleft) 2005 DojoDave for K3
 * http://www.0j0.org | http://arduino.berlios.de
 *
 * @author: David Cuartielles
 * @date: 20 Oct. 2005
 */

int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;
int delayTime = 500;

void setup() {
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
}

void loop() {
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  delay(delayTime);
}
```

## Example 2: Stepper Unipolar Advanced

```
/* Stepper Unipolar Advanced
 * -------------------------
 *
 * Program to drive a stepper motor coming from a 5'25 disk drive
 * according to the documentation I found, this stepper: "[...] motor
 * made by Copal Electronics, with 1.8 degrees per step and 96 ohms
 * per winding, with center taps brought out to separate leads [...]"
 * [http://www.cs.uiowa.edu/~jones/step/example.html]
 *
 * It is a unipolar stepper motor with 5 wires:
 *
 * - red: power connector, I have it at 5V and works fine
 * - orange and black: coil 1
 * - brown and yellow: coil 2
 *
 * (cleft) 2005 DojoDave for K3
 * http://www.0j0.org | http://arduino.berlios.de
 *
 * @author: David Cuartielles
 * @date: 20 Oct. 2005
 */

int motorPins[] = {8, 9, 10, 11};
int count = 0;
int count2 = 0;
int delayTime = 500;
int val = 0;

void setup() {
  for (count = 0; count < 4; count++) {
    pinMode(motorPins[count], OUTPUT);
  }
}

void moveForward() {
  if ((count2 == 0) || (count2 == 1)) {
    count2 = 16;
  }
  count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[count], count2>>count&0x01);
  }
  delay(delayTime);
}

void moveBackward() {
  if ((count2 == 0) || (count2 == 1)) {
    count2 = 16;
  }
  count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[3 - count], count2>>count&0x01);
  }
  delay(delayTime);
}

void loop() {
  val = analogRead(0);
  if (val > 540) {
    // move faster the higher the value from the potentiometer
    delayTime = 2048 - 1024 * val / 512 + 1;
    moveForward();
  } else if (val < 480) {
    // move faster the lower the value from the potentiometer
    delayTime = 1024 * val / 512 + 1;
    moveBackward();
  } else {
    delayTime = 1024;
  }
}
```

## References

In order to work out this example, we have been looking into quite a lot of documentation. The following links may be useful for you to visit in order to understand the theory underlying behind stepper motors:

- information about the motor we are using - here

- basic explanation about steppers - here

- good PDF with basic information - here

## Tutorial.DMXMaster History

Hide minor edits - Show changes to markup

January 30, 2007, at 03:35 PM by David A. Mellis -
Changed lines 2-176 from:

full tutorial coming soon

```
/* DMX Shift Out
 * -------------
 *
 * Shifts data in DMX format out to DMX enabled devices
 * it is extremely restrictive in terms of timing. Therefore
 * the program will stop the interrupts when sending data
 *
 * (cleft) 2006 by Tomek Ness and D. Cuartielles
 * K3 - School of Arts and Communication
 * <http://www.arduino.cc>
 * <http://www.mah.se/k3>
 *
 * @date: 2006-01-19
 * @idea: Tomek Ness
 * @code: D. Cuartielles and Tomek Ness
 * @acknowledgements: Johny Lowgren for his DMX devices
 *
 */

int sig = 3;           // signal (plus / dmx pin 3)
int sigI = 2;          // signal inversion (minus / dmx pin 2)
int count = 0;


/* Sends a DMX byte out on a pin.  Assumes a 16 MHz clock.
 * Disables interrupts, which will disrupt the millis() function if used
 * too frequently. */
void shiftDmxOut(int pin, int ipin, int theByte)
{
        int theDelay = 1;
        int count = 0;
        int portNumber = port_to_output[digitalPinToPort(pin)];
        int pinNumber = digitalPinToBit(pin);
        int iPortNumber = port_to_output[digitalPinToPort(ipin)];
        int iPinNumber = digitalPinToBit(ipin);

        // the first thing we do is to write te pin to high
        // it will be the mark between bytes. It may be also
        // high from before
        _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
        _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        delayMicroseconds(20);

        if (digitalPinToPort(pin) != NOT_A_PIN) {
                // If the pin that support PWM output, we need to turn it off
                // before doing a digital write.

                if (analogOutPinToBit(pin) == 1)
                        timer1PWMAOff();

                if (analogOutPinToBit(pin) == 2)
                        timer1PWMBOff();
        }

        // disable interrupts, otherwise the timer 0 overflow interrupt that
        // tracks milliseconds will make us delay longer than we want.
        cli();

        // DMX starts with a start-bit that must always be zero
        _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
        _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        delayMicroseconds(theDelay);
        delayMicroseconds(theDelay);

        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;

        // the last thing we do is to write te pin to high
        // it will be the mark between bytes.
        _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);

        // reenable interrupts.
        sei();
}

void setup() {
  pinMode(sig, OUTPUT);
  pinMode(sigI, OUTPUT);
}

void loop()  {
   digitalWrite(sig, LOW);
   digitalWrite(sigI, HIGH);
   delay(10);

   //sending the start byte
   shiftDmxOut(3,2,0);

   //set all adresses/channels to 60%
   for (count = 1; count<=512; count++){
     shiftDmxOut(3,2,155);
   }
 }
```

to:

Please see this updated tutorial on the playground.

Restore
January 19, 2006, at 05:03 PM by 195.178.229.101 -
Added lines 1-176:

## DMX Master Device

full tutorial coming soon

```
/* DMX Shift Out
 * -------------
 *
 * Shifts data in DMX format out to DMX enabled devices
 * it is extremely restrictive in terms of timing. Therefore
 * the program will stop the interrupts when sending data
 *
 * (cleft) 2006 by Tomek Ness and D. Cuartielles
 * K3 - School of Arts and Communication
 * <http://www.arduino.cc>
 * <http://www.mah.se/k3>
 *
 * @date: 2006-01-19
 * @idea: Tomek Ness
 * @code: D. Cuartielles and Tomek Ness
 * @acknowledgements: Johny Lowgren for his DMX devices
 *
 */

int sig = 3;           // signal (plus / dmx pin 3)
int sigI = 2;          // signal inversion (minus / dmx pin 2)
int count = 0;


/* Sends a DMX byte out on a pin.  Assumes a 16 MHz clock.
 * Disables interrupts, which will disrupt the millis() function if used
 * too frequently. */
void shiftDmxOut(int pin, int ipin, int theByte)
{
        int theDelay = 1;
        int count = 0;
        int portNumber = port_to_output[digitalPinToPort(pin)];
        int pinNumber = digitalPinToBit(pin);
        int iPortNumber = port_to_output[digitalPinToPort(ipin)];
        int iPinNumber = digitalPinToBit(ipin);

        // the first thing we do is to write te pin to high
        // it will be the mark between bytes. It may be also
        // high from before
        _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
        _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        delayMicroseconds(20);

        if (digitalPinToPort(pin) != NOT_A_PIN) {
                // If the pin that support PWM output, we need to turn it off
                // before doing a digital write.

                if (analogOutPinToBit(pin) == 1)
                        timer1PWMAOff();

                if (analogOutPinToBit(pin) == 2)
                        timer1PWMBOff();
        }

        // disable interrupts, otherwise the timer 0 overflow interrupt that
```

```
    // tracks milliseconds will make us delay longer than we want.
    cli();

    // DMX starts with a start-bit that must always be zero
    _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
    _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
    delayMicroseconds(theDelay);
    delayMicroseconds(theDelay);

    if (theByte & 01) {
      _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
    }
    else {
      _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
    }
    delayMicroseconds(theDelay);
    theByte>>=1;
    if (theByte & 01) {
      _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
    }
    else {
      _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
    }
    delayMicroseconds(theDelay);
    theByte>>=1;
    if (theByte & 01) {
      _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
    }
    else {
      _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
    }
    delayMicroseconds(theDelay);
    theByte>>=1;
    if (theByte & 01) {
      _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
    }
    else {
      _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
    }
    delayMicroseconds(theDelay);
    theByte>>=1;
    if (theByte & 01) {
      _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
    }
    else {
      _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
    }
    delayMicroseconds(theDelay);
    theByte>>=1;
    if (theByte & 01) {
      _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
      _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
    }
    else {
       SFR BYTE( SFR IO8(portNumber)) &= ~ BV(pinNumber);
```

```
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;
        if (theByte & 01) {
          _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) &= ~_BV(iPinNumber);
        }
        else {
          _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
          _SFR_BYTE(_SFR_IO8(iPortNumber)) |= _BV(iPinNumber);
        }
        delayMicroseconds(theDelay);
        theByte>>=1;

        // the last thing we do is to write te pin to high
        // it will be the mark between bytes.
        _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);

        // reenable interrupts.
        sei();
}

void setup() {
  pinMode(sig, OUTPUT);
  pinMode(sigI, OUTPUT);
}

void loop()  {
   digitalWrite(sig, LOW);
   digitalWrite(sigI, HIGH);
   delay(10);

   //sending the start byte
   shiftDmxOut(3,2,0);

   //set all adresses/channels to 60%
   for (count = 1; count<=512; count++){
     shiftDmxOut(3,2,155);
   }
 }
```

Restore

# DMX Master Device

Please see this updated tutorial on the playground.

Bit masks are used to access specific bits in a byte of data. This is often useful as a method of iteration, for example when sending a byte of data serially out a single pin. In this example the pin needs to change it's state from high to low for each bit in the byte to be transmitted. This is accomplished using what are known as bitwise operations and a bit mask.

Bitwise operations perform logical functions that take affect on the bit level. Standard bitwise operations include AND (&) OR (|) Left Shift (<<) and Right Shift (>>).

The AND (&) operator will result in a 1 at each bit position where both input values were 1. For example:

```
    x:  10001101
    y:  01010111
x & y:  00000101
```

The OR (|) operator (also known as Inclusive Or) will result in a 1 at each bit position where either input values were 1. For example:

```
    x:  10001101
    y:  01010111
x | y:  11011111
```

The Left Shift (<<) operator will shift a value to the left the specified number of times. For example:

```
        y = 1010
        x = y << 1
yields: x = 0100
```

All the bits in the byte get shifted one position to the left and the bit on the left end drops off.

The Right Shift (>>) operator works identically to left shift except that it shifts the value to the right the specified number of times For example:

```
        y = 1010
        x = y >> 1
yields: x = 0101
```

All the bits in the byte get shifted one position to the right and the bit on the right end drops off.

For a practical example, let's take the value 170, binary 10101010. To pulse this value out of pin 7 the code might look as follows:

```
byte transmit = 7; //define our transmit pin
byte data = 170; //value to transmit, binary 10101010
byte mask = 1; //our bitmask
byte bitDelay = 100;

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

void loop()
{
  for (mask = 00000001; mask>0; mask <<= 1) { //iterate through bit mask
    if (data & mask){ // if bitwise AND resolves to true
      digitalWrite(transmit,HIGH); // send 1
    }
    else{ //if bitwise and resolves to false
      digitalWrite(transmit,LOW); // send 0
    }
    delayMicroseconds(bitDelay); //delay
  }
}
```

Here we use a FOR loop to iterate through a bit mask value, shifting the value one position left each time through the loop. In this example we use the <<= operator which is exactly like the << operator except that it compacts the statement `mask = mask << 1` into a shorter line. We then perform a bitwise AND operation on the value and the bitmask. This way as the bitmask shifts left through each position in the byte it will be compared against each bit in the byte we are sending sequentially and can then be used to set our output pin either high or low accordingly. So in this example, first time through the loop the mask = 00000001 and the value = 10101010 so our operation looks like:

```
  00000001
& 10101010
  ________
  00000000
```

And our output pin gets set to 0. Second time throught he loop the mask = 00000010, so our operation looks like:

```
  00000010
& 10101010
  ________
  00000010
```

And our output pin gets set to 1. The loop will continue to iterate through each bit in the mask until the 1 gets shifted left off the end of the 8 bits and our mask =0. Then all 8 bits have been sent and our loop exits.

## Tutorial.SoftwareSerial History

Hide minor edits - Show changes to markup

February 26, 2007, at 10:40 AM by David A. Mellis -
Added lines 3-4:

**Note:** *If you just want to use a software serial interface, see the SoftwareSerial library included with Arduino 0007 and later. Read on if you'd like to know how that library works.*

Restore
September 05, 2006, at 12:55 PM by Heather Dewey-Hagborg -
Changed lines 214-216 from:

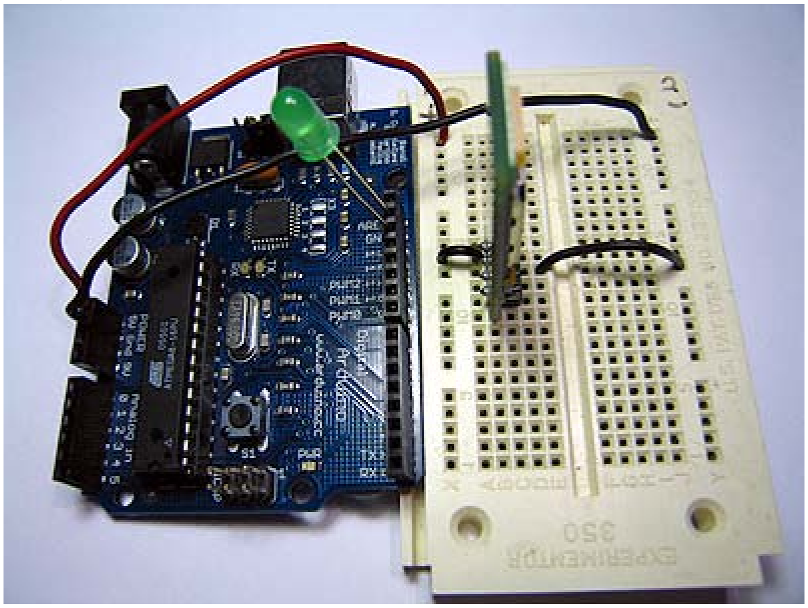
}@]

to:

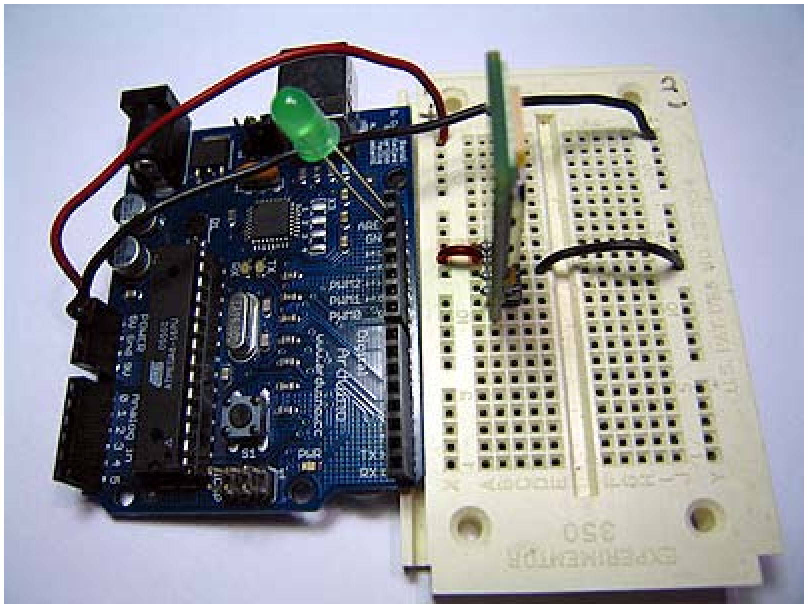
}@]

*code and tutorial by Heather Dewey-Hagborg*

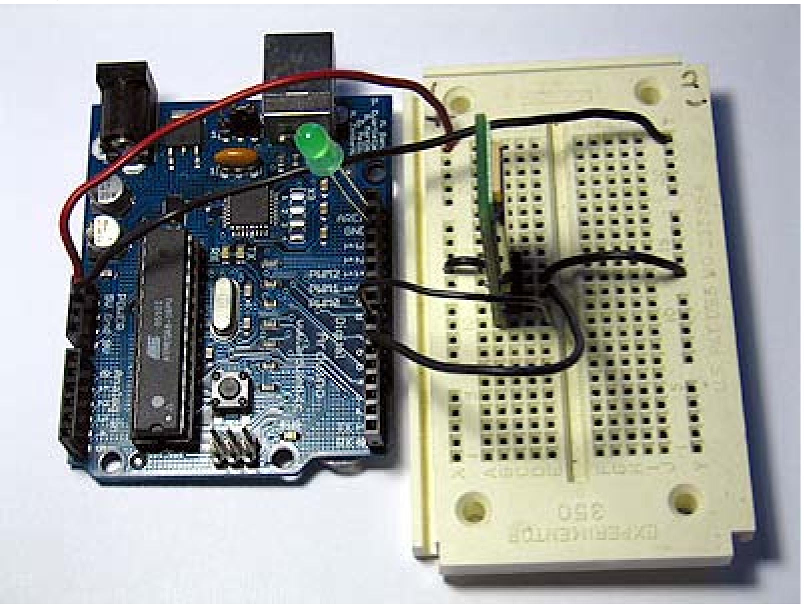
Restore
August 29, 2006, at 12:46 PM by Heather Dewey-Hagborg -
Changed lines 53-54 from:



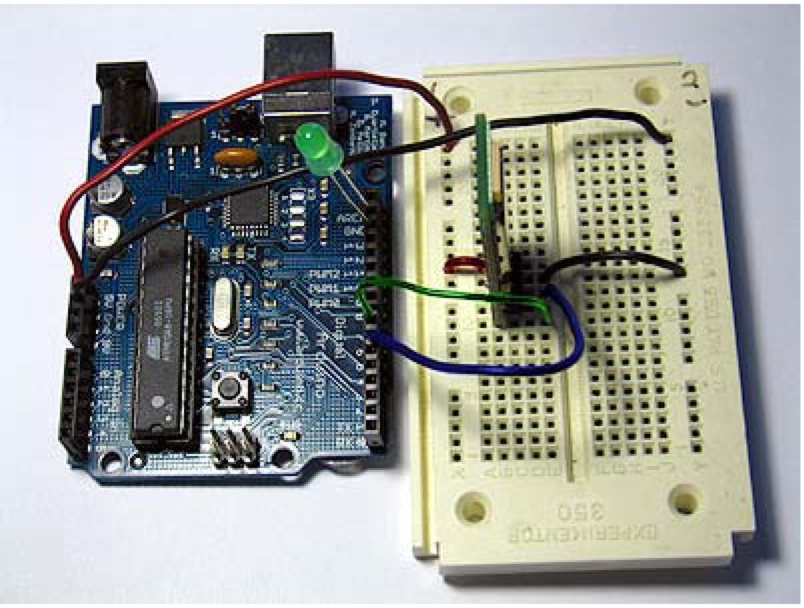
to:



Changed lines 57-58 from:



to:



Restore

August 29, 2006, at 12:34 PM by Heather Dewey-Hagborg -

Changed lines 11-12 from:

Returns a byte long integer value

to:

Returns a byte long integer value from the software serial connection

Restore
August 29, 2006, at 12:25 PM by Heather Dewey-Hagborg -
Changed line 10 from:

### SWread();

to:

**SWread();**

Changed line 20 from:

### SWprint();

to:

**SWprint();**

Restore
August 29, 2006, at 12:19 PM by Heather Dewey-Hagborg -
Added lines 10-13:

### SWread();

Returns a byte long integer value

Example:

Changed lines 15-16 from:

SWread();

to:

byte RXval; RXval = SWread();

Changed lines 18-19 from:

Returns a byte long integer value

to:

### SWprint();

Sends a byte long integer value out the software serial connection

Added line 24:
Changed lines 26-27 from:

byte RXval; RXval = SWread();

to:

byte TXval = 'h'; byte TXval2 = 126; SWprint(TXval); SWprint(TXval2);

Added line 32:

Definitions Needed:

Changed lines 34-37 from:

SWprint();

to:

1. define bit9600Delay 84
2. define halfBit9600Delay 42

3. define bit4800Delay 188
4. define halfBit4800Delay 94

Deleted lines 38-55:

Sends a byte long integer value out the software serial connection

Example:

```
byte TXval = 'h';
byte TXval2 = 126;
SWprint(TXval);
SWprint(TXval2);
```

Definitions Needed:

```
#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94
```

Restore
August 29, 2006, at 12:18 PM by Heather Dewey-Hagborg -
Added line 10:

[@

Added line 12:

@]

Added line 16:

[@

Changed lines 19-21 from:
to:

@]

[@

Added line 23:

@]

Added lines 27-28:

[@

Changed lines 33-34 from:
to:

@]

Changed line 36 from:
to:

[@

Changed line 41 from:
to:

@]

Restore
August 29, 2006, at 12:15 PM by Heather Dewey-Hagborg -
Changed lines 5-6 from:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page. For a good explanation of serial communication see Wikipedia.

to:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page. For a good explanation of serial communication see Wikipedia. The software serial connection can run at 4800 baud or 9600 baud reliably.

Functions Available:

SWread(); Returns a byte long integer value

Example: byte RXval; RXval = SWread();

SWprint(); Sends a byte long integer value out the software serial connection

Example: byte TXval = 'h'; byte TXval2 = 126; SWprint(TXval); SWprint(TXval2);

Definitions Needed:

1. define bit9600Delay 84
2. define halfBit9600Delay 42
3. define bit4800Delay 188
4. define halfBit4800Delay 94

These definitions set the delays necessary for 9600 baud and 4800 baud software serial operation.

Restore
August 23, 2006, at 02:09 PM by Heather Dewey-Hagborg -
Changed line 113 from:

//Created July 2006

to:

//Created August 15 2006

Restore
August 23, 2006, at 02:08 PM by Heather Dewey-Hagborg -
Added lines 113-116:

//Created July 2006 //Heather Dewey-Hagborg //http://www.arduino.cc

Restore
August 15, 2006, at 08:16 PM by Tom Igoe -
Changed lines 38-39 from:

First we include the file ctype.h in our application. This gives us access to the `toupper()` function from the Character Operations C library which we will use later in our main loop. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates. The `#define bit9600Delay 84` line causes the compiler to substitute the number 84 where ever it encounters the label "bit9600Delay". Pre-processor definitions are often used for constants because they don't take up any program memory space on the chip.

to:

First we include the file ctype.h in our application. This gives us access to the `toupper()` function from the Character Operations C library which we will use later in our main loop. This library is part of the Arduino install, so you don't need to do anything other than type the #include line in order to use it. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates. The `#define bit9600Delay 84` line causes the compiler to substitute the number 84 where ever it encounters the label "bit9600Delay". Pre-processor definitions are often used for constants because they don't take up any program memory space on the chip.

Changed line 86 from:

```
    delayMicroseconds(halfbit9600Delay);
```

to:

```
    delayMicroseconds(halfBit9600Delay);
```

Changed line 160 from:

```
    delayMicroseconds(halfbit9600Delay);
```

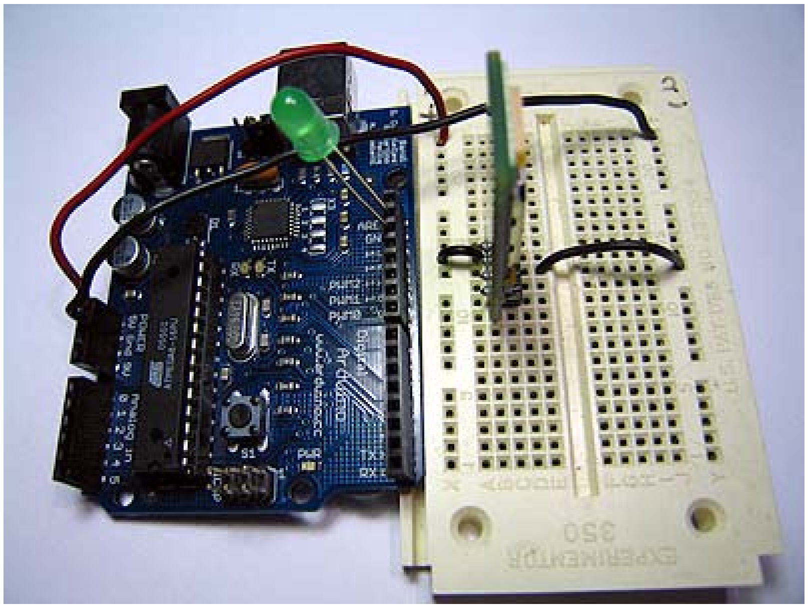
to:

```
    delayMicroseconds(halfBit9600Delay);
```

Restore
August 15, 2006, at 02:32 PM by Heather Dewey-Hagborg -
Changed lines 19-20 from:

Attach: pwr_wires_web.jpg

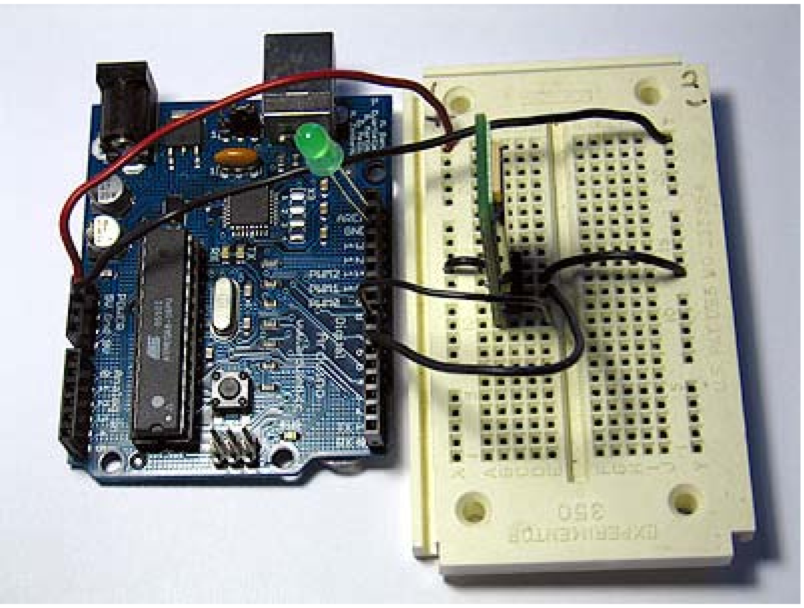
to:



Changed lines 23-24 from:

Attach: ser_wires_web.jpg

to:



Restore
August 15, 2006, at 02:30 PM by Heather Dewey-Hagborg -
Changed lines 19-20 from:

### picture of device with connections

to:

Attach: pwr_wires_web.jpg

Changed lines 23-24 from:

### picture of device with serial connections

to:

Attach: ser_wires_web.jpg

Restore
August 15, 2006, at 11:44 AM by Heather Dewey-Hagborg -
Restore
August 15, 2006, at 11:42 AM by Heather Dewey-Hagborg -
Changed lines 78-79 from:

This is the SWprint function. First the transmit line is pulled low to signal a start bit. Then we itterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted. Finally we pull the line high again to signal a stop bit. For each bit we transmit we hold the line high or low for the specified delay. In this example we are using a 9600 baudrate. To use 4800 simply replace the variable `bit9600Delay` with `bit4800Delay`.

to:

This is the SWprint function. First the transmit line is pulled low to signal a start bit. Then we itterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted. Finally we pull the line high again to signal a stop bit. For each bit we transmit we hold the line high or low for the specified delay. In this example we are using a 9600 baudrate. To use 4800 simply replace the variable `bit9600Delay` with `bit4800Delay`.

Restore
August 15, 2006, at 11:40 AM by Heather Dewey-Hagborg -
Restore
August 15, 2006, at 11:37 AM by Heather Dewey-Hagborg -
Changed line 3 from:

In this tutorial you will learn how to implement serial

to:

In this tutorial you will learn how to implement Asynchronous serial

Changed lines 5-6 from:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page.

to:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page. For a good explanation of serial communication see Wikipedia.

Changed lines 27-28 from:

Now we will write the code to enable serial data transmission. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. We will walk through the code in small sections.

to:

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. We will walk through the code in small sections.

Restore
August 15, 2006, at 11:22 AM by Heather Dewey-Hagborg -

Changed lines 36-37 from:

Here we set up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons. First we include the file ctype.h in our application. This gives us access to the `toupper()` function from the Character Operations C library which we will use later in our main loop. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates. The `#define bit9600Delay 84` line causes the compiler to substitute the number 84 where ever it encounters the label "bit9600Delay". Pre-processor definitions are often used for constants because they don't take up any program memory space on the chip.

to:

Here we set up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

First we include the file ctype.h in our application. This gives us access to the `toupper()` function from the Character Operations C library which we will use later in our main loop. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates. The `#define bit9600Delay 84` line causes the compiler to substitute the number 84 where ever it encounters the label "bit9600Delay". Pre-processor definitions are often used for constants because they don't take up any program memory space on the chip.

Restore
August 15, 2006, at 11:21 AM by Heather Dewey-Hagborg -
Changed lines 36-37 from:

Here we import the file ctype.h to our application. This gives us access to the `toupper()` function from the Character Operations C library. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates.

to:

Here we set up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons. First we include the file ctype.h in our application. This gives us access to the `toupper()` function from the Character Operations C library which we will use later in our main loop. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates. The `#define bit9600Delay 84` line causes the compiler to substitute the number 84 where ever it encounters the label "bit9600Delay". Pre-processor definitions are often used for constants because they don't take up any program memory space on the chip.

Restore
August 15, 2006, at 10:57 AM by Heather Dewey-Hagborg -
Changed lines 104-105 from:

Finally we implement our main program loop. In this program we simply wait for characters to arrive, chnge them to uppercase and send them back. This is always a good program to run when you want to make sure a serial connection is working properly.

to:

Finally we implement our main program loop. In this program we simply wait for characters to arrive, change them to uppercase and send them back. This is always a good program to run when you want to make sure a serial connection is working properly.

Restore
August 13, 2006, at 11:19 AM by Heather Dewey-Hagborg -
Changed line 101 from:

```
    SWprint(to_upper(SWval));
```

to:

```
    SWprint(toupper(SWval));
```

Changed line 173 from:

```
    SWprint(to_upper(SWval));
```

to:

```
    SWprint(toupper(SWval));
```

Restore
August 13, 2006, at 11:12 AM by Heather Dewey-Hagborg -
Changed lines 36-37 from:

Here we import the file ctype.h to our application. This gives us access to the `toupper()` function from the standard C library. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates.

to:

Here we import the file ctype.h to our application. This gives us access to the `toupper()` function from the Character Operations C library. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates.

Restore
August 13, 2006, at 11:03 AM by Heather Dewey-Hagborg -
Changed lines 5-6 from:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page.

to:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page.

Added line 48:

```
  digitalWrite(13,HIGH); //turn on debugging LED
```

Changed lines 54-55 from:

Here we initialize the lines and print a debugging message to confirm all is working as planned. We can pass inidvidual characters or numbers to the SWprint function.

to:

Here we initialize the lines, turn on our debugging LED and print a debugging message to confirm all is working as planned. We can pass inidvidual characters or numbers to the SWprint function.

Added line 126:

```
  digitalWrite(13,HIGH); //turn on debugging LED
```

Restore
August 13, 2006, at 11:00 AM by Heather Dewey-Hagborg -
Changed lines 39-41 from:

byte tx = 7;@] byte SWval;

to:

byte tx = 7; byte SWval;@]

Added lines 102-172:

Finally we implement our main program loop. In this program we simply wait for characters to arrive, chnge them to uppercase and send them back. This is always a good program to run when you want to make sure a serial connection is working properly.

For lots of fun serial devices check out the Sparkfun online catalog. They have lots of easy to use serial modules for GPS, bluetooth, wi-fi, LCDs, etc.

For easy copy and pasting the full program text of this tutorial is below:

```
#include <ctype.h>

#define bit9600Delay 84
```

```
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94

byte rx = 6;
byte tx = 7;
byte SWval;

void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}

void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}

int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfbit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}

void loop()
{
    SWval = SWread();
    SWprint(to_upper(SWval));
}
```

Restore
August 13, 2006, at 10:55 AM by Heather Dewey-Hagborg -
Changed lines 29-31 from:

[@#define bit9600Delay 84

to:

[@#include <ctype.h>

1. define bit9600Delay 84

Changed lines 36-37 from:

Here we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates.

to:

Here we import the file ctype.h to our application. This gives us access to the `toupper()` function from the standard C library. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates.

Changed lines 40-42 from:

Here we set our transmit (tx) and recieve (rx) pins. Change the pin numbers to suit your application.

to:

byte SWval;

Here we set our transmit (tx) and recieve (rx) pins. Change the pin numbers to suit your application. We also allocate a variable to store our recieved data in, `SWval`.

Added lines 96-101:

```
void loop()
{
    SWval = SWread();
    SWprint(to_upper(SWval));
}
```

Restore
August 13, 2006, at 10:48 AM by Heather Dewey-Hagborg -
Changed lines 78-79 from:

```
  // confirm that this is a real start bit, not line noise
```

to:

```
  //wait for start bit
```

Deleted lines 79-80:

```
    // frame start indicated by a falling edge and low start bit
    // jump to the middle of the low start bit
```

Deleted lines 80-81:

```
    // offset of the bit in the byte: from 0 (LSB) to 7 (MSB)
```

Deleted line 81:

```
     // jump to middle of next bit
```

Deleted lines 82-83:

```
     // read bit
```

Changed line 85 from:

```
    //pause for stop bit
```

to:

```
    //wait for stop bit + extra
```

Added line 93:
Restore

August 13, 2006, at 10:47 AM by Heather Dewey-Hagborg -
Changed lines 5-6 from:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. And device specific tutorials are on the Tutorial Page.

to:

other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page.

Added lines 73-101:

```
int SWread()
{
  byte val = 0;
  while (digitalRead(rx));

  // confirm that this is a real start bit, not line noise
  if (digitalRead(rx) == LOW) {
    // frame start indicated by a falling edge and low start bit
    // jump to the middle of the low start bit
    delayMicroseconds(halfbit9600Delay);

    // offset of the bit in the byte: from 0 (LSB) to 7 (MSB)
    for (int offset = 0; offset < 8; offset++) {
     // jump to middle of next bit
     delayMicroseconds(bit9600Delay);

     // read bit
     val |= digitalRead(rx) << offset;
    }
    //pause for stop bit
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}
```

This is the SWread function. This will wait for a byte to arrive on the recieve pin and then return it to the allocated variable. First we wait for the recieve line to be pulled low. We check after a half bit delay to make sure the line is still low and we didn't just recieve line noise. Then we iterate through a bit mask and shift 1s or 0s into our output byte based on what we recieve. Finally we allow a pause for the stop bit and then return the value.

Restore
August 13, 2006, at 10:38 AM by Heather Dewey-Hagborg -
Changed lines 72-73 from:

This is the SWprint function. First the transmit line is pulled low to signal a start bit. Then we itterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted. Finally we pull the line high again to signal a stop bit. For each bit we transmit we hold the line high or low for the specified delay. In this example we are using a 9600 baudrate. To use 4800 simply replace the variable "bit9600Delay" with "bit4800Delay".

to:

This is the SWprint function. First the transmit line is pulled low to signal a start bit. Then we itterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted. Finally we pull the line high again to signal a stop bit. For each bit we transmit we hold the line high or low for the specified delay. In this example we are using a 9600 baudrate. To use 4800 simply replace the variable `bit9600Delay` with `bit4800Delay`.

Restore
August 13, 2006, at 10:38 AM by Heather Dewey-Hagborg -
Changed line 29 from:

[@#define bit9600Delay 84 //total 104us

to:

[@#define bit9600Delay 84

Changed line 31 from:

1. define bit4800Delay 188 //total 208us

to:

1. define bit4800Delay 188

Changed lines 34-35 from:
to:

Here we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates.

```
byte rx = 6;
byte tx = 7;
```

Here we set our transmit (tx) and recieve (rx) pins. Change the pin numbers to suit your application.

```
void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}
```

Here we initialize the lines and print a debugging message to confirm all is working as planned. We can pass inidvidual characters or numbers to the SWprint function.

```
void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}
```

This is the SWprint function. First the transmit line is pulled low to signal a start bit. Then we itterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted. Finally we pull the line high again to signal a stop bit. For each bit we transmit we hold the line high or low for the specified delay. In this example we are using a 9600 baudrate. To use 4800 simply replace the variable "bit9600Delay" with "bit4800Delay".

Restore
August 13, 2006, at 10:27 AM by Heather Dewey-Hagborg -
Added lines 22-36:

## picture of device with serial connections

### Program the Arduino

Now we will write the code to enable serial data transmission. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. We will walk through the code in small sections.

```
#define bit9600Delay 84  //total 104us
#define halfBit9600Delay 42
#define bit4800Delay 188 //total 208us
#define halfBit4800Delay 94
```

Restore
August 13, 2006, at 10:19 AM by Heather Dewey-Hagborg -
Changed lines 17-19 from:

Insert the device you want to communicate with in the breadboard. Connect ground on the breadboard to ground from the microcontroller. If your device uses 5v power connect 5v from the microcontoller to 5v on the breadboard. Otherwise connect power and ground from an alternate power source to the breadboard in the same fashion. Make any other connections necessary for your device.

## picture of device with connections

to:

Insert the device you want to communicate with in the breadboard. Connect ground on the breadboard to ground from the microcontroller. If your device uses 5v power connect 5v from the microcontoller to 5v on the breadboard. Otherwise connect power and ground from an alternate power source to the breadboard in the same fashion. Make any other connections necessary for your device. Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground.

## picture of device with connections

Decide which pins you want to use for transmitting and receiving. In this example we will use pin 7 for transmitting and pin 6 for receiving, but any of the digital pins should work.

Restore
August 13, 2006, at 10:05 AM by Heather Dewey-Hagborg -
Changed lines 17-19 from:

Insert the device you want to communicate with in the breadboard.

to:

Insert the device you want to communicate with in the breadboard. Connect ground on the breadboard to ground from the microcontroller. If your device uses 5v power connect 5v from the microcontoller to 5v on the breadboard. Otherwise connect power and ground from an alternate power source to the breadboard in the same fashion. Make any other connections necessary for your device.

## picture of device with connections

Restore
August 13, 2006, at 09:51 AM by Heather Dewey-Hagborg -
Added lines 1-17:

# Arduino Software Serial Interface

In this tutorial you will learn how to implement serial communication on the Arduino in software to communicate with other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. And device specific tutorials are on the Tutorial Page.

Materials needed:

- Device to communicate with
- Solderless breadboard
- Hookup wire
- Arduino Microcontroller Module
- Light emitting Diode (LED) - optional, for debugging

### Prepare the breadboard

Insert the device you want to communicate with in the breadboard.

Restore

# Arduino Software Serial Interface

**Note:** *If you just want to use a software serial interface, see the SoftwareSerial library included with Arduino 0007 and later. Read on if you'd like to know how that library works.*

In this tutorial you will learn how to implement Asynchronous serial communication on the Arduino in software to communicate with other serial devices. Using software serial allows you to create a serial connection on any of the digital i/o pins on the Arduino. This should be used when multiple serial connections are necessary. If only one serial connection is necessary the hardware serial port should be used. This is a general purpose software tutorial, NOT a specific device tutorial. A tutorial on communicating with a computer is here. Device specific tutorials are on the Tutorial Page. For a good explanation of serial communication see Wikipedia. The software serial connection can run at 4800 baud or 9600 baud reliably.

Functions Available:

**SWread();** Returns a byte long integer value from the software serial connection

Example:

```
byte RXval;
RXval = SWread();
```

**SWprint();** Sends a byte long integer value out the software serial connection

Example:

```
byte TXval = 'h';
byte TXval2 = 126;
SWprint(TXval);
SWprint(TXval2);
```

Definitions Needed:

```
#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94
```

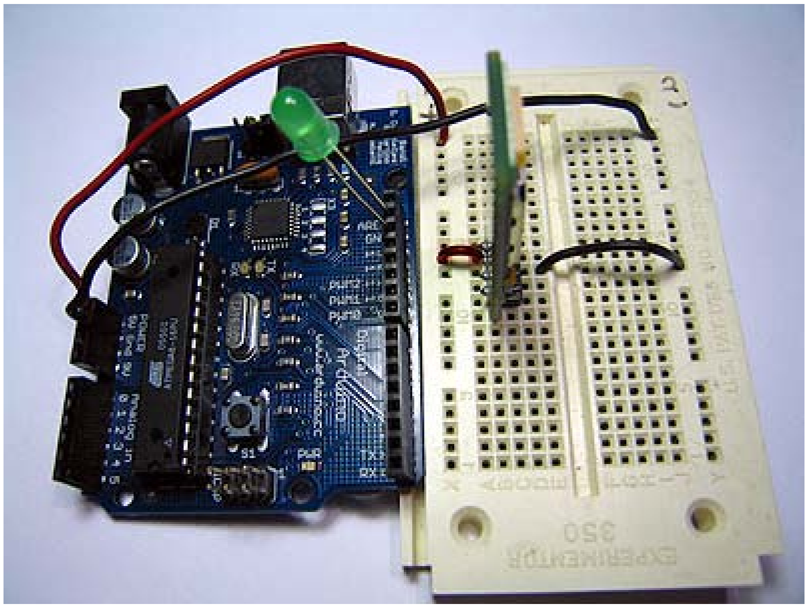
These definitions set the delays necessary for 9600 baud and 4800 baud software serial operation.

Materials needed:

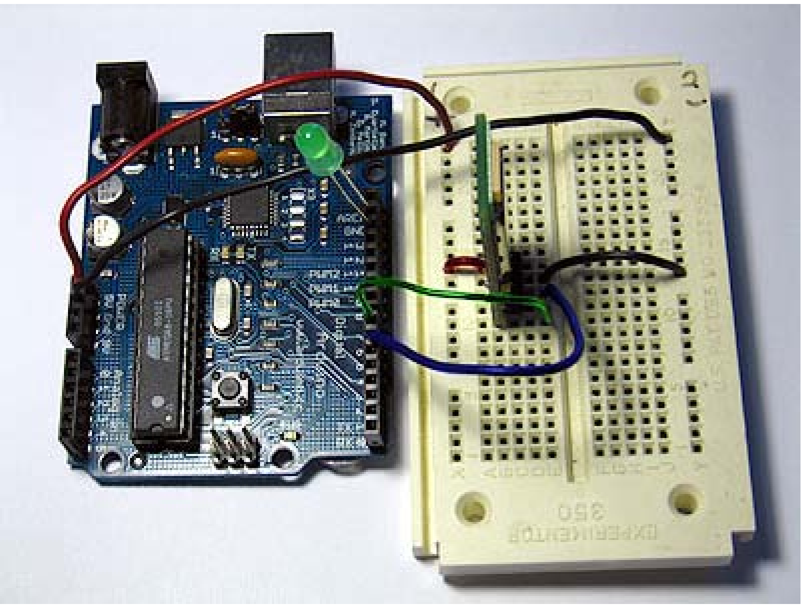
- Device to communicate with
- Solderless breadboard
- Hookup wire
- Arduino Microcontroller Module
- Light emitting Diode (LED) - optional, for debugging

## Prepare the breadboard

Insert the device you want to communicate with in the breadboard. Connect ground on the breadboard to ground from the microcontroller. If your device uses 5v power connect 5v from the microcontoller to 5v on the breadboard. Otherwise connect power and ground from an alternate power source to the breadboard in the same fashion. Make any other connections necessary for your device. Additionally you may want to connect an LED for debugging purposes between pin 13 and Ground.



Decide which pins you want to use for transmitting and receiving. In this example we will use pin 7 for transmitting and pin 6 for receiving, but any of the digital pins should work.



## Program the Arduino

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. We will walk through the code in small sections.

```
#include <ctype.h>

#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94
```

Here we set up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

First we include the file ctype.h in our application. This gives us access to the `toupper()` function from the Character Operations C library which we will use later in our main loop. This library is part of the Arduino install, so you don't need to do anything other than type the #include line in order to use it. Next we establish our baudrate delay definitions. These are pre-processor directives that define the delays for different baudrates. The `#define bit9600Delay 84` line causes the compiler to substitute the number 84 where ever it encounters the label "bit9600Delay". Pre-processor definitions are often used for constants because they don't take up any program memory space on the chip.

```
byte rx = 6;
byte tx = 7;
byte SWval;
```

Here we set our transmit (tx) and recieve (rx) pins. Change the pin numbers to suit your application. We also allocate a variable to store our recieved data in, `SWval`.

```
void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  digitalWrite(13,HIGH); //turn on debugging LED
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}
```

Here we initialize the lines, turn on our debugging LED and print a debugging message to confirm all is working as planned. We can pass inidvidual characters or numbers to the SWprint function.

```
void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}
```

This is the SWprint function. First the transmit line is pulled low to signal a start bit. Then we itterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted. Finally we pull the line high again to signal a stop bit. For each bit we transmit we hold the line high or low for the specified delay. In this example we are using a 9600 baudrate. To use 4800 simply replace the variable `bit9600Delay` with `bit4800Delay`.

```
int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfBit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}
```

This is the SWread function. This will wait for a byte to arrive on the recieve pin and then return it to the allocated variable. First we wait for the recieve line to be pulled low. We check after a half bit delay to make sure the line is still low and we didn't just recieve line noise. Then we iterate through a bit mask and shift 1s or 0s into our output byte based on what we recieve. Finally we allow a pause for the stop bit and then return the value.

```
void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```

Finally we implement our main program loop. In this program we simply wait for characters to arrive, change them to uppercase and send them back. This is always a good program to run when you want to make sure a serial connection is working properly.

For lots of fun serial devices check out the Sparkfun online catalog. They have lots of easy to use serial modules for GPS, bluetooth, wi-fi, LCDs, etc.

For easy copy and pasting the full program text of this tutorial is below:

```
//Created August 15 2006
//Heather Dewey-Hagborg
//http://www.arduino.cc

#include <ctype.h>

#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94

byte rx = 6;
byte tx = 7;
byte SWval;

void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  digitalWrite(13,HIGH); //turn on debugging LED
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}

void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}

int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfBit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}

void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```

*code and tutorial by Heather Dewey-Hagborg*

# Tutorial.ArduinoSoftwareRS232 History

Hide minor edits - Show changes to markup

September 05, 2006, at 12:56 PM by Heather Dewey-Hagborg -
Changed lines 144-145 from:

*code and tutorial by Heather Dewey-Hagborg Photos by Thomas Dexter*

to:

*code and tutorial by Heather Dewey-Hagborg, photos by Thomas Dexter*

Restore
September 05, 2006, at 12:56 PM by Heather Dewey-Hagborg -
Added line 144:

*code and tutorial by Heather Dewey-Hagborg*

Restore
August 29, 2006, at 12:07 PM by Heather Dewey-Hagborg -
Changed lines 3-4 from:

In this tutorial you will learn how to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino. A general purpose software serial tutorial can be found http://www.arduino.cc/en/Tutorial/SoftwareSerial?.

to:

In this tutorial you will learn how to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino. A general purpose software serial tutorial can be found here.

Restore
August 29, 2006, at 12:06 PM by Heather Dewey-Hagborg -
Changed lines 3-4 from:

In this tutorial you will learn how to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino.

to:

In this tutorial you will learn how to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino. A general purpose software serial tutorial can be found http://www.arduino.cc/en/Tutorial/SoftwareSerial?.

Restore
August 29, 2006, at 12:03 PM by Heather Dewey-Hagborg -
Changed lines 57-58 from:
to:

*TX wires Green, RX wires Blue, +5v wires are red, GND wires are black*

Restore
August 29, 2006, at 12:01 PM by Heather Dewey-Hagborg -
Changed lines 23-24 from:

"+5v wires are red, GND wires are black"

to:

*+5v wires are red, GND wires are black*

Changed lines 28-29 from:
to:

*+5v wires are red, GND wires are black*

Changed lines 33-34 from:
to:

*TX wire Green, RX wire Blue, +5v wires are red, GND wires are black*

Restore
August 29, 2006, at 11:59 AM by Heather Dewey-Hagborg -
Changed lines 23-24 from:
to:

"+5v wires are red, GND wires are black"

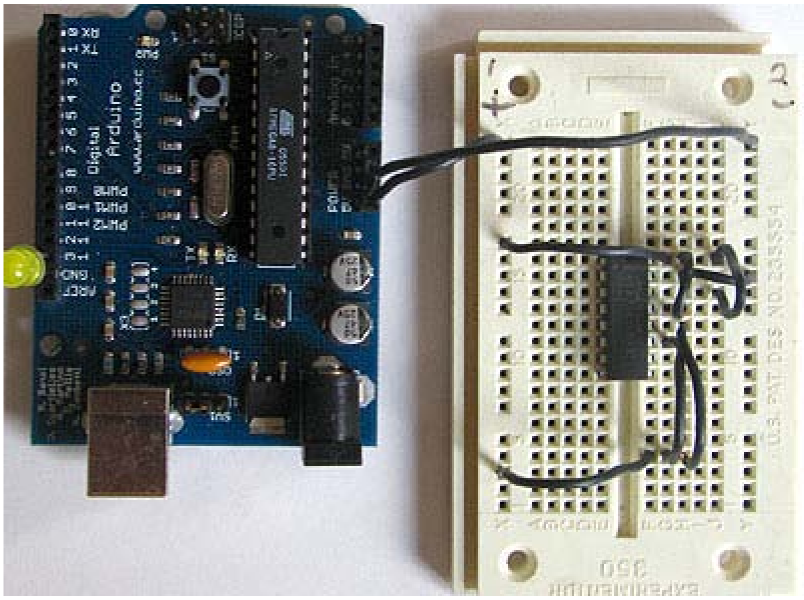
Changed lines 59-60 from:

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. Upload the follwoing code into the Arduino microcontroller module:

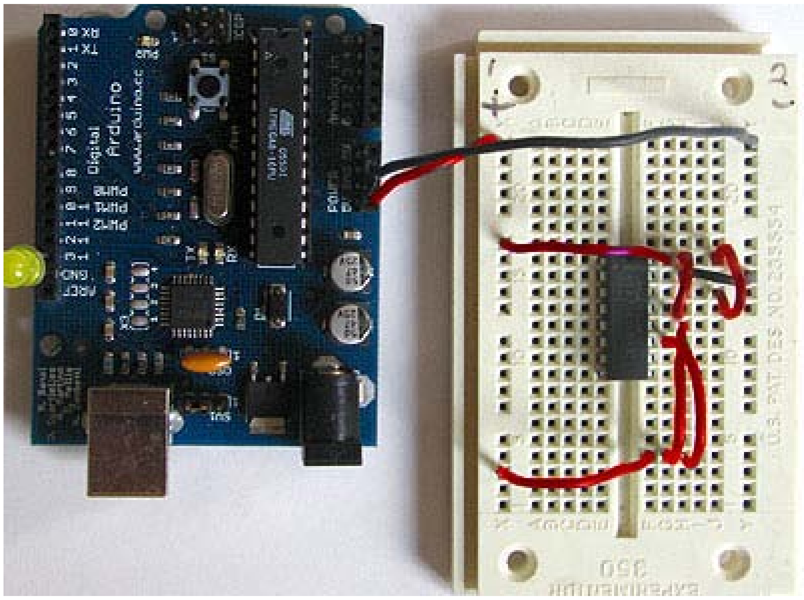
to:

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. Upload the following code into the Arduino microcontroller module:

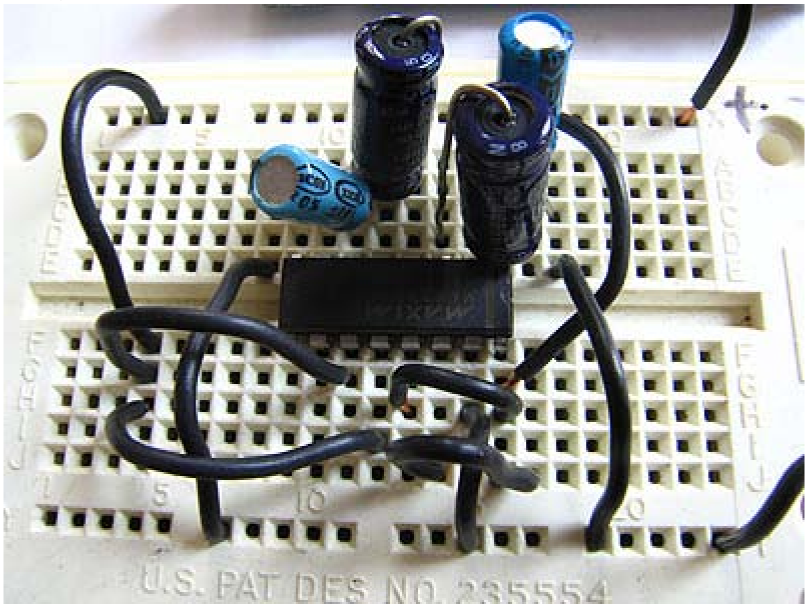
Restore
August 29, 2006, at 11:55 AM by Heather Dewey-Hagborg -
Changed lines 22-23 from:



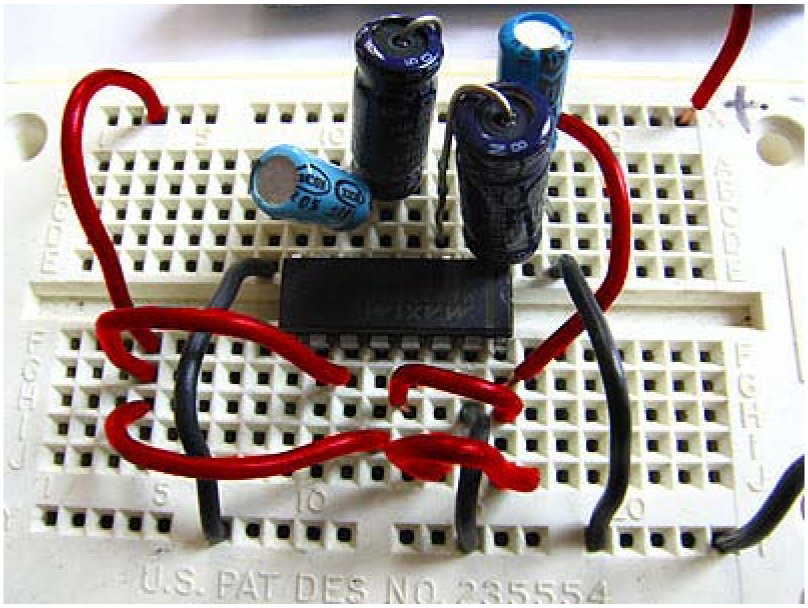
to:



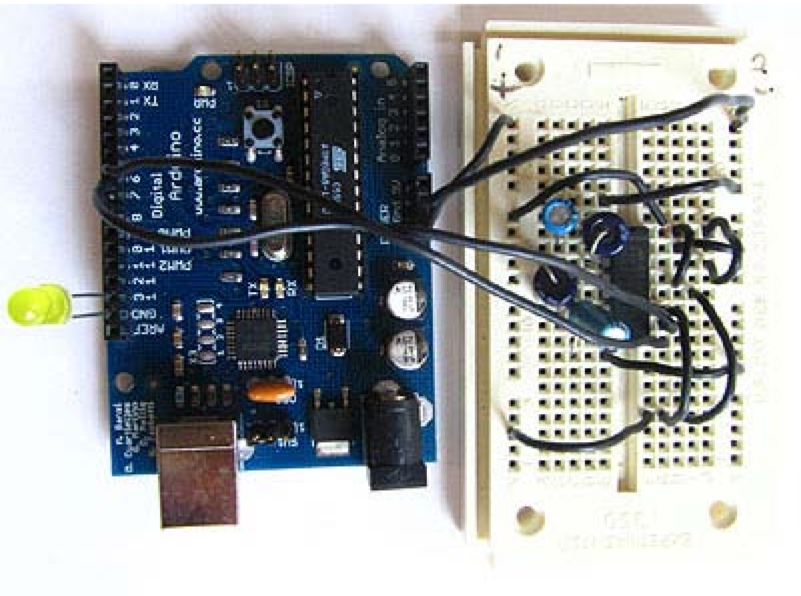
Changed lines 26-27 from:



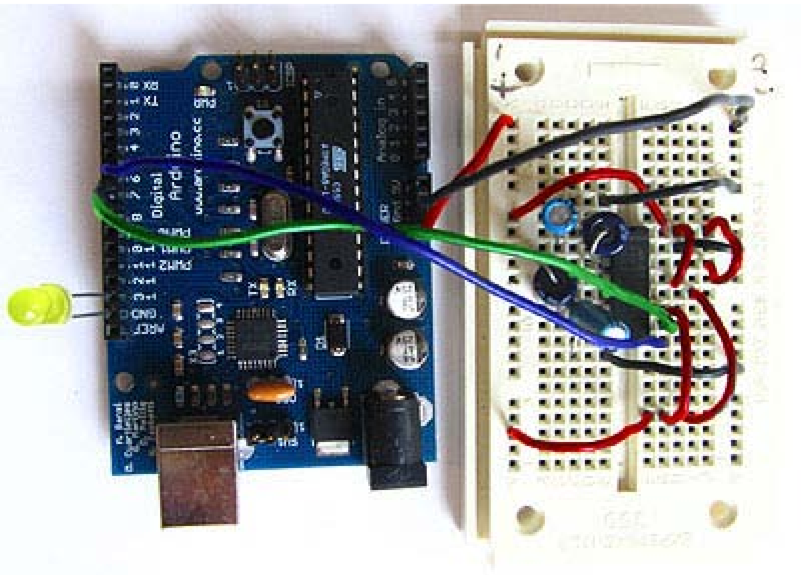
to:



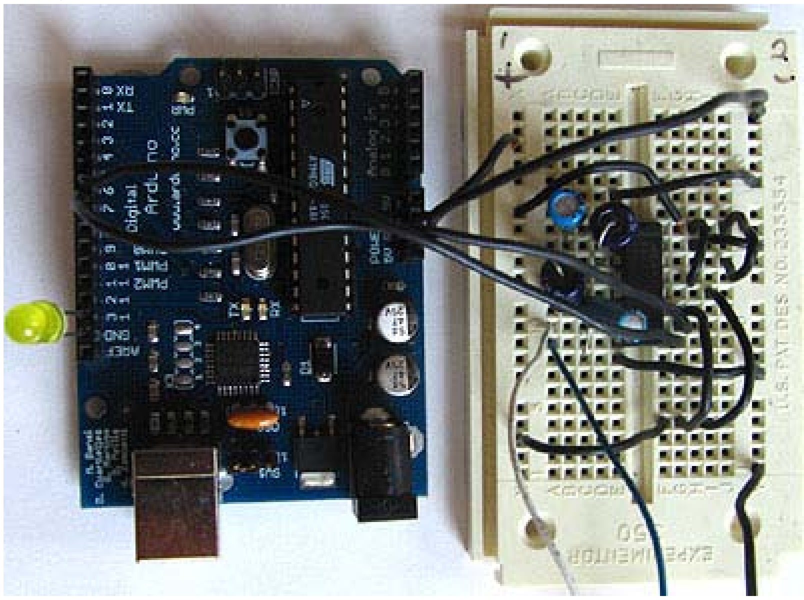
Changed lines 30-31 from:



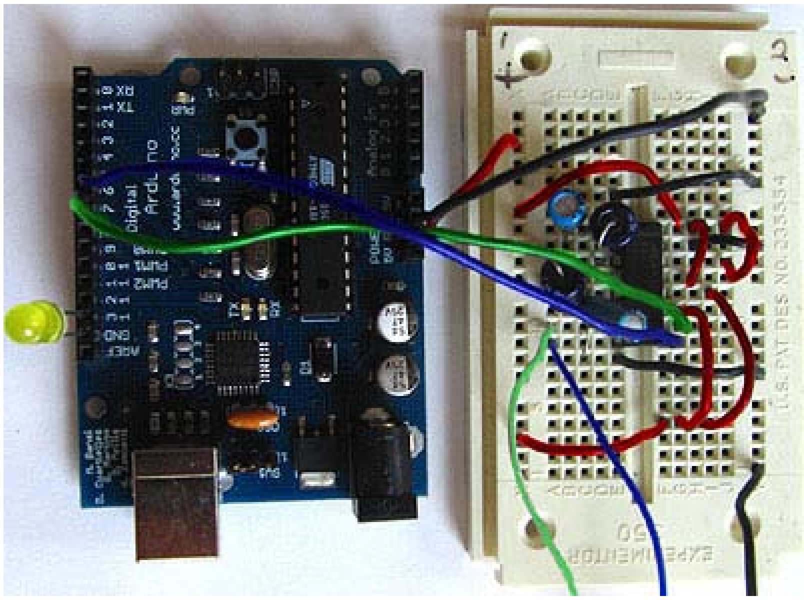
to:



Changed lines 53-55 from:



to:



Restore
August 23, 2006, at 02:09 PM by Heather Dewey-Hagborg -
Added lines 61-64:

//Created August 23 2006 //Heather Dewey-Hagborg //http://www.arduino.cc

Restore
August 23, 2006, at 02:05 PM by Heather Dewey-Hagborg -
Changed lines 135-137 from:

If this works, congratulations! Your serial connection is working as planned. You can now use your new serial/computer connection to print debugging statements from your code, and to send commands to your microcontroller.

to:

If this works, congratulations! Your serial connection is working as planned. You can now use your new serial/computer connection to print debugging statements from your code, and to send commands to your microcontroller.

*Photos by Thomas Dexter*

Restore
August 23, 2006, at 02:03 PM by Heather Dewey-Hagborg -
Changed lines 35-36 from:

*DB9 Serial Connector Pin Diagram*

to:

*(DB9 Serial Connector Pin Diagram)*

Restore
August 23, 2006, at 02:02 PM by Heather Dewey-Hagborg -
Changed lines 32-33 from:

### Cables

to:

## Cables

Changed lines 35-36 from:

("DB9 Serial Connector Pins")

to:

*DB9 Serial Connector Pin Diagram*

Changed lines 55-56 from:

**Program the Arduino**

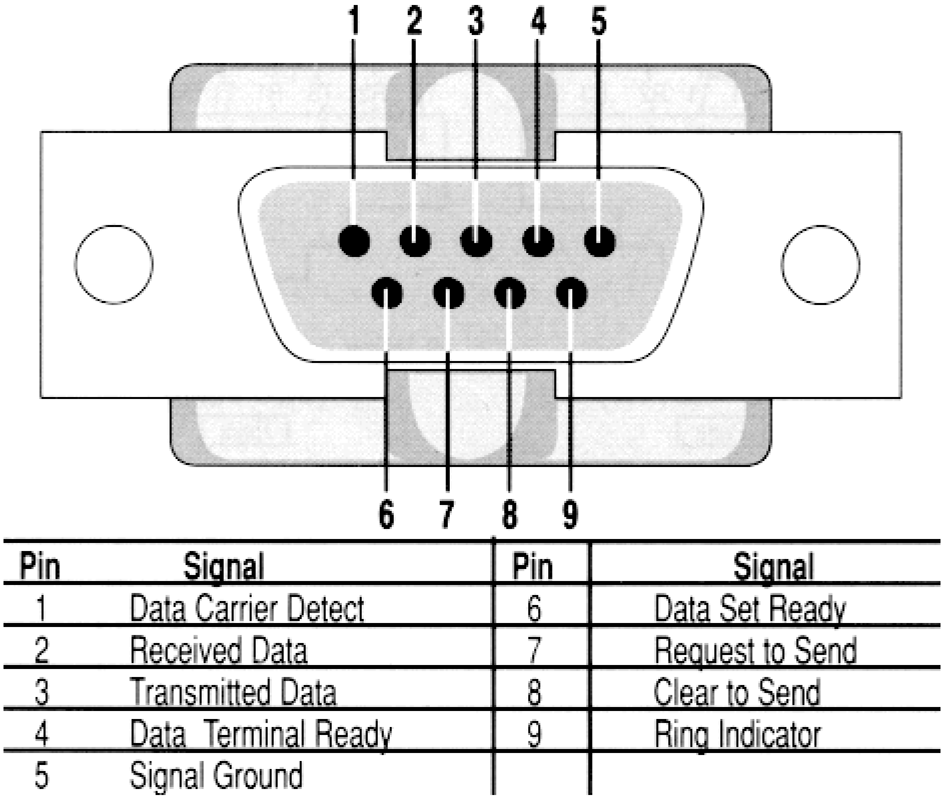
to:

## Program the Arduino

Restore

August 23, 2006, at 02:00 PM by Heather Dewey-Hagborg -

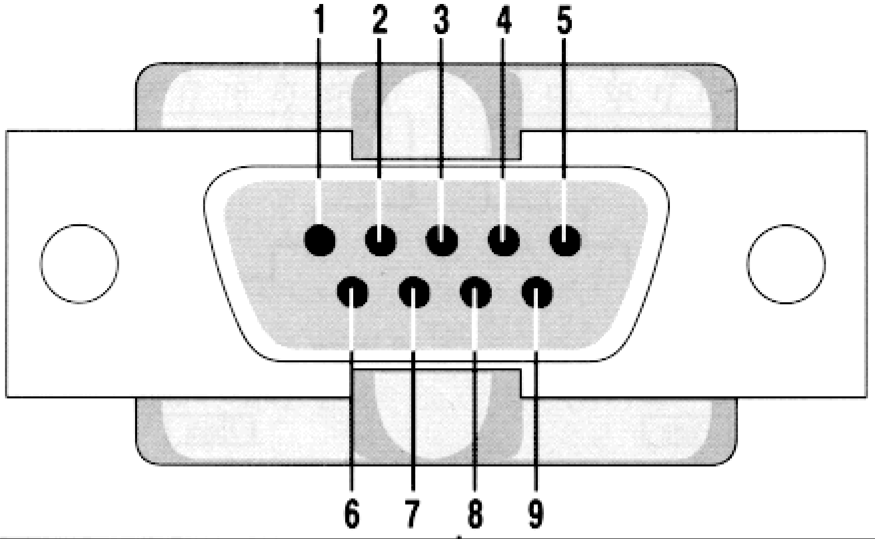
Added lines 34-36:

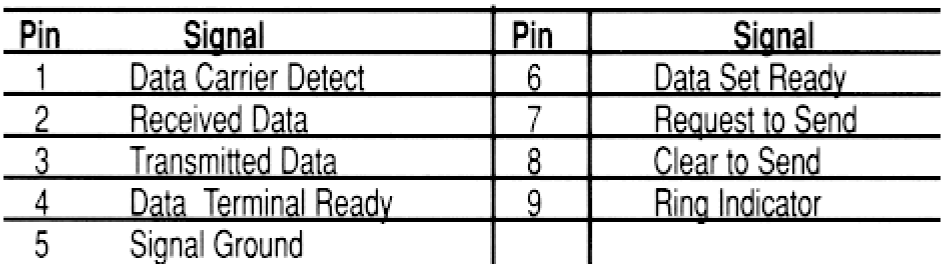


| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data  Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

("DB9 Serial Connector Pins")

Deleted lines 48-51:





| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data  Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

Restore

August 23, 2006, at 01:58 PM by Heather Dewey-Hagborg -

Added lines 43-44:

Added line 48:

Added line 55:

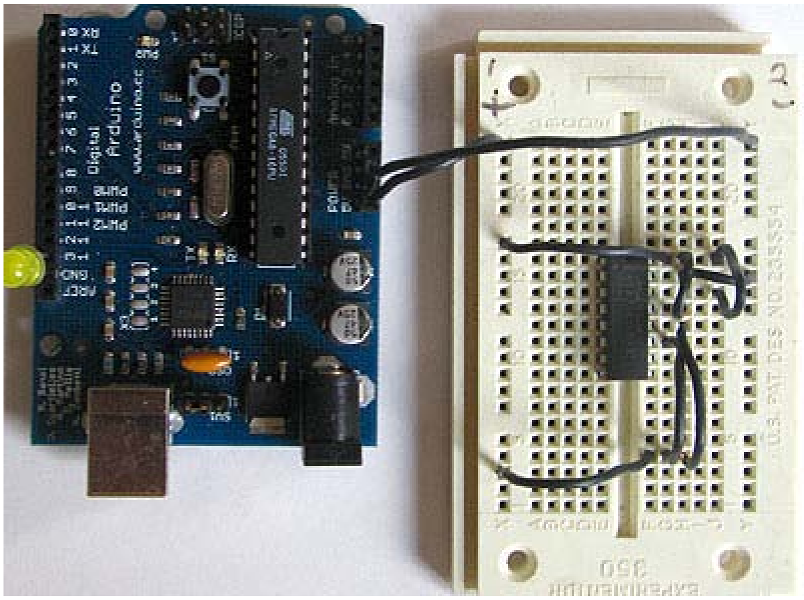
Restore

August 23, 2006, at 01:55 PM by Heather Dewey-Hagborg -

Changed lines 22-23 from:

PICTURE

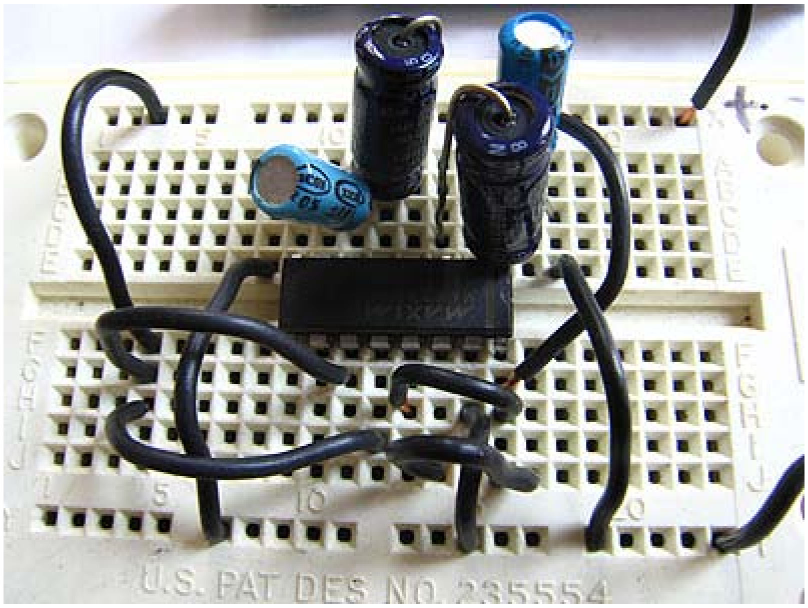
to:



Changed lines 26-27 from:

PICTURE

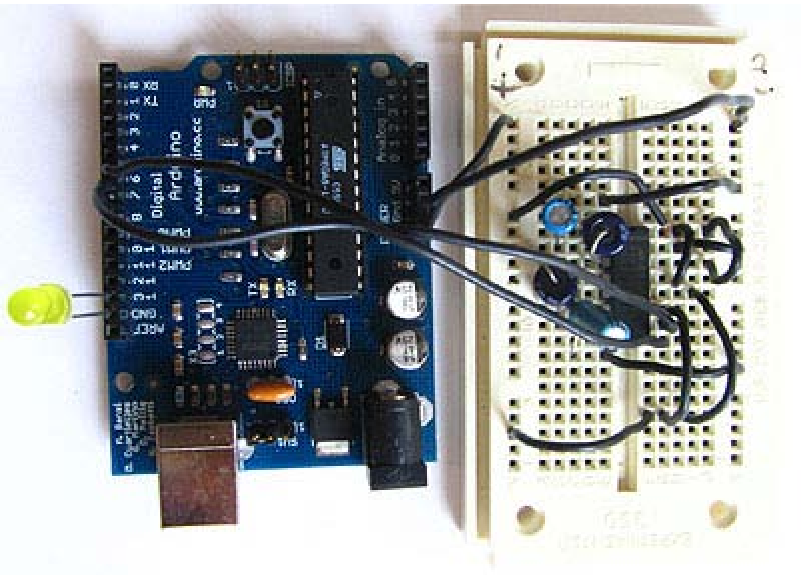
to:



Changed lines 30-31 from:

PICTURE

to:



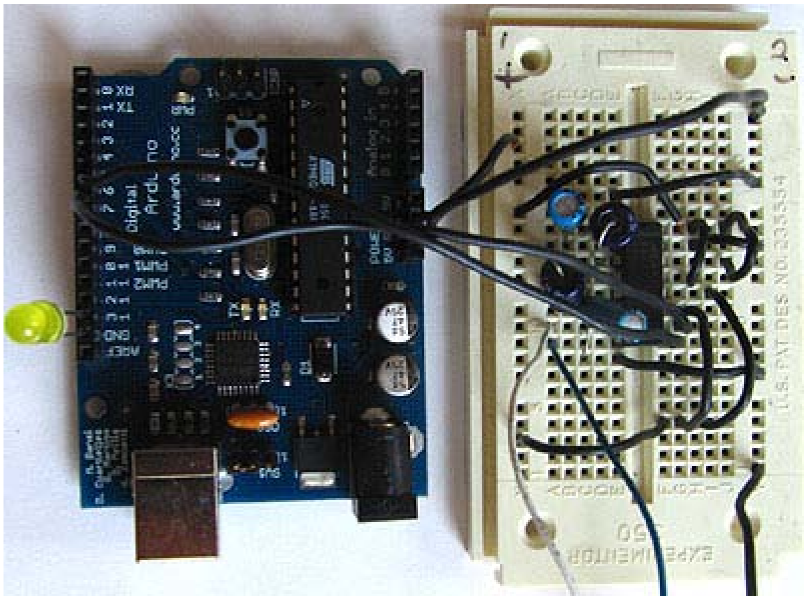
Changed lines 48-52 from:

Connect the TX line from your computer to pin 8 (R1IN) on the MAX233 and the RX line to pin 7 (T1OUT).

PICTURE

to:

Connect the TX line from your computer to pin 8 (R1IN) on the MAX233 and the RX line to pin 7 (T1OUT). Connect the ground line from your computer to ground on the breadboard.



Restore
August 23, 2006, at 01:11 PM by Heather Dewey-Hagborg -
Changed lines 20-21 from:

Insert the MAX3323 chip in the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect pin 15 on the MAX233 chip to ground and pins 16 and 14 - 11 to 5V. Connect a 1uF capacitor across pins 1 and 3, another across pins 4 and 5, another between pin 1 and ground, and the last between pin 6 and ground. If you are using polarized capacitors make sure the negative pins connect to the negative sides (pins 3 and 5 and ground).

to:

Insert the MAX3323 chip in the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect pin 15 on the MAX233 chip to ground and pins 16 and 14 - 11 to 5V. If you are using an LED connect it between pin 13 and ground.

Added lines 24-27:

Connect a 1uF capacitor across pins 1 and 3, another across pins 4 and 5, another between pin 1 and ground, and the last between pin 6 and ground. If you are using polarized capacitors make sure the negative pins connect to the negative sides (pins 3 and 5 and ground).

PICTURE

Restore
August 23, 2006, at 01:01 PM by Heather Dewey-Hagborg -
Changed lines 3-4 from:

In this tutorial you will learn how to communicate with a computer using a MAX233 multichannel RS-232 driver/receiver and a software serial connection on the Arduino.

to:

In this tutorial you will learn how to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino.

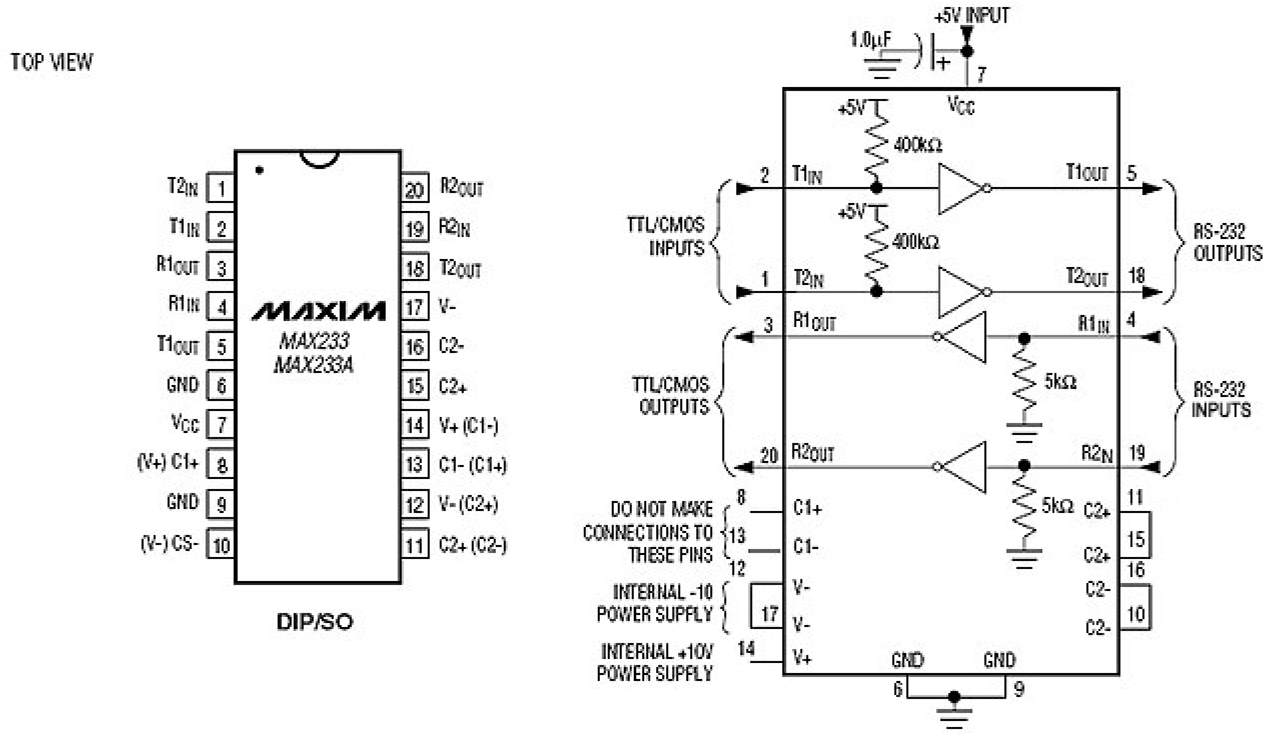
Changed lines 9-10 from:

- MAX233 chip (or similar)
- 1uf polarized capacitor

to:

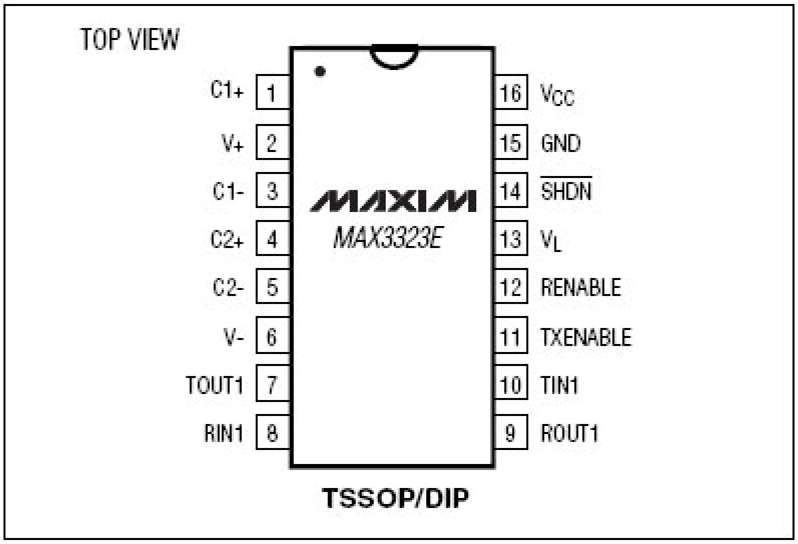
- MAX3323 chip (or similar)
- 4 1uf capacitors

Changed lines 18-21 from:



Insert the MAX233 chip in the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect pin 6 and pin 9 on the MAX233 chip to ground and pin 7 to 5V. Connect the 1uF capacitor across pins 6 and 7 so that the negative pin connects to pin 6 and the positive pin to pin 7. Connect pin 10 to pin 16 pin 11 to pin 15 and pin 12 to pin 17 on the breadboard.

to:



Insert the MAX3323 chip in the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect pin 15 on the MAX233 chip to ground and pins 16 and 14 - 11 to 5V. Connect a 1uF capacitor across pins 1 and 3, another across pins 4 and 5, another between pin 1 and ground, and the last between pin 6 and ground. If you are using polarized capacitors make sure the negative pins connect to the negative sides (pins 3 and 5 and ground).

Changed lines 24-27 from:

The MAX233 chip has two sets of RS-232 line shifters built in and can handle two simultaneous duplex serial ports. For the purposes of this tutorial we will only being using one port, with corresponding pins referred to as T1IN, T1OUT, R1IN and R1OUT in the MAX233 schematic.

Determine which Arduino pins you want to use for your transmit (TX) and recieve (RX) lines. In this tutorial we will be using Arduino pin 6 for receiving and pin 7 for transmitting. Connect your TX pin (7) to MAX233 pin 2 (T1IN). Connect your RX pin

(6) to MAX233 pin 3 (R1OUT).

to:

Determine which Arduino pins you want to use for your transmit (TX) and recieve (RX) lines. In this tutorial we will be using Arduino pin 6 for receiving and pin 7 for transmitting. Connect your TX pin (7) to MAX3323 pin 10 (T1IN). Connect your RX pin (6) to MAX3323 pin 9 (R1OUT).

Changed lines 44-45 from:

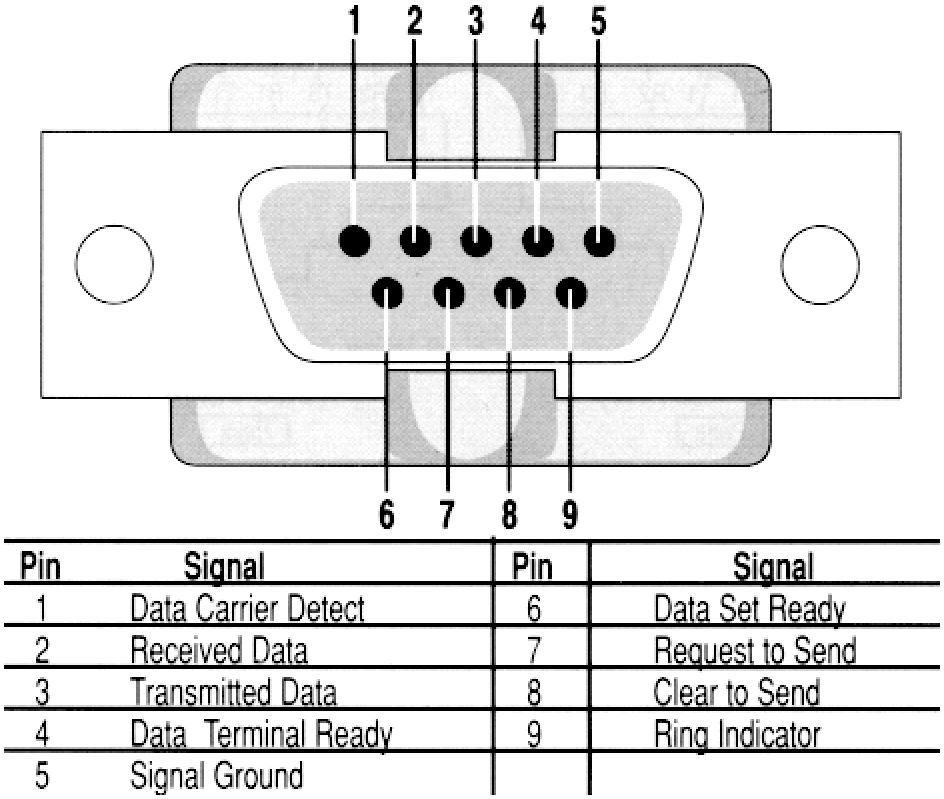
Connect the TX line from your computer to pin 4 (R1IN) on the MAX233 and the RX line to pin 5 (T1OUT).

to:

Connect the TX line from your computer to pin 8 (R1IN) on the MAX233 and the RX line to pin 7 (T1OUT).

Restore
August 17, 2006, at 01:06 PM by Heather Dewey-Hagborg -
Restore
August 17, 2006, at 01:05 PM by Heather Dewey-Hagborg -
Changed lines 32-34 from:

If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. To do this, pick up a female DB9 connector from radioshack. Pick three different colors of wire, one for TX, one for RX, and one for ground. Solder your TX wire to pin 2 of the DB9 connector, RX wire to pin 3 and Ground to pin 5. Connect pins 1 and 6 to pin 4 and pin 7 to pin 8. Heatshrink the wire connections to avoid accidental shorts. Enclose the connector in a backshell to further protect the signal and enable easy unplugging from your serial port.



| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

to:

If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. To do this, pick up a female DB9 connector from radioshack. Pick three different colors of wire, one for TX, one for RX, and one for ground. Solder your TX wire to pin 2 of the DB9 connector, RX wire to pin 3 and Ground to pin 5.

Added lines 35-37:

Connect pins 1 and 6 to pin 4 and pin 7 to pin 8. Heatshrink the wire connections to avoid accidental shorts.

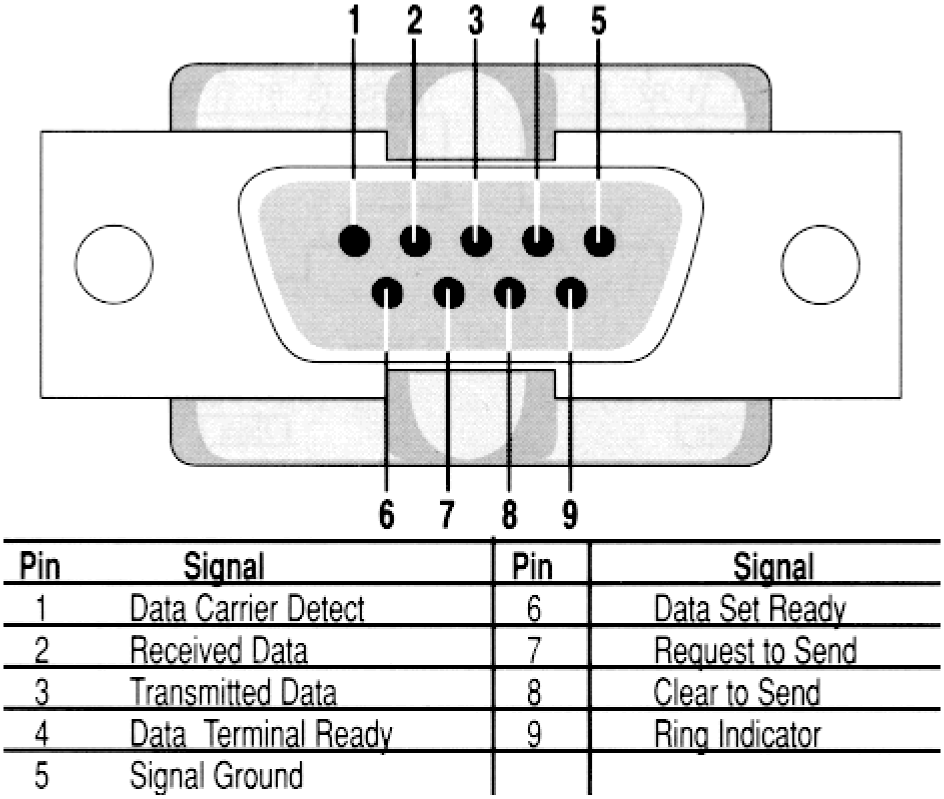
Added lines 39-40:

Enclose the connector in a backshell to further protect the signal and enable easy unplugging from your serial port.

Changed lines 42-44 from:

PICTURE in back shell

to:



| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

Restore

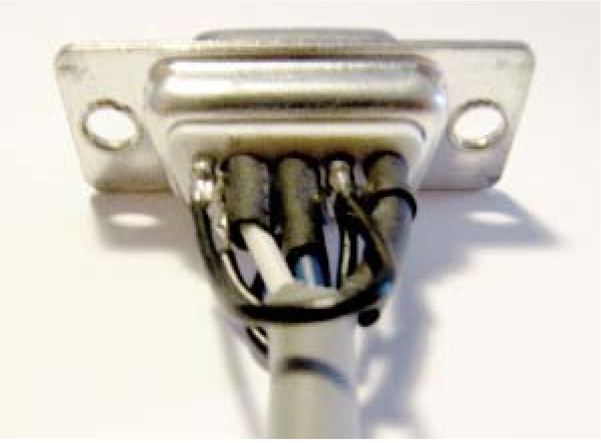
August 17, 2006, at 01:02 PM by Heather Dewey-Hagborg -
Added lines 30-31:

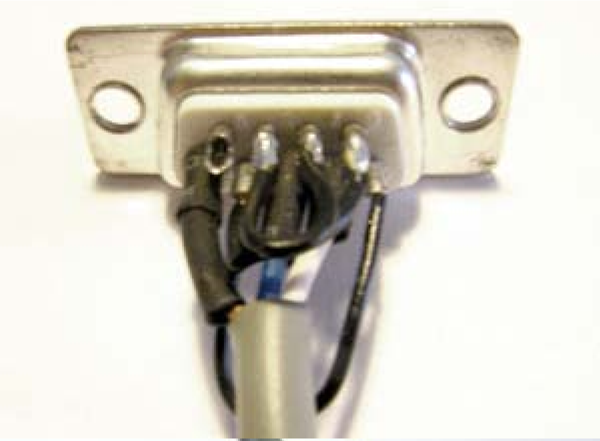
### Cables

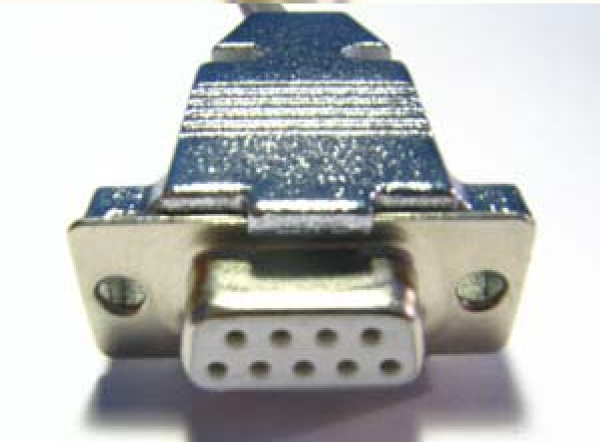
Changed lines 35-36 from:

PICTURE connector soldered,

to:







Restore
August 17, 2006, at 01:01 PM by Heather Dewey-Hagborg -
Changed lines 34-35 from:

PICTURE connector soldered, in back shell

to:

PICTURE connector soldered, PICTURE in back shell

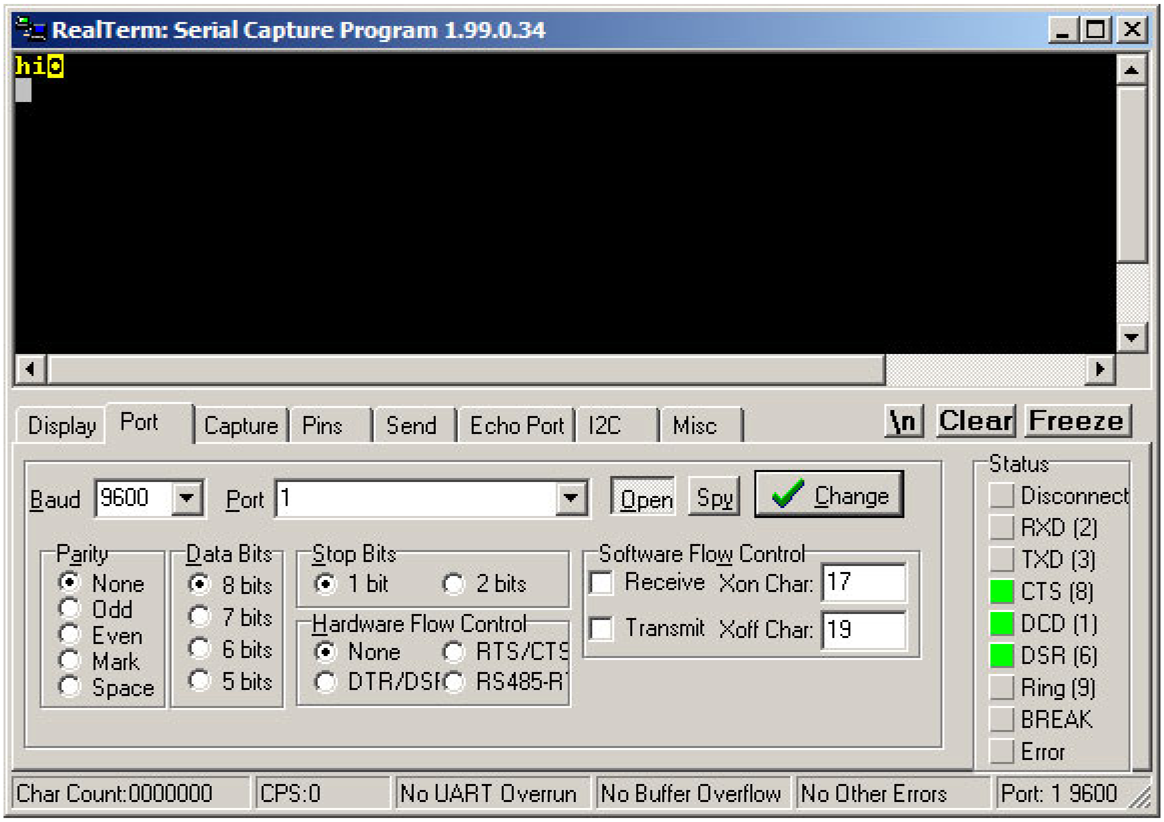
Restore
August 17, 2006, at 10:45 AM by Heather Dewey-Hagborg -
Changed lines 112-113 from:

Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by a line feed character and/or an advancement to the next line. Here are two shots of what it might look like, one in Hyperterminal the free pre-installed windows terminal application, and one in Realterm, another free application with more advanced options.

to:

Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by an advancement to the next line. Here is a shot of what it should look like in Hyperterminal, the free pre-installed windows terminal application.

Changed lines 115-116 from:



RealTerm: Serial Capture Program 1.99.0.34

```
hi
```

Baud 9600 Port 1

Parity: ● None ○ Odd ○ Even ○ Mark ○ Space

Data Bits: ● 8 bits ○ 7 bits ○ 6 bits ○ 5 bits

Stop Bits: ● 1 bit ○ 2 bits

Hardware Flow Control: ● None ○ RTS/CTS ○ DTR/DSR ○ RS485-RT

Software Flow Control: □ Receive Xon Char: 17 □ Transmit Xoff Char: 19

Status: □ Disconnect □ RXD (2) □ TXD (3) 🟩 CTS (8) 🟩 DCD (1) 🟩 DSR (6) □ Ring (9) □ BREAK □ Error

Char Count:0000000 | CPS:0 | No UART Overrun | No Buffer Overflow | No Other Errors | Port: 1 9600

to:

Restore

August 17, 2006, at 10:43 AM by Heather Dewey-Hagborg -

Changed lines 112-113 from:

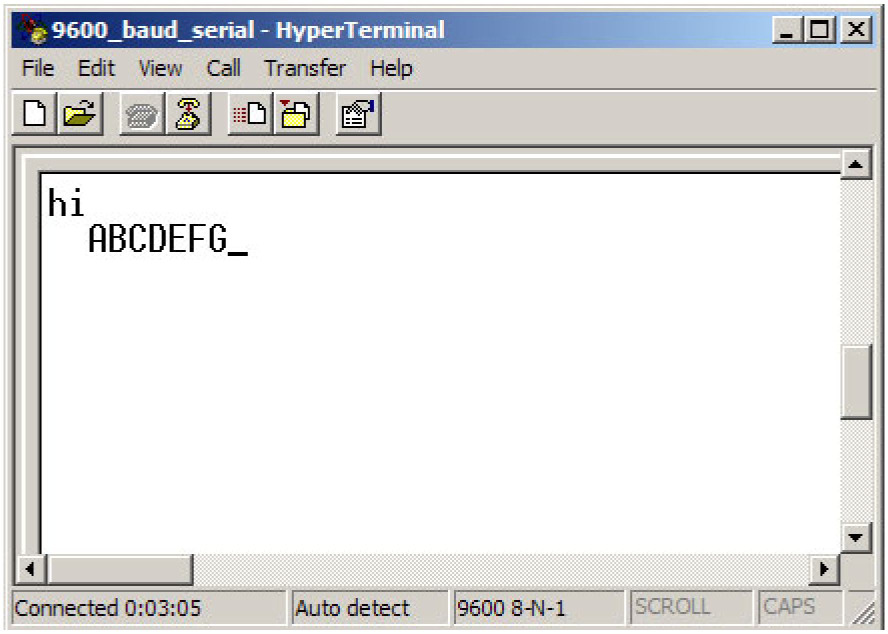
Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by a line feed character and/or an advancement to the next line. Here are two shots of what it might look like, one in Hyperterminal the free pre-installed windows terminal application, and one in Realterm, another free application with more options.

to:

Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by a line feed character and/or an advancement to the next line. Here are two shots of what it might look like, one in Hyperterminal the free pre-installed windows terminal application, and one in Realterm, another free application with more advanced options.

Added lines 117-121:

Now, try typing a lowercase character into the terminal window. You should see the letter you typed return to you in uppercase.


9600_baud_serial - HyperTerminal
File Edit View Call Transfer Help

```
hi
  ABCDEFG_
```

Connected 0:03:05
Auto detect
9600 8-N-1
SCROLL
CAPS

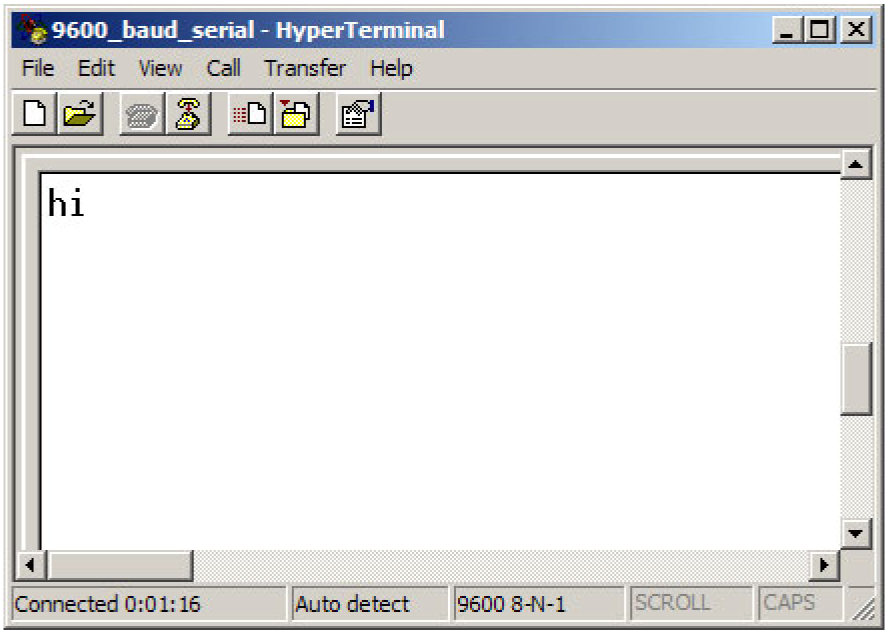
If this works, congratulations! Your serial connection is working as planned. You can now use your new serial/computer connection to print debugging statements from your code, and to send commands to your microcontroller.

Restore
August 17, 2006, at 10:26 AM by Heather Dewey-Hagborg -
Changed lines 112-116 from:

Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by a line feed character and an advancement to the next line.

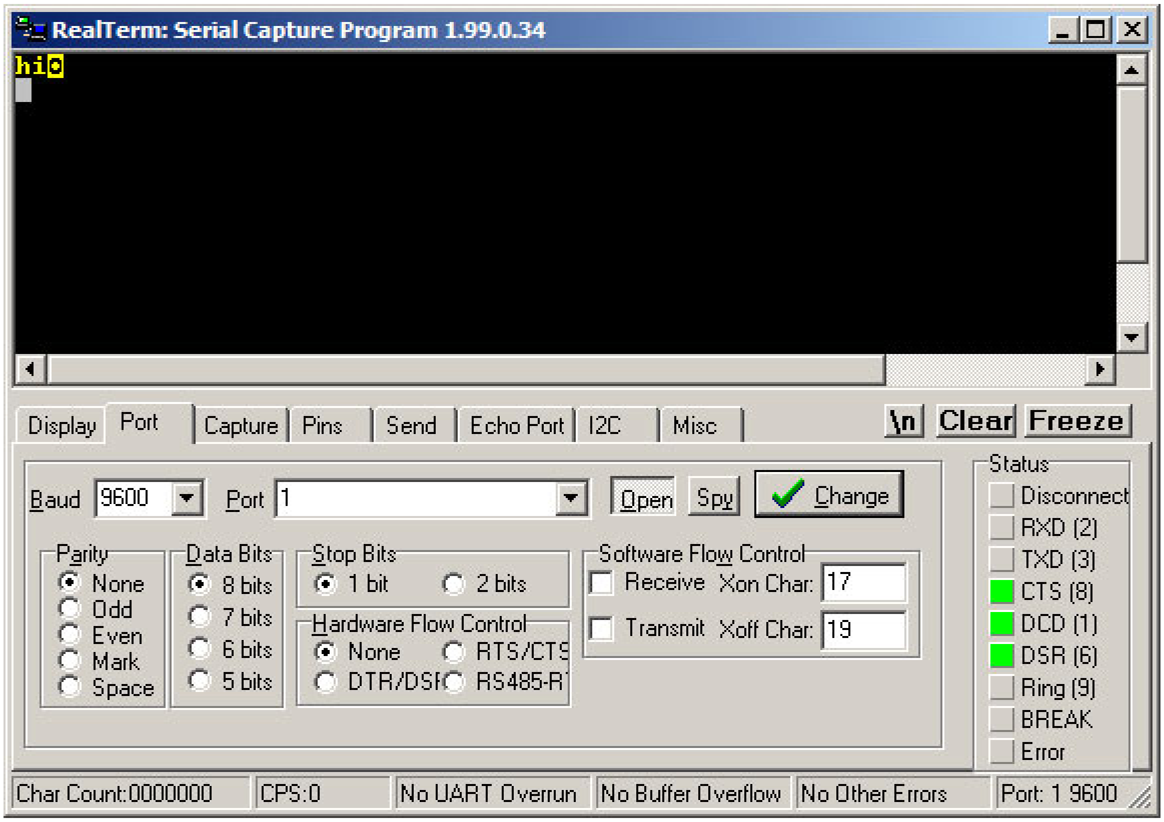
to:

Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by a line feed character and/or an advancement to the next line. Here are two shots of what it might look like, one in Hyperterminal the free pre-installed windows terminal application, and one in Realterm, another free application with more options.


9600_baud_serial - HyperTerminal
File Edit View Call Transfer Help

```
hi
```

Connected 0:01:16
Auto detect
9600 8-N-1
SCROLL
CAPS



```
hi
```

Baud 9600 Port 1

Parity: ● None ○ Odd ○ Even ○ Mark ○ Space

Data Bits: ● 8 bits ○ 7 bits ○ 6 bits ○ 5 bits

Stop Bits: ● 1 bit ○ 2 bits

Hardware Flow Control: ● None ○ RTS/CTS ○ DTR/DSR ○ RS485-RT

Software Flow Control: □ Receive Xon Char: 17 □ Transmit Xoff Char: 19

Char Count:0000000 | CPS:0 | No UART Overrun | No Buffer Overflow | No Other Errors | Port: 1 9600

Restore
August 17, 2006, at 10:19 AM by Heather Dewey-Hagborg -
Changed line 34 from:

PICTURE connector soldered

to:

PICTURE connector soldered, in back shell

Changed lines 110-112 from:

@]

to:

@]

Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by a line feed character and an advancement to the next line.

Restore
August 17, 2006, at 10:02 AM by Heather Dewey-Hagborg -
Changed lines 30-31 from:

If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. To do this, pick up a female DB9 connector from radioshack. Pick three different colors of wire, one for TX, one for RX, and one for ground. Solder your TX wire to pin 2 of the DB9 connector, RX wire to pin 3 and Ground to pin 5. Connect pins 1 and 6 to pin 4 and pin 7 to pin 8. Heatshrink the wire connections to avoid accidental shorts.

to:

If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. To do this, pick up a female DB9 connector from radioshack. Pick three different colors of wire, one for TX, one for RX, and one for ground. Solder your TX wire to pin 2 of the DB9 connector, RX wire to pin 3 and Ground to pin 5. Connect pins 1 and 6 to pin 4 and pin 7 to pin 8. Heatshrink the wire connections to avoid accidental shorts. Enclose the connector in a backshell to further protect the signal and enable easy unplugging from your serial port.

Restore
August 17, 2006, at 10:01 AM by Heather Dewey-Hagborg -
Changed line 9 from:

- MAX233 chip

to:

- MAX233 chip (or similar)

Added lines 40-110:

### Program the Arduino

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. Upload the follwoing code into the Arduino microcontroller module:

```
#include <ctype.h>

#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94

byte rx = 6;
byte tx = 7;
byte SWval;

void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  digitalWrite(13,HIGH); //turn on debugging LED
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}

void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}

int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfBit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}

void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```

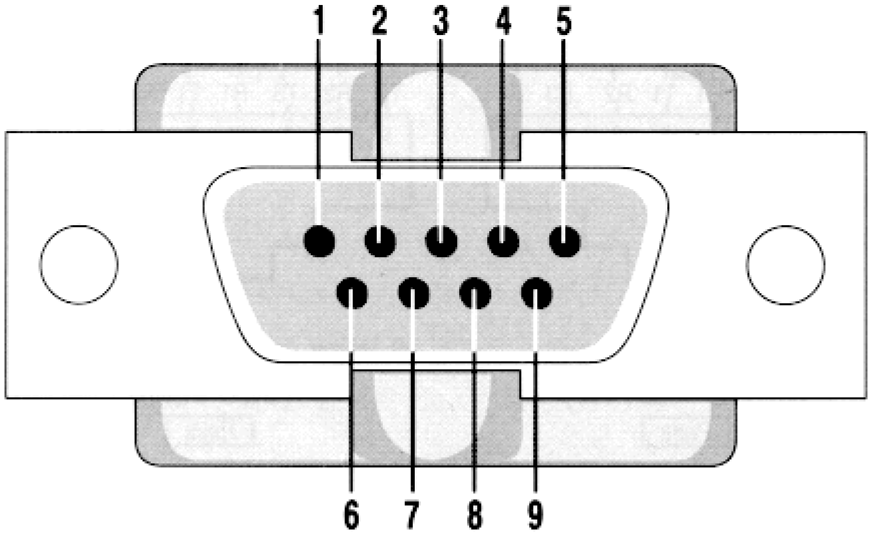
Restore

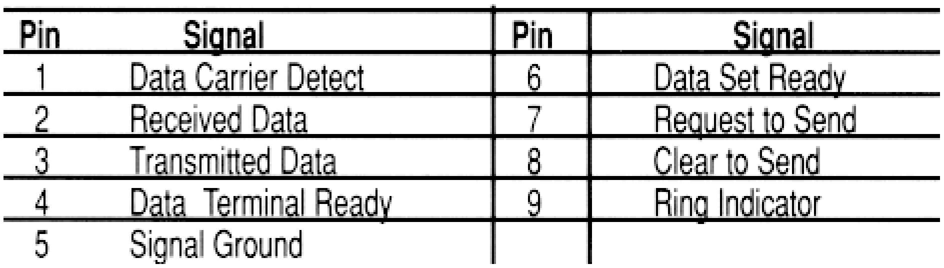
August 15, 2006, at 04:37 PM by Heather Dewey-Hagborg -
Changed lines 30-31 from:

If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. Instructions for doing this can be found .

to:

If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. To do this, pick up a female DB9 connector from radioshack. Pick three different colors of wire, one for TX, one for RX, and one for ground. Solder your TX wire to pin 2 of the DB9 connector, RX wire to pin 3 and Ground to pin 5. Connect pins 1 and 6 to pin 4 and pin 7 to pin 8. Heatshrink the wire connections to avoid accidental shorts.



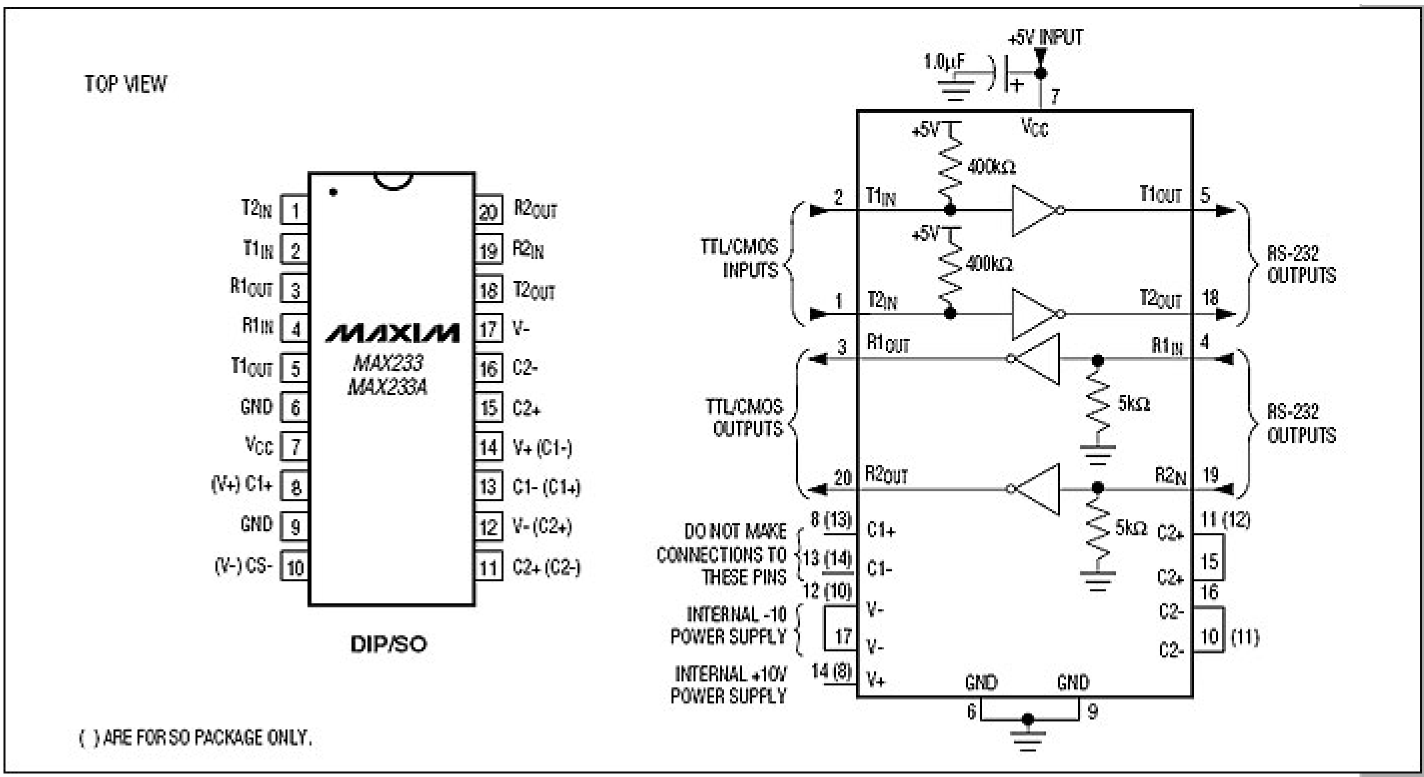


| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data  Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

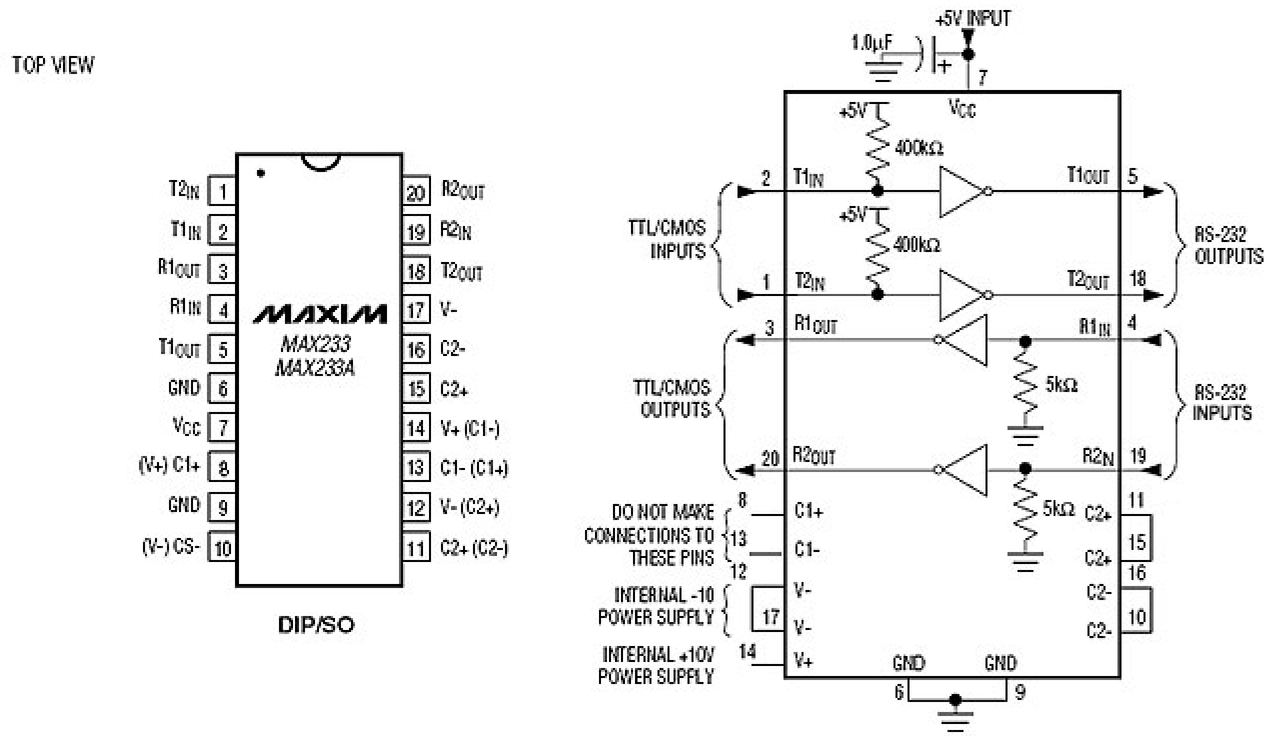
PICTURE connector soldered

Restore

August 15, 2006, at 04:17 PM by Heather Dewey-Hagborg -
Changed lines 18-19 from:



to:



Restore

August 15, 2006, at 03:53 PM by Heather Dewey-Hagborg -

Changed lines 22-23 from:

Attach:rs232pwr_web.jpg Δ

to:

PICTURE

Changed lines 28-29 from:

Attach:rs232ttl_web.jpg Δ

to:

PICTURE

Added lines 32-35:

Connect the TX line from your computer to pin 4 (R1IN) on the MAX233 and the RX line to pin 5 (T1OUT).

PICTURE

Restore
August 15, 2006, at 03:49 PM by Heather Dewey-Hagborg -
Added line 6:
Added line 8:

- Serial-Breadboard cable

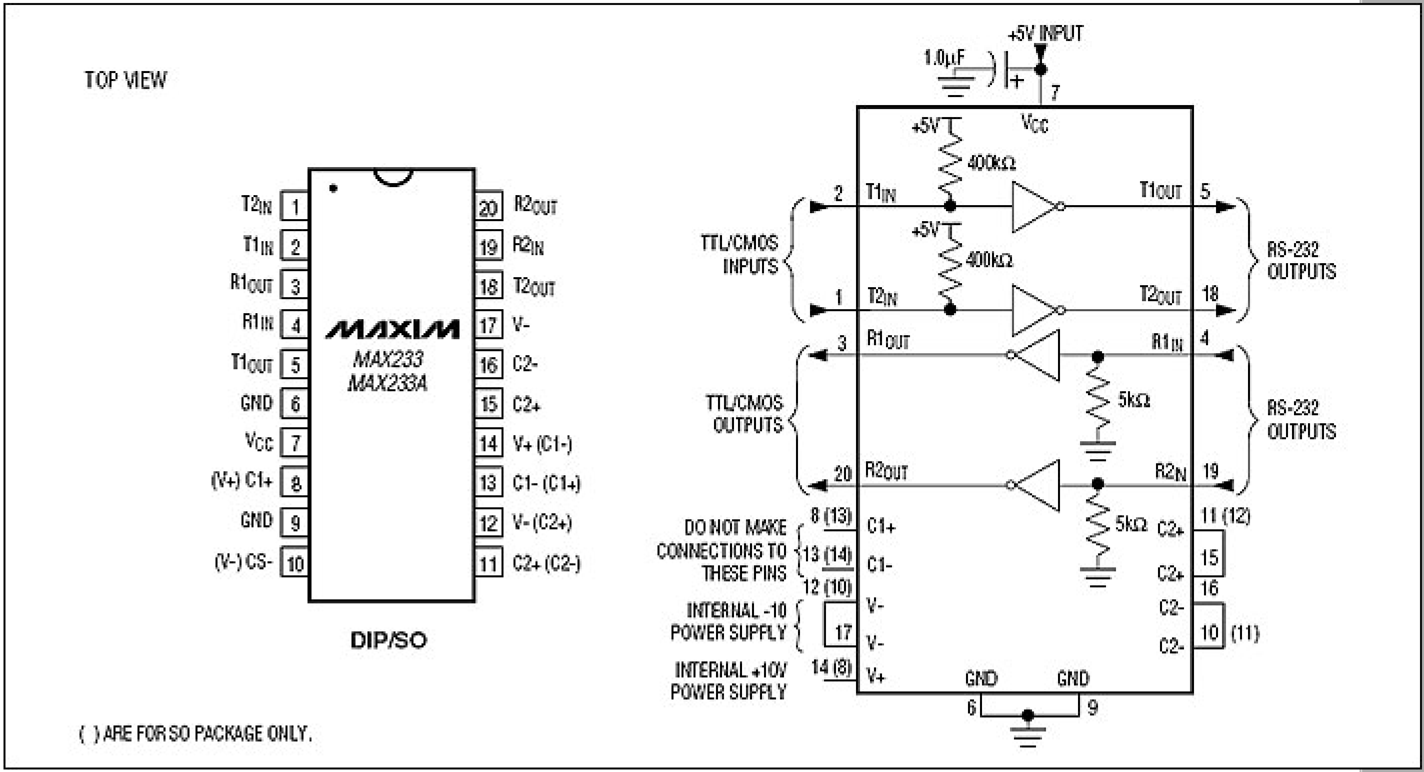
Changed lines 14-31 from:

- Light emitting Diode (LED) - optional, for debugging

to:

- Light emitting Diode (LED) - optional, for debugging

## Prepare the breadboard



Insert the MAX233 chip in the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect pin 6 and pin 9 on the MAX233 chip to ground and pin 7 to 5V. Connect the 1uF capacitor across pins 6 and 7 so that the negative pin connects to pin 6 and the positive pin to pin 7. Connect pin 10 to pin 16 pin 11 to pin 15 and pin 12 to pin 17 on the breadboard.

Attach:rs232pwr_web.jpg Δ

The MAX233 chip has two sets of RS-232 line shifters built in and can handle two simultaneous duplex serial ports. For the purposes of this tutorial we will only being using one port, with corresponding pins referred to as T1IN, T1OUT, R1IN and R1OUT in the MAX233 schematic.

Determine which Arduino pins you want to use for your transmit (TX) and recieve (RX) lines. In this tutorial we will be using Arduino pin 6 for receiving and pin 7 for transmitting. Connect your TX pin (7) to MAX233 pin 2 (T1IN). Connect your RX pin (6) to MAX233 pin 3 (R1OUT).

Attach:rs232ttl_web.jpg Δ

If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. Instructions for doing this can be found .

Restore
August 15, 2006, at 03:23 PM by Heather Dewey-Hagborg -
Changed lines 6-13 from:

```
    * Computer with a terminal program installed (ie. HyperTerminal or RealTerm on the PC, Zterm on Mac)
    * MAX233 chip
    * 1uf polarized capacitor
    * Solderless breadboard
    * Hookup wire
    * Arduino Microcontroller Module
    * Light emitting Diode (LED) - optional, for debugging
```

to:

- Computer with a terminal program installed (ie. HyperTerminal or RealTerm on the PC, Zterm on Mac)
- MAX233 chip
- 1uf polarized capacitor
- Solderless breadboard
- Hookup wire
- Arduino Microcontroller Module
- Light emitting Diode (LED) - optional, for debugging

Restore
August 15, 2006, at 03:22 PM by Heather Dewey-Hagborg -
Added lines 1-13:

# RS-232

In this tutorial you will learn how to communicate with a computer using a MAX233 multichannel RS-232 driver/receiver and a software serial connection on the Arduino.

Materials needed:

```
    * Computer with a terminal program installed (ie. HyperTerminal or RealTerm on the PC, Zterm on Mac)
    * MAX233 chip
    * 1uf polarized capacitor
    * Solderless breadboard
    * Hookup wire
    * Arduino Microcontroller Module
    * Light emitting Diode (LED) - optional, for debugging
```

Restore

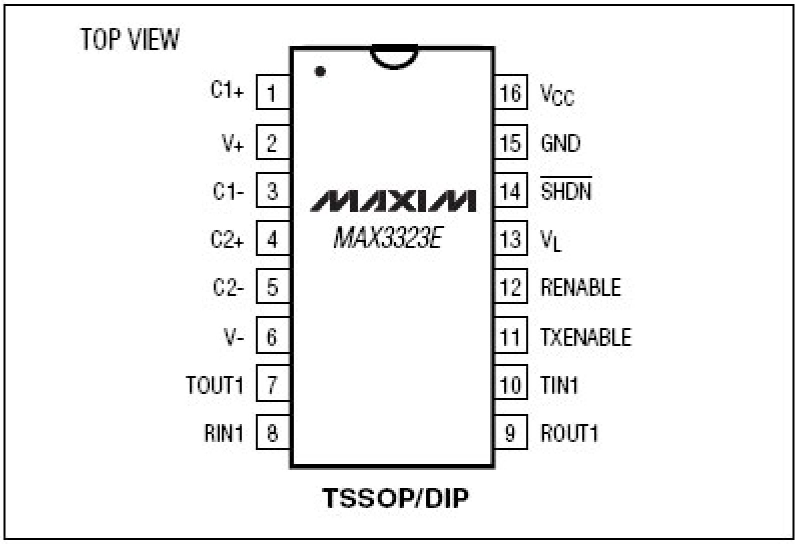
# RS-232

In this tutorial you will learn how to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino. A general purpose software serial tutorial can be found here.

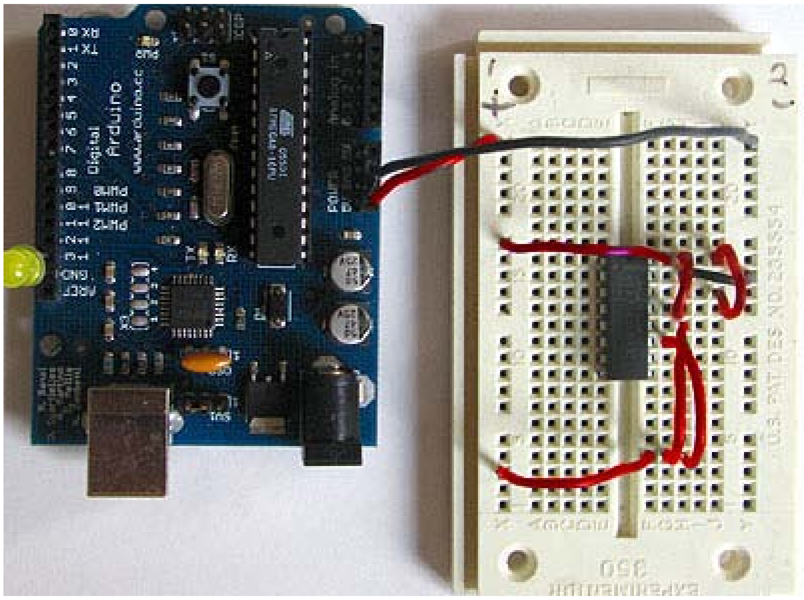
Materials needed:

- Computer with a terminal program installed (ie. HyperTerminal or RealTerm on the PC, Zterm on Mac)
- Serial-Breadboard cable
- MAX3323 chip (or similar)
- 4 1uf capacitors
- Solderless breadboard
- Hookup wire
- Arduino Microcontroller Module
- Light emitting Diode (LED) - optional, for debugging

## Prepare the breadboard

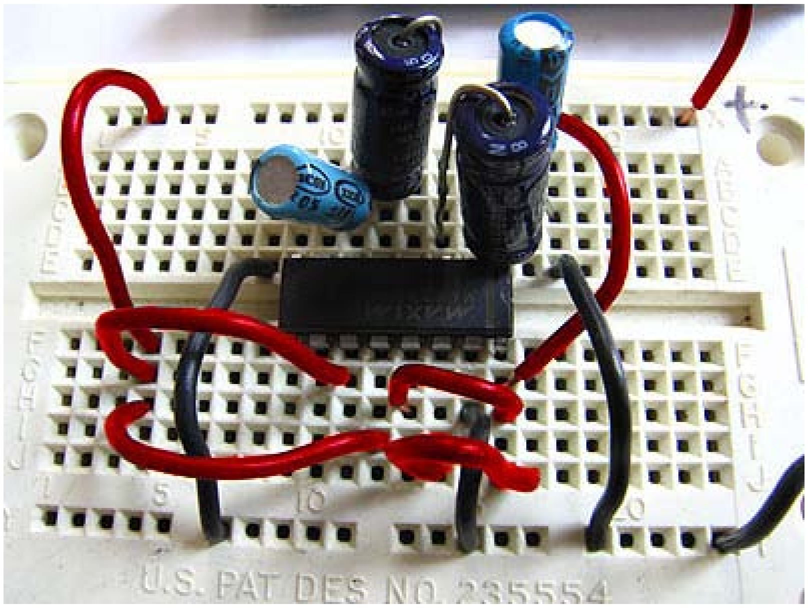


Insert the MAX3323 chip in the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect pin 15 on the MAX233 chip to ground and pins 16 and 14 - 11 to 5V. If you are using an LED connect it between pin 13 and ground.



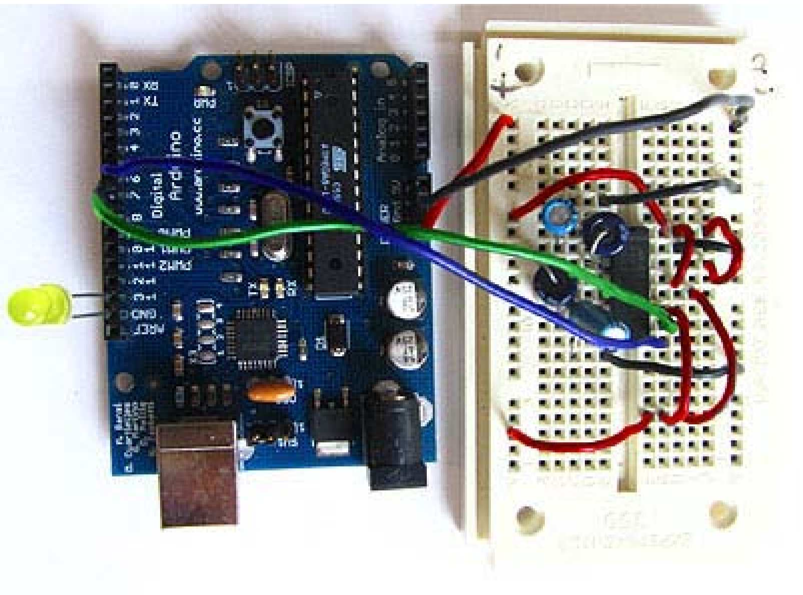
*+5v wires are red, GND wires are black*

Connect a 1uF capacitor across pins 1 and 3, another across pins 4 and 5, another between pin 1 and ground, and the last between pin 6 and ground. If you are using polarized capacitors make sure the negative pins connect to the negative sides (pins 3 and 5 and ground).



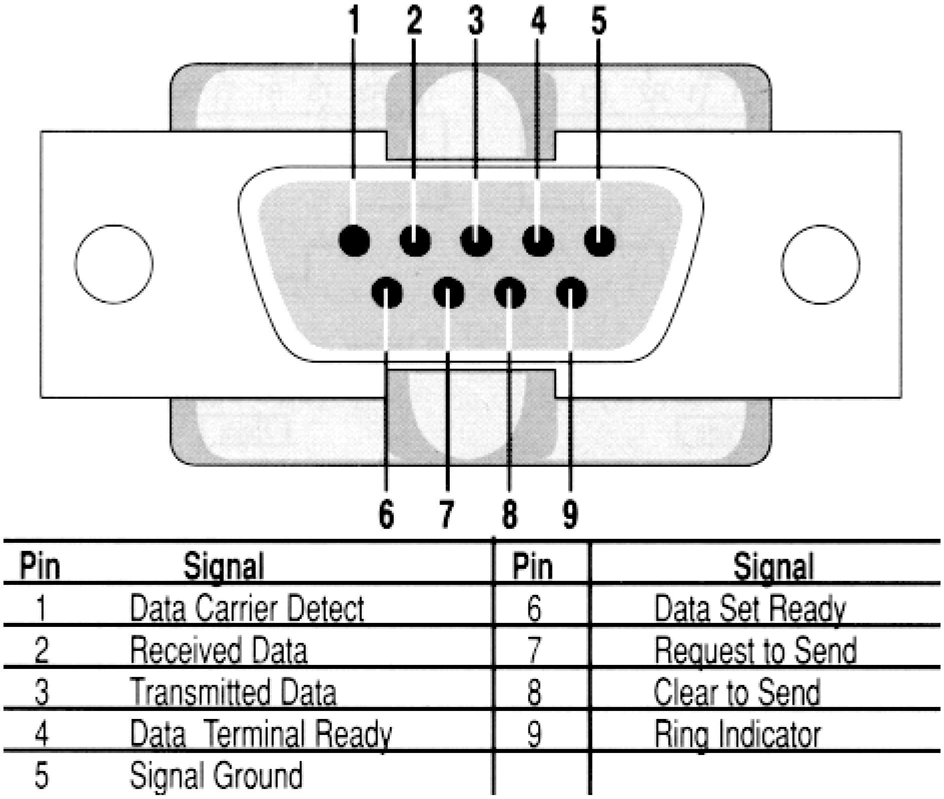
*+5v wires are red, GND wires are black*

Determine which Arduino pins you want to use for your transmit (TX) and recieve (RX) lines. In this tutorial we will be using Arduino pin 6 for receiving and pin 7 for transmitting. Connect your TX pin (7) to MAX3323 pin 10 (T1IN). Connect your RX pin (6) to MAX3323 pin 9 (R1OUT).



*TX wire Green, RX wire Blue, +5v wires are red, GND wires are black*

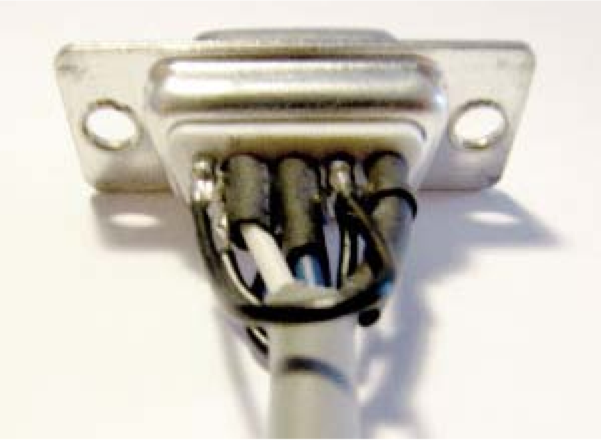
## Cables



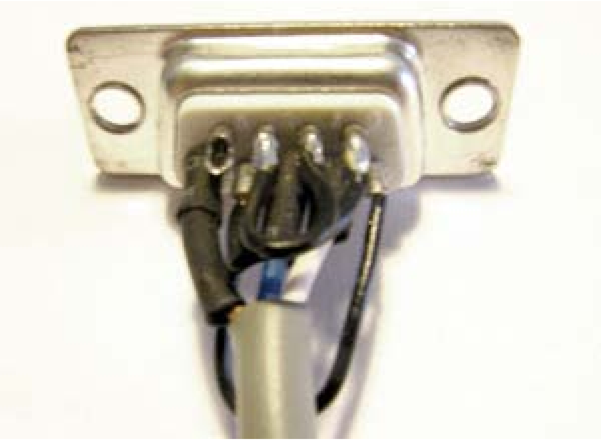
| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data  Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

*(DB9 Serial Connector Pin Diagram)*

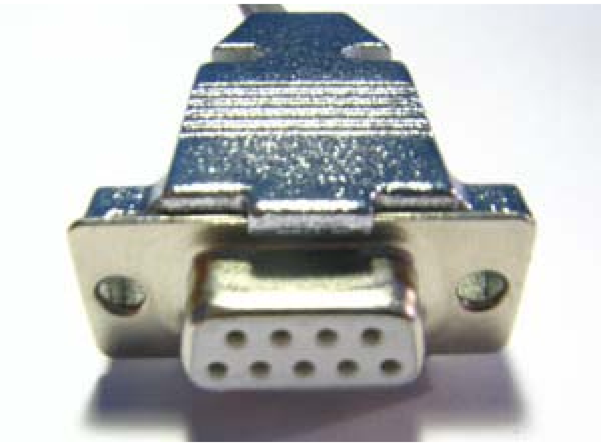
If you do not have one already, you need to make a cable to connect from the serial port (or USB-serial adapter) on your computer and the breadboard. To do this, pick up a female DB9 connector from radioshack. Pick three different colors of wire, one for TX, one for RX, and one for ground. Solder your TX wire to pin 2 of the DB9 connector, RX wire to pin 3 and Ground to pin 5.



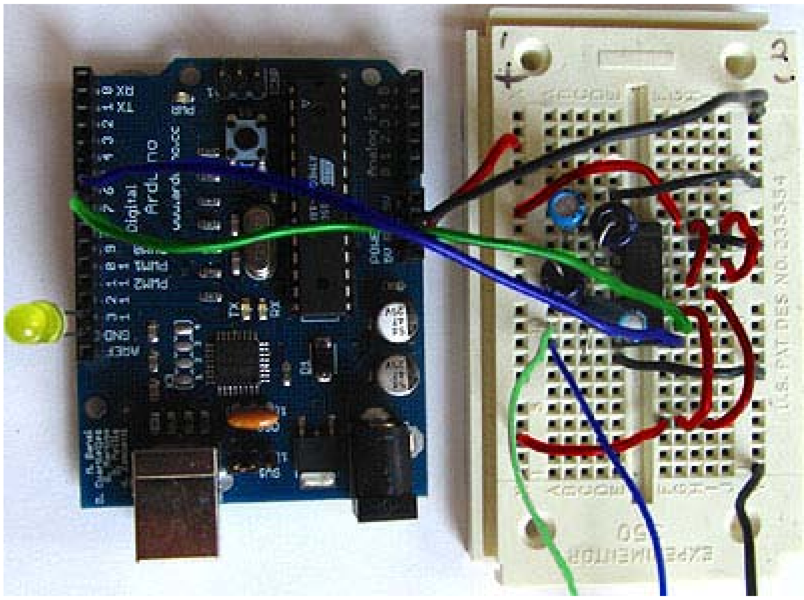
Connect pins 1 and 6 to pin 4 and pin 7 to pin 8. Heatshrink the wire connections to avoid accidental shorts.



Enclose the connector in a backshell to further protect the signal and enable easy unplugging from your serial port.



Connect the TX line from your computer to pin 8 (R1IN) on the MAX233 and the RX line to pin 7 (T1OUT). Connect the ground line from your computer to ground on the breadboard.



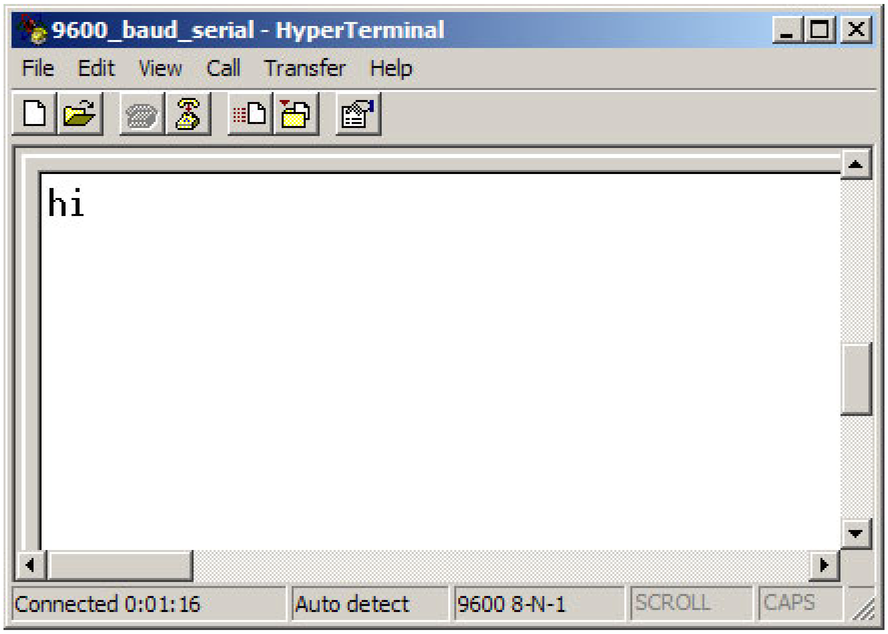
*TX wires Green, RX wires Blue, +5v wires are red, GND wires are black*

## Program the Arduino

Now we will write the code to enable serial data communication. This program will simply wait for a character to arrive in the serial recieving port and then spit it back out in uppercase out the transmit port. This is a good general purpose serial debugging program and you should be able to extrapolate from this example to cover all your basic serial needs. Upload the following code into the Arduino microcontroller module:

```
//Created August 23 2006
//Heather Dewey-Hagborg
//http://www.arduino.cc

#include <ctype.h>

#define bit9600Delay 84
#define halfBit9600Delay 42
#define bit4800Delay 188
#define halfBit4800Delay 94

byte rx = 6;
byte tx = 7;
byte SWval;

void setup() {
  pinMode(rx,INPUT);
  pinMode(tx,OUTPUT);
  digitalWrite(tx,HIGH);
  digitalWrite(13,HIGH); //turn on debugging LED
  SWprint('h');  //debugging hello
  SWprint('i');
  SWprint(10); //carriage return
}

void SWprint(int data)
{
  byte mask;
  //startbit
  digitalWrite(tx,LOW);
  delayMicroseconds(bit9600Delay);
  for (mask = 0x01; mask>0; mask <<= 1) {
    if (data & mask){ // choose bit
     digitalWrite(tx,HIGH); // send 1
    }
    else{
     digitalWrite(tx,LOW); // send 0
    }
    delayMicroseconds(bit9600Delay);
  }
  //stop bit
  digitalWrite(tx, HIGH);
  delayMicroseconds(bit9600Delay);
}

int SWread()
{
  byte val = 0;
  while (digitalRead(rx));
  //wait for start bit
  if (digitalRead(rx) == LOW) {
    delayMicroseconds(halfBit9600Delay);
    for (int offset = 0; offset < 8; offset++) {
     delayMicroseconds(bit9600Delay);
     val |= digitalRead(rx) << offset;
    }
    //wait for stop bit + extra
    delayMicroseconds(bit9600Delay);
    delayMicroseconds(bit9600Delay);
    return val;
  }
}

void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```

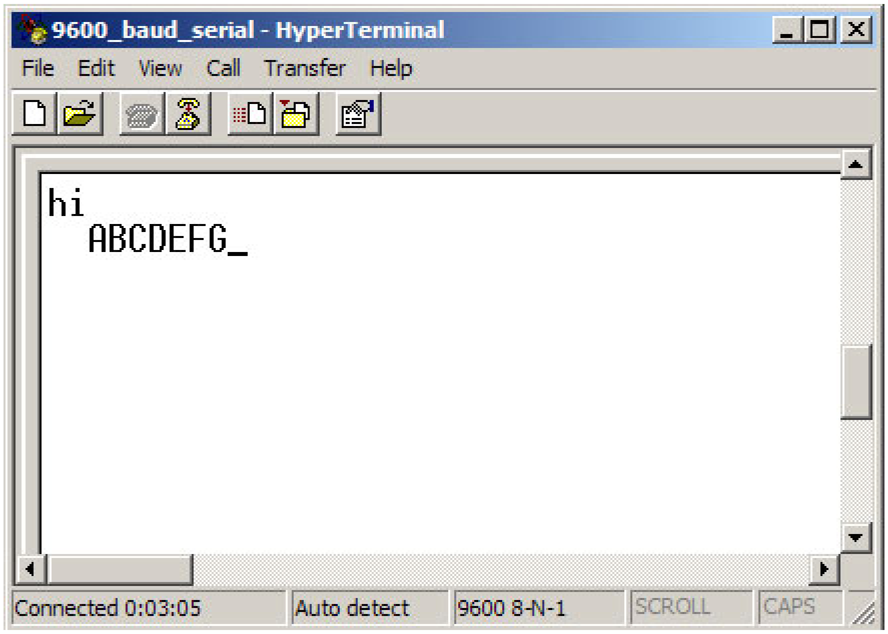
Open up your serial terminal program and set it to 9600 baud, 8 data bits, 1 stop bit, no parity, no hardware flow control. Press the reset button on the arduino board. The word "hi" should appear in the terminal window followed by an advancement to the next line. Here is a shot of what it should look like in Hyperterminal, the free pre-installed windows terminal application.


9600_baud_serial - HyperTerminal
File Edit View Call Transfer Help

```
hi
```

Connected 0:01:16
Auto detect
9600 8-N-1
SCROLL
CAPS

Now, try typing a lowercase character into the terminal window. You should see the letter you typed return to you in uppercase.


9600_baud_serial - HyperTerminal
File Edit View Call Transfer Help
Connected 0:03:05
Auto detect
9600 8-N-1
SCROLL
CAPS

```
hi
  ABCDEFG_
```

If this works, congratulations! Your serial connection is working as planned. You can now use your new serial/computer connection to print debugging statements from your code, and to send commands to your microcontroller.

*code and tutorial by Heather Dewey-Hagborg, photos by Thomas Dexter*

(Printable View of http://www.arduino.cc/en/Tutorial/ArduinoSoftwareRS232)

## Tutorial.SPIEEPROM History

Hide minor edits - Show changes to markup

October 07, 2006, at 11:30 AM by Heather Dewey-Hagborg -
Changed lines 16-17 from:

- Master In Slave Out (MISO) - The Master line for sending data to the peripherals,
- Master Out Slave In (MOSI) - The Slave line for sending data to the master,

to:

- Master In Slave Out (MISO) - The Slave line for sending data to the master,
- Master Out Slave In (MOSI) - The Master line for sending data to the peripherals,

Restore
September 06, 2006, at 04:30 PM by David A. Mellis - adding note about internal eeprom on arduino
Changed lines 3-4 from:

In this tutorial you will learn how to interface with an AT25HP512 Atmel serial EEPROM using the Serial Peripheral Interface (SPI) protocol. EEPROM chips such as this are very useful for data storage, and the steps we will cover for implementing SPI communication can be modified for use with most other SPI devices.

to:

In this tutorial you will learn how to interface with an AT25HP512 Atmel serial EEPROM using the Serial Peripheral Interface (SPI) protocol. EEPROM chips such as this are very useful for data storage, and the steps we will cover for implementing SPI communication can be modified for use with most other SPI devices. Note that the chip on the Arduino board contains an internal EEPROM, so follow this tutorial only if you need more space than it provides.

Restore
September 06, 2006, at 03:38 PM by Heather Dewey-Hagborg -
Changed lines 7-10 from:

1. AT25HP512 Serial EEPROM chip (or similar)
2. Hookup wire
3. Arduino Microcontroller Module

to:

- AT25HP512 Serial EEPROM chip (or similar)
- Hookup wire
- Arduino Microcontroller Module

Restore
September 05, 2006, at 12:57 PM by Heather Dewey-Hagborg -
Changed lines 329-331 from:

@]

to:

@]

*code and tutorial by Heather Dewey-Hagborg, photos by Thomas Dexter*

Restore
August 31, 2006, at 01:19 PM by Heather Dewey-Hagborg -
Changed lines 70-71 from:

Connect EEPROM pin 1 to Arduino pin 10 (Slave Select), EEPROM pin 2 to Arduino pin 12 (Master In Slave Out), EEPROM pin 5 to Arduino pin 11 (Master Out Slave In), and EEPROM pin 6 to Arduino pin 13 (Serial Clock).

to:

Connect EEPROM pin 1 to Arduino pin 10 (Slave Select - SS), EEPROM pin 2 to Arduino pin 12 (Master In Slave Out - MISO), EEPROM pin 5 to Arduino pin 11 (Master Out Slave In - MOSI), and EEPROM pin 6 to Arduino pin 13 (Serial Clock - SCK).

Restore
August 31, 2006, at 01:17 PM by Heather Dewey-Hagborg -
Changed lines 68-69 from:
to:

*+5v wires are red, GND wires are black*

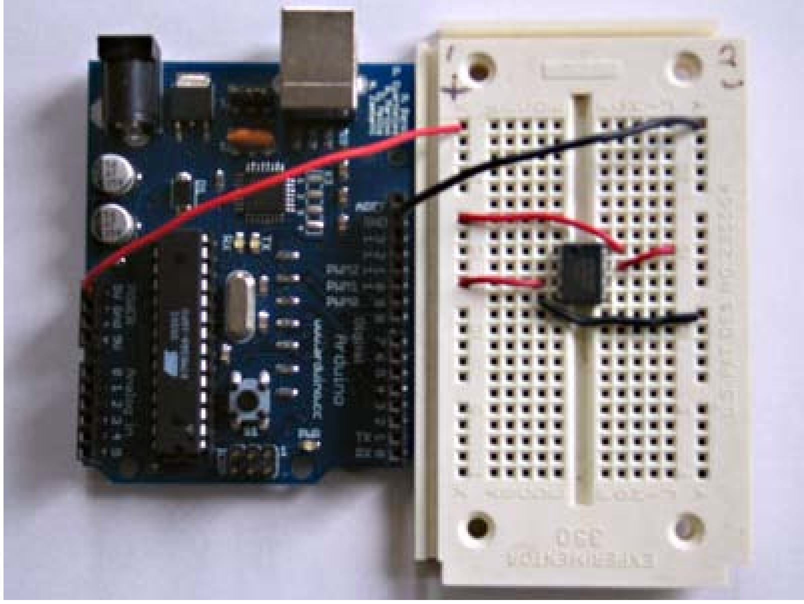
Changed lines 73-74 from:
to:

*SS wire is white, MISO wire is yellow, MOSI wire is blue, SCK wire is green*

Restore
August 31, 2006, at 01:13 PM by Heather Dewey-Hagborg -
Changed lines 67-68 from:

PICTURE of pwr wires

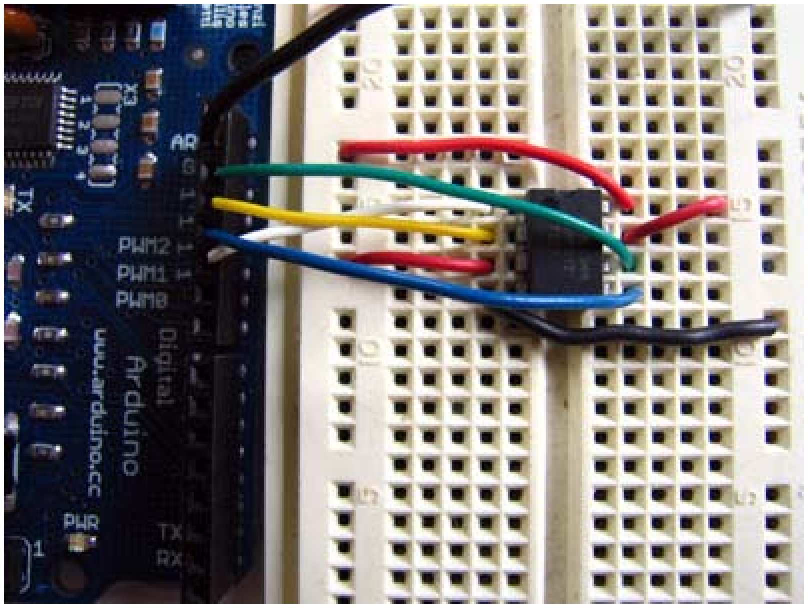
to:



Changed lines 71-72 from:

PICTURE of SPI wires

to:



Restore
August 30, 2006, at 11:08 AM by Heather Dewey-Hagborg -
Changed lines 2-3 from:

(IN PROGRESS)

to:
Restore
August 30, 2006, at 11:05 AM by Heather Dewey-Hagborg -
Changed lines 180-182 from:

This function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application:

to:

The fill_buffer function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application:

Changed lines 191-192 from:

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

to:

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

Changed lines 203-204 from:

This function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data:

to:

The read_eeprom function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data:

Restore

August 30, 2006, at 11:05 AM by Heather Dewey-Hagborg -
Added lines 78-81:

The first step is setting up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

We define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Then we define our opcodes for the EEPROM. Opcodes are control commands:

Changed lines 96-99 from:

Here we set up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

First we define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Then we define our opcodes for the EEPROM. Opcodes are control commands.

to:

Here we allocate the global variables we will be using later in the program. Note char `buffer [128];`. this is a 128 byte array we will be using to store the data for the EEPROM write:

Changed lines 106-107 from:

Here we allocate the global variables we will be using later in the program. Note char `buffer [128];`. this is a 128 byte array we will be using to store the data for the EEPROM write.

to:

First we initialize our serial connection, set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise:

Changed lines 119-120 from:

First we initialize our serial connection, set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise.

to:

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variable to clear out any spurious data from past runs:

Changed lines 131-133 from:

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variabl to clear out any spurious data from past runs.

to:

Here we fill our data array with numbers and send a write enable instruction to the EEPROM. The EEPROM MUST be write enabled before every write instruction. To send the instruction we pull the SLAVESELECT line low, enabling the device, and then send the instruction using the spi_transfer function. Note that we use the WREN opcode we defined at the beginning of the program. Finally we pull the SLAVESELECT line high again to release it:

Changed lines 141-142 from:

Here we fill our data array with numbers and send a write enable instruction to the EEPROM. The EEPROM MUST be write enabled before every write instruction. To send the instruction we pull the SLAVESELECT line low, enabling the device, and then send the instruction using the spi_transfer function. Note that we use the WREN opcode we defined at the beginning of the program. Finally we pull the SLAVESELECT line high again to release it.

to:

Now we pull the SLAVESELECT line low to select the device again after a brief delay. We send a WRITE instruction to tell the

EEPROM we will be sending data to record into memory. We send the 16 bit address to begin writing at in two bytes, Most Significant Bit first. Next we send our 128 bytes of data from our buffer array, one byte after another without pause. Finally we set the SLAVESELECT pin high to release the device and pause to allow the EEPROM to write the data:

Changed lines 159-160 from:

Now we pull the SLAVESELECT line low to select the device again after a brief delay. We send a WRITE instruction to tell the EEPROM we will be sending data to record into memory. We send the 16 bit address to begin writing at in two bytes, Most Significant Bit first. Next we send our 128 bytes of data from our buffer array, one byte after another without pause. Finally we set the SLAVESELECT pin high to release the device and pause to allow the EEPROM to write the data.

to:

We end the setup function by sending the word "hi" plus a line feed out the built in serial port for debugging purposes. This way if our data comes out looking funny later on we can tell it isn't just the serial port acting up:

Changed lines 168-169 from:

We end the setup function by sending the word "hi" plus a line feed out the built in serial port for debugging purposes. This way if our data comes out looking funny later on we can tell it isn't just the serial port acting up.

to:

In our main loop we just read one byte at a time from the EEPROM and print it out the serial port. We add a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 because we have only filled 128 addresses in the EEPROM with data:

Changed lines 181-182 from:

In our main loop we just read one byte at a time from the EEPROM and print it out the serial port. We add a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 because we have only filled 128 addresses in the EEPROM with data.

to:

This function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application:

Changed lines 192-193 from:

This function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application.

to:

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

Changed lines 204-205 from:

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM.

to:

This function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data:

Changed lines 220-222 from:

This function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data.

to:

Restore

August 30, 2006, at 11:01 AM by Heather Dewey-Hagborg -
Changed line 107 from:

void fill_buffer()

to:

void setup()

Deleted lines 108-132:

```
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
```

} @]

This function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application.

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait for the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM.

The following setup function is long so we will take it in parts.

[@ void setup() {

Changed lines 163-164 from:

```
  delay(1000);@]
```

to:

```
  delay(1000);
```

}@]

Changed line 169 from:

byte read_eeprom(int EEPROM_address)

to:

void loop()

Changed lines 171-179 from:

```
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
```

to:

```
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  delay(500); //pause for readability
```

Changed lines 178-179 from:

This function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data.

to:

In our main loop we just read one byte at a time from the EEPROM and print it out the serial port. We add a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 because we have only filled 128 addresses in the EEPROM with data.

Changed line 181 from:

void loop()

to:

void fill_buffer()

Changed lines 183-187 from:

```
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  delay(500); //pause for readability
```

to:

```
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
```

Changed lines 189-193 from:

Finally we get to our main loop, the simplest function in the program! Here we just read one byte at a time from the EEPROM and print it out the serial port. We add a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 because we have only filled 128 addresses in the EEPROM with data.

For easy copy and pasting the full program text of this tutorial is below:

to:

This function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application.

Changed lines 192-212 from:

1. define DATAOUT 11//MOSI
2. define DATAIN 12//MISO
3. define SPICLOCK 13//sck
4. define SLAVESELECT 10//ss

//opcodes

1. define WREN 6
2. define WRDI 4
3. define RDSR 5
4. define WRSR 1
5. define READ 3
6. define WRITE 2

byte eeprom_output_data; byte eeprom_input_data=0; byte clr; int address=0; //data buffer char buffer [128];

void fill_buffer()

to:

char spi_transfer(volatile char data)

Changed lines 194-195 from:

```
  for (int I=0;I<128;I++)
```

to:

```
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait for the end of the transmission
```

Deleted lines 196-204:

```
    buffer[I]=I;
  }
```

}

char spi_transfer(volatile char data) {

```
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
```

Changed lines 199-201 from:

}

void setup()

to:

}@]

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM.

[@ byte read_eeprom(int EEPROM_address)

Changed lines 206-222 from:

```
  Serial.begin(9600);
```

```
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  //fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
```

to:

```
  //READ EEPROM
  int data;
```

Deleted lines 208-234:

```
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
  delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
  Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
```

}

byte read_eeprom(int EEPROM_address) {

```
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
```

Changed lines 215-228 from:

}

void loop() {

```
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  if (address == 128)
    address = 0;
  delay(500); //pause for readability
```

} @]

to:

} @]

This function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data.

For easy copy and pasting the full program text of this tutorial is below:

Added lines 223-327:

1. define DATAOUT 11//MOSI
2. define DATAIN 12//MISO
3. define SPICLOCK 13//sck
4. define SLAVESELECT 10//ss

//opcodes

1. define WREN 6
2. define WRDI 4
3. define RDSR 5
4. define WRSR 1
5. define READ 3

6. define WRITE 2

byte eeprom_output_data; byte eeprom_input_data=0; byte clr; int address=0; //data buffer char buffer [128];

void fill_buffer() {

```
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
```

}

char spi_transfer(volatile char data) {

```
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
```

}

void setup() {

```
  Serial.begin(9600);

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  //fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
  delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
  Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
```

}

byte read_eeprom(int EEPROM_address) {

```
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
```

}

void loop() {

```
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  if (address == 128)
    address = 0;
  delay(500); //pause for readability
```

} @]

[@

Restore
August 30, 2006, at 10:57 AM by Heather Dewey-Hagborg -
Changed lines 53-54 from:

Once you have your SPI Control Register set correctly you just need to figure out how long you need to pause between instructions and you are ready to go. Now that you have a feel for how SPI works, let's take a look at the EEPROM chip.

to:

Once you have your SPI Control Register set correctly you just need to figure out how long you need to pause between instructions and you are ready to go. Now that you have a feel for how SPI works, let's take a look at the details of the EEPROM chip.

Changed lines 62-63 from:

The device is enabled by pulling the Chip Select (CS) pin low. Instructions are 8 bit opcodes and are shifted in on the rising edge of the data clock. It takes the EEPROM about 10 milliseconds to write a page (128 bytes) of data, so a 10ms pause should follow each EEPROM write routine.

to:

The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 8 bit operational codes (opcodes) and are shifted in on the rising edge of the data clock. It takes the EEPROM about 10 milliseconds to write a page (128 bytes) of data, so a 10ms pause should follow each EEPROM write routine.

Changed lines 127-128 from:

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete. It then returns any data that has been shifted in to the data register by the EEPROM.

to:

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM.

Changed lines 154-155 from:

After setting our control register up we clear any spurious data from the Status and Control registers.

to:

After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variabl to clear out any spurious data from past runs.

Restore

August 30, 2006, at 10:51 AM by Heather Dewey-Hagborg -
Changed lines 53-54 from:

Once you have your SPI Control Register set correctly you just need to figure out how long you need to pause between instructions and you are ready to go.

to:

Once you have your SPI Control Register set correctly you just need to figure out how long you need to pause between instructions and you are ready to go. Now that you have a feel for how SPI works, let's take a look at the EEPROM chip.

Restore
August 30, 2006, at 10:44 AM by Heather Dewey-Hagborg -
Changed lines 24-32 from:

All SPI settings are determined by the Arduino SPI Control Register (SPCR). The SPCR has 8 bits each of which control a particular SPI setting.

to:

All SPI settings are determined by the Arduino SPI Control Register (SPCR). A register is just a byte of microcontroller memory that can be read from or written to. Registers generally serve three purposes, control, data and status.

Control registers code control settings for various microcontroller functionalities. Usually each bit in a control register effects a particular setting, such as speed or polarity.

Data registers simply hold bytes. For example, the SPI data register (SPDR) holds the byte which is about to be shifted out the MOSI line, and the data which has just been shifted in the MISO line.

Status registers change their state based on various microcontroller conditions. For example, the seventh bit of the SPI status register (SPSR) gets set to 1 when a value is shifted in or out of the SPI.

The SPI control register (SPCR) has 8 bits, each of which control a particular SPI setting.

Changed lines 48-52 from:

-Is data shifted in MSB or LSB first? -Is the data clock idle when high or low? -Are samples on the rising or falling edge of clock pulses? -What speed is the SPI running at?

to:

- Is data shifted in MSB or LSB first?
- Is the data clock idle when high or low?
- Are samples on the rising or falling edge of clock pulses?
- What speed is the SPI running at?

Restore
August 30, 2006, at 09:58 AM by Tom Igoe -
Changed lines 14-15 from:

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by Microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers. With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices, Master In Slave Out (MISO) - The Master line for sending data to the peripherals, Master Out Slave In (MOSI) - The Slave line for sending data to the master, and Serial Clock (SCK) - The clock pulses which synchronize data transmission generated by the master. Additionally there is generally a Slave Select pin allocated on each device which the master can use to enable and disable specific devices and avoid false transmissions due to line noise.

to:

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by Microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices,

- Master In Slave Out (MISO) - The Master line for sending data to the peripherals,
- Master Out Slave In (MOSI) - The Slave line for sending data to the master,
- Serial Clock (SCK) - The clock pulses which synchronize data transmission generated by the master, and
- Slave Select pin - allocated on each device which the master can use to enable and disable specific devices and avoid false transmissions due to line noise.

Restore
August 29, 2006, at 06:51 PM by Heather Dewey-Hagborg -
Added lines 46-49:

The AT25HP512 is a 65,536 byte serial EEPROM. It supports SPI modes 0 and 3, runs at up to 10MHz at 5v and can run at slower speeds down to 1.8v. It's memory is organized as 512 pages of 128 bytes each. It can only be written 128 bytes at a time, but it can be read 1-128 bytes at a time. The device also offers various degerees of write protection and a hold pin, but we won't be covering those in this tutorial.

The device is enabled by pulling the Chip Select (CS) pin low. Instructions are 8 bit opcodes and are shifted in on the rising edge of the data clock. It takes the EEPROM about 10 milliseconds to write a page (128 bytes) of data, so a 10ms pause should follow each EEPROM write routine.

Restore
August 29, 2006, at 05:46 PM by Heather Dewey-Hagborg -
Changed lines 16-17 from:

The difficult part about SPI is that the standard is loose and each device implements it a little differently. Generally speaking there are three modes of transmission numbered 0 - 3. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal, and whether the clock is idle when high or low.

to:

The difficult part about SPI is that the standard is loose and each device implements it a little differently. This means you have to pay special attention to the datasheet when writing your interface code. Generally speaking there are three modes of transmission numbered 0 - 3. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal, and whether the clock is idle when high or low.

Changed lines 33-35 from:

The eighth bit sets the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first or Least Significant, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest).

to:

This means that to write code for a new SPI device you need to note several things and set the SPCR accordingly: -Is data shifted in MSB or LSB first? -Is the data clock idle when high or low? -Are samples on the rising or falling edge of clock pulses? -What speed is the SPI running at?

Once you have your SPI Control Register set correctly you just need to figure out how long you need to pause between instructions and you are ready to go.

Restore
August 29, 2006, at 04:07 PM by Heather Dewey-Hagborg -
Changed lines 14-15 from:

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by Microcontrollers for communicating with peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers. With an SPI connection

to:

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by Microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers. With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices, Master In Slave Out (MISO) - The Master line for sending data to the peripherals, Master Out Slave In (MOSI) - The Slave line for sending data to the master, and Serial Clock (SCK) - The clock pulses which synchronize data transmission generated by the master. Additionally there is generally a Slave Select pin allocated on each device which the master can use to enable and disable specific devices and avoid false transmissions due to line noise.

The difficult part about SPI is that the standard is loose and each device implements it a little differently. Generally speaking there are three modes of transmission numbered 0 - 3. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal, and whether the clock is idle when high or low.

All SPI settings are determined by the Arduino SPI Control Register (SPCR). The SPCR has 8 bits each of which control a particular SPI setting.

```
SPCR
| 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| SPIE | SPE  | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 |

SPIE - Enables the SPI interrupt when 1
SPE - Enables the SPI when 1
DORD - Sends data least Significant Bit First when 1, most Significant Bit first when 0
MSTR - Sets the Arduino in master mode when 1, slave mode when 0
CPOL - Sets the data clock to be idle when high if set to 1, idle when low if set to 0
CPHA - Samples data on the falling edge of the data clock when 1, rising edge when 0
SPR1 and SPR0 - Sets the SPI speed, 00 is fastest (4MHz) 11 is slowest (250KHz)
```

The eighth bit sets the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first or Least Significant, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest).

Restore
August 29, 2006, at 02:56 PM by Heather Dewey-Hagborg -
Added lines 14-15:

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by Microcontrollers for communicating with peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers. With an SPI connection

Restore
August 29, 2006, at 02:45 PM by Heather Dewey-Hagborg -
Changed lines 33-34 from:

[@#define DATAOUT 11//MOSI

to:

[@

1. define DATAOUT 11//MOSI

Changed lines 51-52 from:

[@byte eeprom_output_data;

to:

[@ byte eeprom_output_data;

Changed lines 61-62 from:

[@void fill_buffer()

to:

[@ void fill_buffer()

Changed lines 72-73 from:

[@char spi_transfer(volatile char data)

to:

[@ char spi_transfer(volatile char data)

Changed lines 86-87 from:

[@void setup()

to:

[@ void setup()

Changed lines 99-100 from:

[@ // SPCR = 01010000

to:

[@

```
  // SPCR = 01010000
```

Changed lines 111-112 from:

[@//fill buffer with data

to:

[@

```
  //fill buffer with data
```

Changed lines 121-122 from:

[@delay(10);

to:

[@

```
  delay(10);
```

Changed lines 139-140 from:

[@Serial.print('h',BYTE);

to:

[@

```
  Serial.print('h',BYTE);
```

Changed lines 147-148 from:

[@byte read_eeprom(int EEPROM_address)

to:

[@ byte read_eeprom(int EEPROM_address)

Changed lines 163-164 from:

[@void loop()

to:

[@ void loop()

Changed lines 178-179 from:

[@#define DATAOUT 11//MOSI

to:

[@

1. define DATAOUT 11//MOSI

Restore
August 29, 2006, at 02:43 PM by Heather Dewey-Hagborg -
Changed lines 162-164 from:

Finally we get to our main loop, the simplest function in the program! Here we just read one byte at a time from the EEPROM and print it out the serial port plus a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 since we have only filled 128 addresses in the EEPROM with data.

to:

Finally we get to our main loop, the simplest function in the program! Here we just read one byte at a time from the EEPROM and print it out the serial port. We add a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 because we have only filled

128 addresses in the EEPROM with data.

Restore
August 29, 2006, at 02:42 PM by Heather Dewey-Hagborg -
Added lines 59-164:

```
void fill_buffer()
{
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
}
```

This function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application.

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait for the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

This function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete. It then returns any data that has been shifted in to the data register by the EEPROM.

The following setup function is long so we will take it in parts.

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

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
```

First we initialize our serial connection, set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise.

```
 // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
```

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we clear any spurious data from the Status and Control registers.

```
//fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
```

Here we fill our data array with numbers and send a write enable instruction to the EEPROM. The EEPROM MUST be write enabled before every write instruction. To send the instruction we pull the SLAVESELECT line low, enabling the device, and then send the instruction using the spi_transfer function. Note that we use the WREN opcode we defined at the beginning of the program. Finally we pull the SLAVESELECT line high again to release it.

```
delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
```

Now we pull the SLAVESELECT line low to select the device again after a brief delay. We send a WRITE instruction to tell the EEPROM we will be sending data to record into memory. We send the 16 bit address to begin writing at in two bytes, Most Significant Bit first. Next we send our 128 bytes of data from our buffer array, one byte after another without pause. Finally we set the SLAVESELECT pin high to release the device and pause to allow the EEPROM to write the data.

```
Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
```

We end the setup function by sending the word "hi" plus a line feed out the built in serial port for debugging purposes. This way if our data comes out looking funny later on we can tell it isn't just the serial port acting up.

```
byte read_eeprom(int EEPROM_address)
{
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
}
```

This function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the `data = spi_transfer(0xFF);` up to 128 times for a full page of data.

```
void loop()
{
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  delay(500); //pause for readability
}
```

Finally we get to our main loop, the simplest function in the program! Here we just read one byte at a time from the EEPROM and print it out the serial port plus a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 since we have only filled 128 addresses in the EEPROM with data.

Added lines 166-270:

```
#define DATAOUT 11//MOSI
#define DATAIN  12//MISO
```

```
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss

//opcodes
#define WREN  6
#define WRDI  4
#define RDSR  5
#define WRSR  1
#define READ  3
#define WRITE 2

byte eeprom_output_data;
byte eeprom_input_data=0;
byte clr;
int address=0;
//data buffer
char buffer [128];

void fill_buffer()
{
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
}

char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}

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

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  //fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
  delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
  Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
}

byte read_eeprom(int EEPROM_address)
{
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
}

void loop()
{
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  if (address == 128)
    address = 0;
  delay(500); //pause for readability
}
```

Restore
August 29, 2006, at 01:15 PM by Heather Dewey-Hagborg -
Added lines 30-61:

Now we will write the code to enable SPI communication between the EEPROM and the Arduino. In the setup routine this program fills 128 bytes, or one page of the EEPROM with data. In the main loop it reads that data back out, one byte at a time and prints that byte out the built in serial port. We will walk through the code in small sections.

```
#define DATAOUT 11//MOSI
#define DATAIN  12//MISO
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss

//opcodes
#define WREN  6
#define WRDI  4
#define RDSR  5
#define WRSR  1
#define READ  3
#define WRITE 2
```

Here we set up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

First we define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Then we define our opcodes for the EEPROM. Opcodes are control commands.

```
byte eeprom_output_data;
byte eeprom_input_data=0;
byte clr;
int address=0;
//data buffer
char buffer [128];
```

Here we allocate the global variables we will be using later in the program. Note `char buffer [128];`. this is a 128 byte array we will be using to store the data for the EEPROM write.

For easy copy and pasting the full program text of this tutorial is below:

Restore
August 29, 2006, at 12:58 PM by Heather Dewey-Hagborg -
Added lines 21-28:

Insert the AT25HP512 chip into the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect EEPROM pins 3, 7 and 8 to 5v and pin 4 to ground.

PICTURE of pwr wires

Connect EEPROM pin 1 to Arduino pin 10 (Slave Select), EEPROM pin 2 to Arduino pin 12 (Master In Slave Out), EEPROM pin 5 to Arduino pin 11 (Master Out Slave In), and EEPROM pin 6 to Arduino pin 13 (Serial Clock).

PICTURE of SPI wires

Restore
August 27, 2006, at 01:17 PM by Heather Dewey-Hagborg -
Changed line 1 from:

## Interfacing a serial EEPROM using SPI

to:

## Interfacing a Serial EEPROM Using SPI

Changed lines 12-15 from:

### Serial Peripheral Interface

### Atmel 25HP512 EEPROM chip

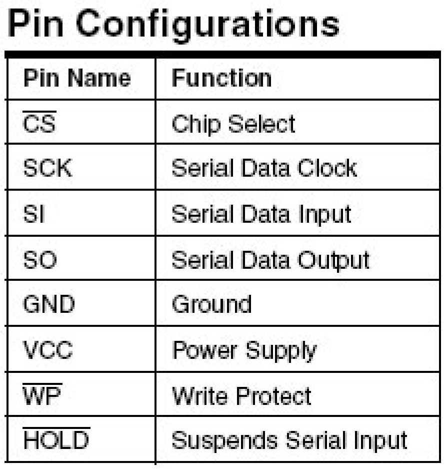
to:

### Introduction to the Serial Peripheral Interface

### Introduction to Serial EEPROM

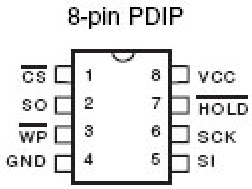
Restore
August 27, 2006, at 01:14 PM by Heather Dewey-Hagborg -
Added lines 14-18:

### Atmel 25HP512 EEPROM chip



Pin Configurations

| Pin Name | Function |
|---|---|
| $\overline{\text{CS}}$ | Chip Select |
| SCK | Serial Data Clock |
| SI | Serial Data Input |
| SO | Serial Data Output |
| GND | Ground |
| VCC | Power Supply |
| $\overline{\text{WP}}$ | Write Protect |
| $\overline{\text{HOLD}}$ | Suspends Serial Input |



Restore
August 27, 2006, at 01:11 PM by Heather Dewey-Hagborg -
Changed lines 2-3 from:
to:

(IN PROGRESS)

Added lines 13-16:

### Prepare the Breadboard

### Program the Arduino

Restore
August 27, 2006, at 01:09 PM by Heather Dewey-Hagborg -
Added lines 1-11:

## Interfacing a serial EEPROM using SPI

In this tutorial you will learn how to interface with an AT25HP512 Atmel serial EEPROM using the Serial Peripheral Interface (SPI) protocol. EEPROM chips such as this are very useful for data storage, and the steps we will cover for implementing SPI communication can be modified for use with most other SPI devices.

Materials Needed:

1. AT25HP512 Serial EEPROM chip (or similar)
2. Hookup wire
3. Arduino Microcontroller Module

### Serial Peripheral Interface

Restore

# Interfacing a Serial EEPROM Using SPI

In this tutorial you will learn how to interface with an AT25HP512 Atmel serial EEPROM using the Serial Peripheral Interface (SPI) protocol. EEPROM chips such as this are very useful for data storage, and the steps we will cover for implementing SPI communication can be modified for use with most other SPI devices. Note that the chip on the Arduino board contains an internal EEPROM, so follow this tutorial only if you need more space than it provides.

Materials Needed:

- AT25HP512 Serial EEPROM chip (or similar)
- Hookup wire
- Arduino Microcontroller Module

## Introduction to the Serial Peripheral Interface

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by Microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices,

- Master In Slave Out (MISO) - The Slave line for sending data to the master,
- Master Out Slave In (MOSI) - The Master line for sending data to the peripherals,
- Serial Clock (SCK) - The clock pulses which synchronize data transmission generated by the master, and
- Slave Select pin - allocated on each device which the master can use to enable and disable specific devices and avoid false transmissions due to line noise.

The difficult part about SPI is that the standard is loose and each device implements it a little differently. This means you have to pay special attention to the datasheet when writing your interface code. Generally speaking there are three modes of transmission numbered 0 - 3. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal, and whether the clock is idle when high or low.

All SPI settings are determined by the Arduino SPI Control Register (SPCR). A register is just a byte of microcontroller memory that can be read from or written to. Registers generally serve three purposes, control, data and status.

Control registers code control settings for various microcontroller functionalities. Usually each bit in a control register effects a particular setting, such as speed or polarity.

Data registers simply hold bytes. For example, the SPI data register (SPDR) holds the byte which is about to be shifted out the MOSI line, and the data which has just been shifted in the MISO line.

Status registers change their state based on various microcontroller conditions. For example, the seventh bit of the SPI status register (SPSR) gets set to 1 when a value is shifted in or out of the SPI.

The SPI control register (SPCR) has 8 bits, each of which control a particular SPI setting.

```
SPCR
| 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| SPIE | SPE  | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 |

SPIE - Enables the SPI interrupt when 1
SPE - Enables the SPI when 1
DORD - Sends data least Significant Bit First when 1, most Significant Bit first when 0
MSTR - Sets the Arduino in master mode when 1, slave mode when 0
CPOL - Sets the data clock to be idle when high if set to 1, idle when low if set to 0
```

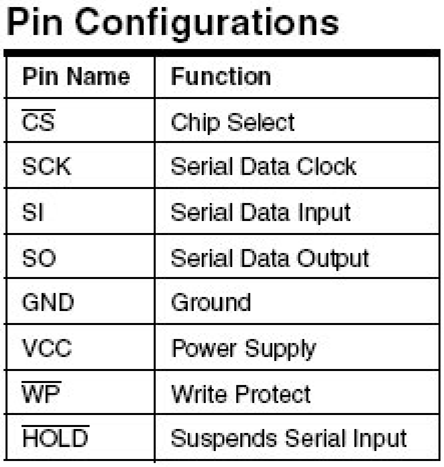
```
CPHA - Samples data on the falling edge of the data clock when 1, rising edge when 0
SPR1 and SPR0 - Sets the SPI speed, 00 is fastest (4MHz) 11 is slowest (250KHz)
```

This means that to write code for a new SPI device you need to note several things and set the SPCR accordingly:

- Is data shifted in MSB or LSB first?
- Is the data clock idle when high or low?
- Are samples on the rising or falling edge of clock pulses?
- What speed is the SPI running at?

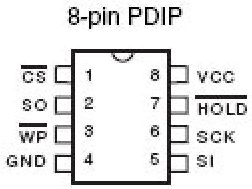
Once you have your SPI Control Register set correctly you just need to figure out how long you need to pause between instructions and you are ready to go. Now that you have a feel for how SPI works, let's take a look at the details of the EEPROM chip.

## Introduction to Serial EEPROM



### Pin Configurations

| Pin Name | Function |
|---|---|
| $\overline{CS}$ | Chip Select |
| SCK | Serial Data Clock |
| SI | Serial Data Input |
| SO | Serial Data Output |
| GND | Ground |
| VCC | Power Supply |
| $\overline{WP}$ | Write Protect |
| $\overline{HOLD}$ | Suspends Serial Input |

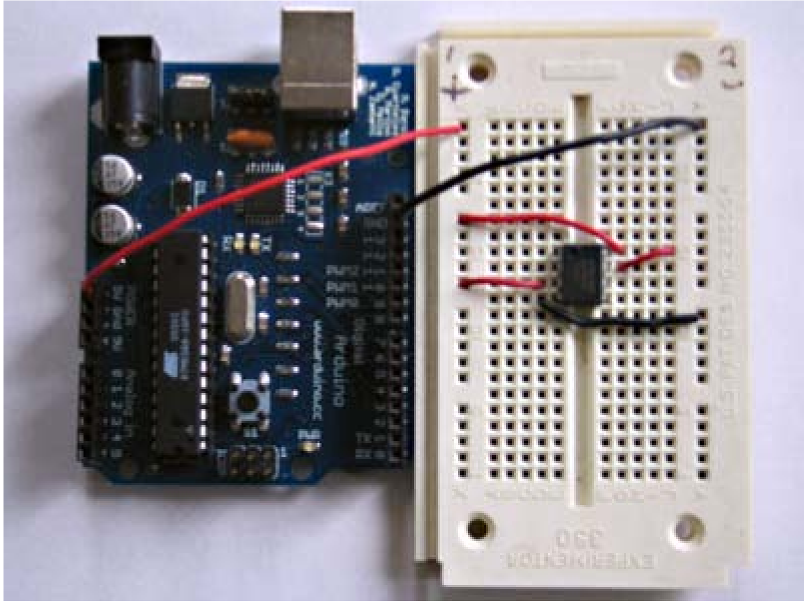


The AT25HP512 is a 65,536 byte serial EEPROM. It supports SPI modes 0 and 3, runs at up to 10MHz at 5v and can run at slower speeds down to 1.8v. It's memory is organized as 512 pages of 128 bytes each. It can only be written 128 bytes at a time, but it can be read 1-128 bytes at a time. The device also offers various degerees of write protection and a hold pin, but we won't be covering those in this tutorial.

The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 8 bit operational codes (opcodes) and are shifted in on the rising edge of the data clock. It takes the EEPROM about 10 milliseconds to write a page (128 bytes) of data, so a 10ms pause should follow each EEPROM write routine.

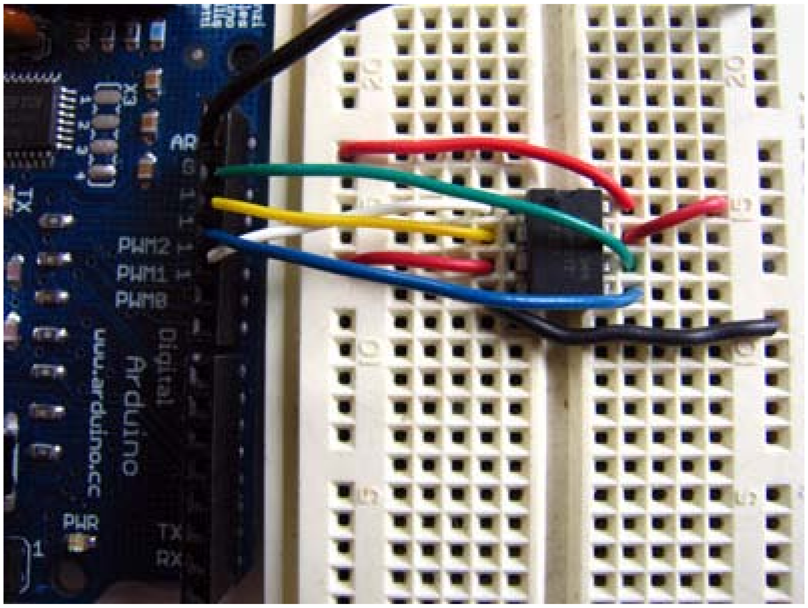
## Prepare the Breadboard

Insert the AT25HP512 chip into the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect EEPROM pins 3, 7 and 8 to 5v and pin 4 to ground.



*+5v wires are red, GND wires are black*

Connect EEPROM pin 1 to Arduino pin 10 (Slave Select - SS), EEPROM pin 2 to Arduino pin 12 (Master In Slave Out - MISO), EEPROM pin 5 to Arduino pin 11 (Master Out Slave In - MOSI), and EEPROM pin 6 to Arduino pin 13 (Serial Clock - SCK).



*SS wire is white, MISO wire is yellow, MOSI wire is blue, SCK wire is green*

## Program the Arduino

Now we will write the code to enable SPI communication between the EEPROM and the Arduino. In the setup routine this program fills 128 bytes, or one page of the EEPROM with data. In the main loop it reads that data back out, one byte at a time and prints that byte out the built in serial port. We will walk through the code in small sections.

The first step is setting up our pre-processor directives. Pre-processor directives are processed before the actual compilation begins. They start with a "#" and do not end with semi-colons.

We define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Then we define our opcodes for the EEPROM. Opcodes are control commands:

```
#define DATAOUT 11//MOSI
```

```
#define DATAIN  12//MISO
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss

//opcodes
#define WREN  6
#define WRDI  4
#define RDSR  5
#define WRSR  1
#define READ  3
#define WRITE 2
```

Here we allocate the global variables we will be using later in the program. Note char buffer [128];. this is a 128 byte array we will be using to store the data for the EEPROM write:

```
byte eeprom_output_data;
byte eeprom_input_data=0;
byte clr;
int address=0;
//data buffer
char buffer [128];
```

First we initialize our serial connection, set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise:

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

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
```

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variable to clear out any spurious data from past runs:

```
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
```

Here we fill our data array with numbers and send a write enable instruction to the EEPROM. The EEPROM MUST be write enabled before every write instruction. To send the instruction we pull the SLAVESELECT line low, enabling the device, and then send the instruction using the spi_transfer function. Note that we use the WREN opcode we defined at the beginning of the program. Finally we pull the SLAVESELECT line high again to release it:

```
  //fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
```

Now we pull the SLAVESELECT line low to select the device again after a brief delay. We send a WRITE instruction to tell the EEPROM we will be sending data to record into memory. We send the 16 bit address to begin writing at in two bytes, Most Significant Bit first. Next we send our 128 bytes of data from our buffer array, one byte after another without pause. Finally we set the SLAVESELECT pin high to release the device and pause to allow the EEPROM to write the data:

```
  delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
```

We end the setup function by sending the word "hi" plus a line feed out the built in serial port for debugging purposes. This way if our data comes out looking funny later on we can tell it isn't just the serial port acting up:

```
  Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
}
```

In our main loop we just read one byte at a time from the EEPROM and print it out the serial port. We add a line feed and a pause for readability. Each time through the loop we increment the eeprom address to read. When the address increments to 128 we turn it back to 0 because we have only filled 128 addresses in the EEPROM with data:

```
void loop()
{
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  delay(500); //pause for readability
}
```

The fill_buffer function simply fills our data array with numbers 0 - 127 for each index in the array. This function could easily be changed to fill the array with data relevant to your application:

```
void fill_buffer()
{
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
}
```

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait for the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

The read_eeprom function allows us to read data back out of the EEPROM. First we set the SLAVESELECT line low to enable the device. Then we transmit a READ instruction, followed by the 16-bit address we wish to read from, Most Significant Bit first. Next we send a dummy byte to the EEPROM for the purpose of shifting the data out. Finally we pull the SLAVESELECT line high again to release the device after reading one byte, and return the data. If we wanted to read multiple bytes at a time we could hold the SLAVESELECT line low while we repeated the data = spi_transfer(0xFF); up to 128 times for a full page of data:

```
byte read_eeprom(int EEPROM_address)
{
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
}
```

For easy copy and pasting the full program text of this tutorial is below:

```
#define DATAOUT 11//MOSI
#define DATAIN  12//MISO
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss

//opcodes
#define WREN  6
#define WRDI  4
#define RDSR  5
#define WRSR  1
#define READ  3
#define WRITE 2

byte eeprom_output_data;
byte eeprom_input_data=0;
byte clr;
int address=0;
//data buffer
char buffer [128];

void fill_buffer()
{
  for (int I=0;I<128;I++)
  {
    buffer[I]=I;
  }
}

char spi_transfer(volatile char data)
{
  SPDR = data;                     // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                     // return the received byte
}

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

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  //fill buffer with data
  fill_buffer();
  //fill eeprom w/ buffer
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WREN); //write enable
  digitalWrite(SLAVESELECT,HIGH);
  delay(10);
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(WRITE); //write instruction
  address=0;
  spi_transfer((char)(address>>8));   //send MSByte address first
  spi_transfer((char)(address));      //send LSByte address
  //write 128 bytes
  for (int I=0;I<128;I++)
  {
    spi_transfer(buffer[I]); //write data byte
  }
  digitalWrite(SLAVESELECT,HIGH); //release chip
  //wait for eeprom to finish writing
  delay(3000);
  Serial.print('h',BYTE);
  Serial.print('i',BYTE);
  Serial.print('\n',BYTE);//debug
  delay(1000);
}

byte read_eeprom(int EEPROM_address)
{
  //READ EEPROM
  int data;
  digitalWrite(SLAVESELECT,LOW);
  spi_transfer(READ); //transmit read opcode
  spi_transfer((char)(EEPROM_address>>8));   //send MSByte address first
  spi_transfer((char)(EEPROM_address));      //send LSByte address
  data = spi_transfer(0xFF); //get data byte
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
  return data;
}

void loop()
{
  eeprom_output_data = read_eeprom(address);
  Serial.print(eeprom_output_data,DEC);
  Serial.print('\n',BYTE);
  address++;
  if (address == 128)
    address = 0;
  delay(500); //pause for readability
}
```

*code and tutorial by Heather Dewey-Hagborg, photos by Thomas Dexter*

## Tutorial.SPIDigitalPot History

Hide minor edits - Show changes to markup

September 06, 2006, at 04:02 PM by Heather Dewey-Hagborg -
Changed lines 137-138 from:

VIDEO ? LEDs

to:

LED video

Restore
September 06, 2006, at 03:55 PM by Heather Dewey-Hagborg -
Changed lines 204-206 from:

@]

to:

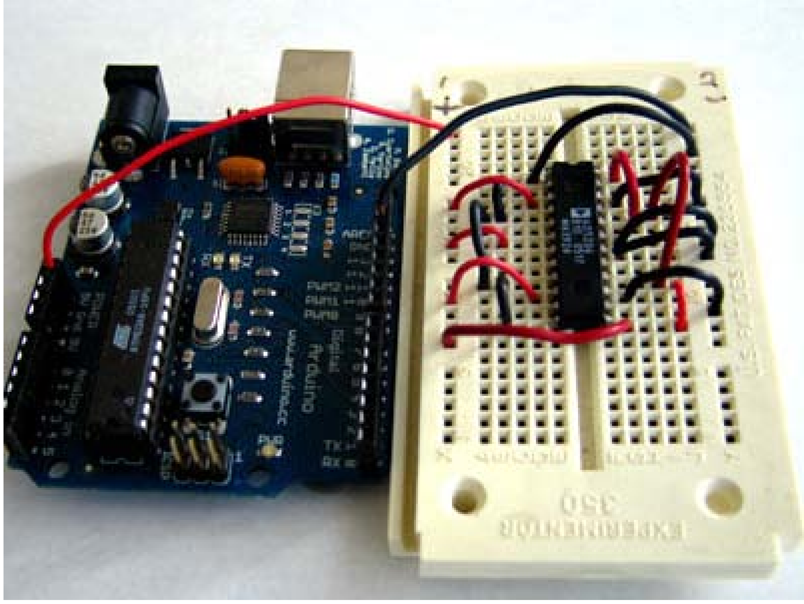
@]

*code, tutorial and photos by Heather Dewey-Hagborg*

Restore
September 06, 2006, at 03:49 PM by Heather Dewey-Hagborg -
Changed lines 28-29 from:

PICTURE power

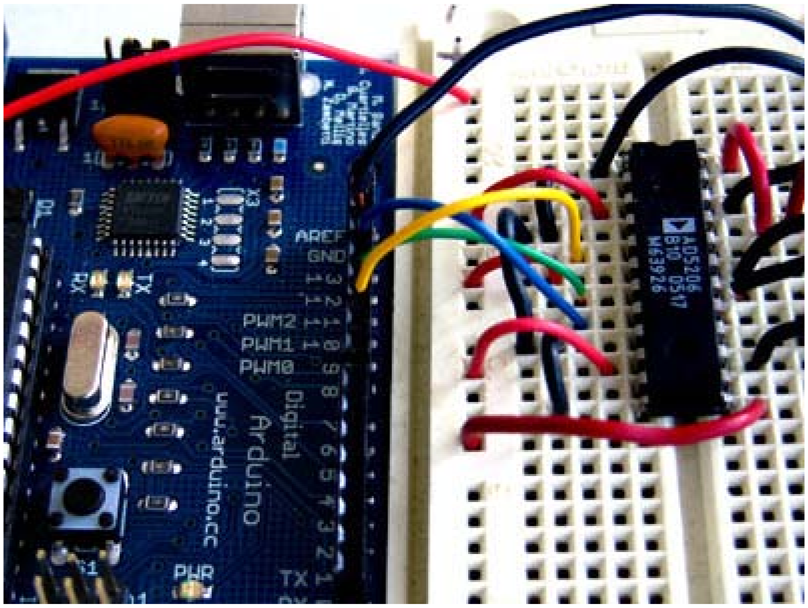
to:



Changed lines 32-33 from:

PICTURE datacom

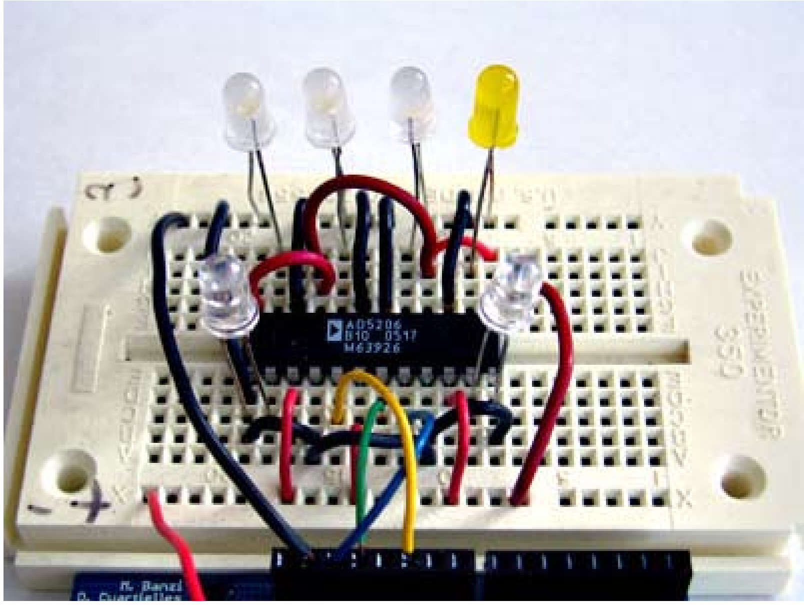
to:



Changed lines 36-37 from:

PICTURE leds

to:



Restore

September 06, 2006, at 03:38 PM by Heather Dewey-Hagborg -
Added lines 5-11:

Materials Needed:

- AD5206 Digital Potentiometer
- Arduino Microcontroller Module
- 6 Light Emitting Diodes (LEDs)
- Hookup Wire

Restore

September 06, 2006, at 10:03 AM by Heather Dewey-Hagborg -

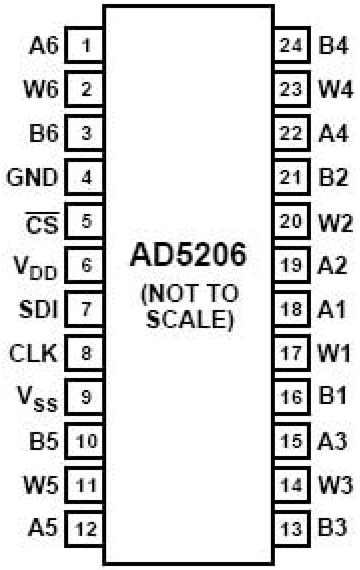
Restore

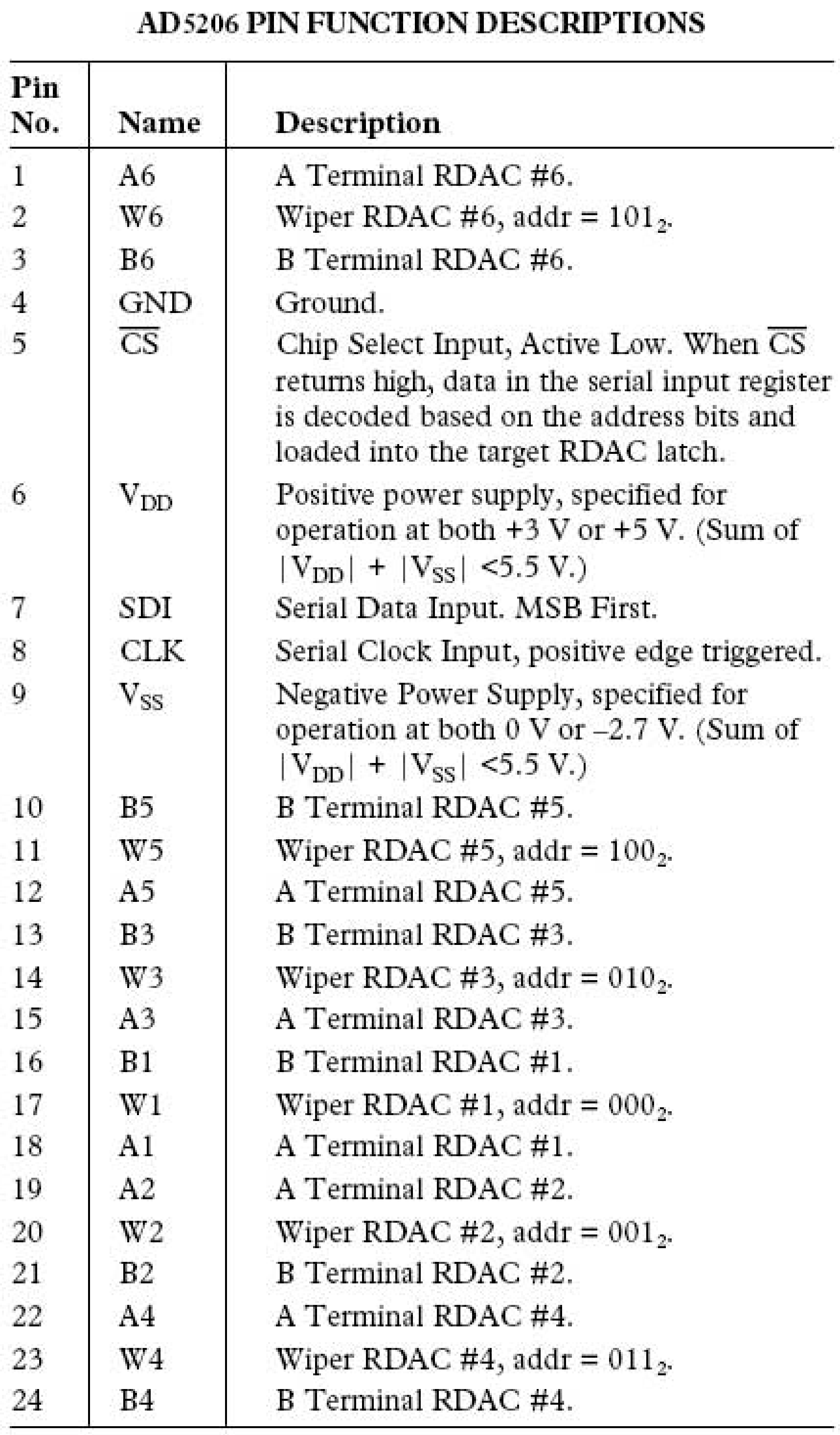
September 06, 2006, at 09:22 AM by Heather Dewey-Hagborg -
Changed lines 7-10 from:

PICTURE pins

PICTURE pin functions

to:


A6
1
W6
2
B6
3
GND
4
CS
5
VDD
6
SDI
7
CLK
8
VSS
9
B5
10
W5
11
A5
12
AD5206
(NOT TO SCALE)
24
B4
23
W4
22
A4
21
B2
20
W2
19
A2
18
A1
17
W1
16
B1
15
A3
14
W3
13
B3



AD5206 PIN FUNCTION DESCRIPTIONS

| Pin No. | Name | Description |
|---|---|---|
| 1 | A6 | A Terminal RDAC #6. |
| 2 | W6 | Wiper RDAC #6, addr = $101_2$. |
| 3 | B6 | B Terminal RDAC #6. |
| 4 | GND | Ground. |
| 5 | $\overline{CS}$ | Chip Select Input, Active Low. When $\overline{CS}$ returns high, data in the serial input register is decoded based on the address bits and loaded into the target RDAC latch. |
| 6 | $V_{DD}$ | Positive power supply, specified for operation at both +3 V or +5 V. (Sum of $\lvert V_{DD}\rvert + \lvert V_{SS}\rvert < 5.5$ V.) |
| 7 | SDI | Serial Data Input. MSB First. |
| 8 | CLK | Serial Clock Input, positive edge triggered. |
| 9 | $V_{SS}$ | Negative Power Supply, specified for operation at both 0 V or –2.7 V. (Sum of $\lvert V_{DD}\rvert + \lvert V_{SS}\rvert < 5.5$ V.) |
| 10 | B5 | B Terminal RDAC #5. |
| 11 | W5 | Wiper RDAC #5, addr = $100_2$. |
| 12 | A5 | A Terminal RDAC #5. |
| 13 | B3 | B Terminal RDAC #3. |
| 14 | W3 | Wiper RDAC #3, addr = $010_2$. |
| 15 | A3 | A Terminal RDAC #3. |
| 16 | B1 | B Terminal RDAC #1. |
| 17 | W1 | Wiper RDAC #1, addr = $000_2$. |
| 18 | A1 | A Terminal RDAC #1. |
| 19 | A2 | A Terminal RDAC #2. |
| 20 | W2 | Wiper RDAC #2, addr = $001_2$. |
| 21 | B2 | B Terminal RDAC #2. |
| 22 | A4 | A Terminal RDAC #4. |
| 23 | W4 | Wiper RDAC #4, addr = $011_2$. |
| 24 | B4 | B Terminal RDAC #4. |

Restore
September 05, 2006, at 03:11 PM by Heather Dewey-Hagborg -
Added lines 130-131:

VIDEO ? LEDs

Restore
September 05, 2006, at 03:07 PM by Heather Dewey-Hagborg -
Restore
September 05, 2006, at 03:03 PM by Heather Dewey-Hagborg -
Changed lines 117-118 from:

The write_pot function allows us to control the individual potentiometers. First we shift the potentiometer address 8 bits to the left to put it in the most significant bit position of the 11 bit data byte. This makes room to add the resistance value which occupies the least significant eight bits of the data byte. When we sum the two together we get our 11 bit opcode to transmit:

to:

The write_pot function allows us to control the individual potentiometers. We set the SLAVESELECT line low to enable the device. Then we transfer the address byte followed by the resistance value byte. Finally, we set the SLAVSELECT line high again to release the chip and signal the end of our data transfer.

Deleted lines 121-128:

```
  int opcode=0;
  address<<=8; //shift pot address 8 left, ie. 101 = 10100000000
  opcode = address+value; //10111111111
```

@]

We set the SLAVESELECT line low to enable the device. Then we transfer the 11 bit opcode in two bytes sending the eight most significant bits first and sending the three least significant bits last. We set the SLAVSELECT line high again to release the chip and signal the end of our data transfer.

[@

Changed lines 124-125 from:

```
  spi_transfer((char)(opcode>>8));   //send MSByte address first a0a1a2d0d1d2d3d4
  spi_transfer((char)(opcode));      //send LSByte address, d5d6d700000
```

to:

```
  spi_transfer(address);
  spi_transfer(value);
```

Deleted lines 173-175:

```
  int opcode=0;
  address<<=8; //shift pot address 8 left, ie. 101 = 10100000000
  opcode = address+value; //10111111111
```

Changed lines 176-177 from:

```
  spi_transfer((char)(opcode>>8));   //send MSByte address first a0a1a2d0d1d2d3d4
  spi_transfer((char)(opcode));      //send LSByte address, d5d6d700000
```

to:

```
  spi_transfer(address);
  spi_transfer(value);
```

Restore
September 05, 2006, at 02:42 PM by Heather Dewey-Hagborg -
Changed lines 11-12 from:

The AD5206 is a 6 channel digital potentiometer. This means it has six variable resistors built in for individual electronic control. There are three pins on the chip for each of the six internal variable resistors, and they can be interfaced with just as you would use a mechanical potentiometer. The individual variable resistor pins are labeled Ax, Bx and Wx, ie. A1, B1 and W1.

to:

The AD5206 is a 6 channel digital potentiometer. This means it has six variable resistors (potentiometers) built in for individual electronic control. There are three pins on the chip for each of the six internal variable resistors, and they can be interfaced with just as you would use a mechanical potentiometer. The individual variable resistor pins are labeled Ax, Bx and Wx, ie. A1, B1 and W1.

Changed lines 85-86 from:

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here?. It then returns any data that has been shifted in to the data register by the EEPROM:

to:

In our main loop we iterate through each resistance value (0-255) for each potentiometer address (0-5). We pause for 10 milliseconds each iteration to make the steps visible. This causes the LEDs to sequentially flash on brightly and then fade out slowly:

Changed line 88 from:

char spi_transfer(volatile char data)

to:

void loop()

Changed lines 90-94 from:

```
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
```

to:

```
     write_pot(pot,resistance);
     delay(10);
     resistance++;
     if (resistance==255)
     {
       pot++;
     }
     if (pot==6)
     {
       pot=0;
     }
```

Added lines 102-205:

@]

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

The write_pot function allows us to control the individual potentiometers. First we shift the potentiometer address 8 bits to the left to put it in the most significant bit position of the 11 bit data byte. This makes room to add the resistance value which occupies the least significant eight bits of the data byte. When we sum the two together we get our 11 bit opcode to transmit:

```
byte write_pot(int address, int value)
{
  int opcode=0;
  address<<=8; //shift pot address 8 left, ie. 101 = 10100000000
  opcode = address+value; //10111111111
```

We set the SLAVESELECT line low to enable the device. Then we transfer the 11 bit opcode in two bytes sending the eight most significant bits first and sending the three least significant bits last. We set the SLAVSELECT line high again to release the chip and signal the end of our data transfer.

```
  digitalWrite(SLAVESELECT,LOW);
  //2 byte opcode
  spi_transfer((char)(opcode>>8));   //send MSByte address first a0a1a2d0d1d2d3d4
  spi_transfer((char)(opcode));      //send LSByte address, d5d6d700000
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
}
```

For easy copy and pasting the full program text of this tutorial is below:

[@

1. define DATAOUT 11//MOSI
2. define DATAIN 12//MISO - not used, but part of builtin SPI
3. define SPICLOCK 13//sck
4. define SLAVESELECT 10//ss

byte pot=0; byte resistance=0;

char spi_transfer(volatile char data) {

```
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
```

}

void setup() {

```
  byte i;
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  for (i=0;i<6;i++)
  {
    write_pot(i,255);
  }
```

}

byte write_pot(int address, int value) {

```
  int opcode=0;
  address<<=8; //shift pot address 8 left, ie. 101 = 10100000000
  opcode = address+value; //10111111111
  digitalWrite(SLAVESELECT,LOW);
  //2 byte opcode
  spi_transfer((char)(opcode>>8));   //send MSByte address first a0a1a2d0d1d2d3d4
  spi_transfer((char)(opcode));      //send LSByte address, d5d6d700000
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
```

}

void loop() {

```
     write_pot(pot,resistance);
     delay(10);
     resistance++;
     if (resistance==255)
     {
       pot++;
     }
     if (pot==6)
     {
       pot=0;
     }

}
```

Restore
September 05, 2006, at 02:25 PM by Heather Dewey-Hagborg -
Added lines 53-96:

First we set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise:

```
void setup()
{
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
```

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variable to clear out any spurious data from past runs:

```
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
```

We conclude the setup function by setting all the potentiometers to full on resistance states thereby turning the LEDs off:

```
  for (i=0;i<6;i++)
  {
    write_pot(i,255);
  }
}
```

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here?. It then returns any data that has been shifted in to the data register by the EEPROM:

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

Restore
September 05, 2006, at 02:14 PM by Heather Dewey-Hagborg -
Changed lines 15-16 from:

The AD5206 is digitally controlled using SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) and are shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs much faster than the Arduino, so we don't need to worry about pre-scaling to slow down the transmission.

to:

The AD5206 is digitally controlled using SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) with the three most significant bits (11-9) defining the address of which potentiometer to adjust and the eight least significant bits (8-1) defining what value to set that potentiometer to from 0-255. Data is shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs much faster than the Arduino, so we don't need to worry about pre-scaling to slow down the transmission.

Added lines 26-52:

Finally, connect an LED between each Wiper pin (AD5206 pins 2, 11, 14, 17, 20 and 23) and ground so that the long pin of the LED connects to the wiper and the short pin, or flat side of the LED connects to ground.

PICTURE leds

## Program the Arduino

Now we will write the code to enable SPI control of the AD5206. This program will sequentially pulse each LED on and then fade it out gradually. This is accomplished in the main loop of the program by individually changing the resistance of each potentiometer from full off to full on over its full range of 255 steps.

We will walk through the code in small sections.

We define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Although we are not reading any data back out of the AD5206 in this program, pin 12 is attached to the builtin SPI so it is best not to use it for other programming functions to avoid any possible errors:

```
#define DATAOUT 11//MOSI
#define DATAIN 12//MISO - not used, but part of builtin SPI
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss
```

Next we allocate variables to store resistance values and address values for the potentiometers:

```
byte pot=0;
byte resistance=0;
```

Restore
September 05, 2006, at 01:59 PM by Heather Dewey-Hagborg -
Changed lines 11-12 from:

The AD5206 is a 6 channel digital potentiometer. This means it has six variable resistors built in for individual electronic control. There are three pins on the chip for each of the six internal variable resistors, and they can be interfaced with just as you would use a mechanical potentiometer.

to:

The AD5206 is a 6 channel digital potentiometer. This means it has six variable resistors built in for individual electronic control. There are three pins on the chip for each of the six internal variable resistors, and they can be interfaced with just as you would use a mechanical potentiometer. The individual variable resistor pins are labeled Ax, Bx and Wx, ie. A1, B1 and W1.

Changed lines 15-25 from:

The AD5206 is digitally controlled using standard SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) and are shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs much faster than the Arduino, so we don't need to worry about pre-scaling to slow down the transmission.

to:

The AD5206 is digitally controlled using SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) and are shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs much faster than the Arduino, so we don't need to worry about pre-scaling to slow down the transmission.

## Prepare the Breadboard

Insert the AD5206 chip into the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect AD5206 pins 3, 6, 10, 13, 16, 21 and 24 to 5v and pins 1, 4, 9, 12, 15, 18, 19, and 22 to ground. We are connecting all the A pins to ground and all of the B pins to 5v to create 6 voltage dividers.

PICTURE power

Connect AD5206 pin 5 to Arduino pin 10 (Slave Select - SS), AD5206 pin 7 to Arduino pin 11 (Master Out Slave In - MOSI), and AD5206 pin 8 to Arduino pin 13 (Serial Clock - SCK).

PICTURE datacom

Restore
September 05, 2006, at 01:34 PM by Heather Dewey-Hagborg -
Changed line 15 from:

The AD5206 is digitally controlled using standard SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) and are shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs up to speeds of 100Mhz.

to:

The AD5206 is digitally controlled using standard SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) and are shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs much faster than the Arduino, so we don't need to worry about pre-scaling to slow down the transmission.

Restore
September 05, 2006, at 01:30 PM by Heather Dewey-Hagborg -
Changed line 15 from:

The AD5206is controlled using standard SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 8 bit operational codes (opcodes) and are shifted in on the rising edge of the data clock.

to:

The AD5206 is digitally controlled using standard SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) and are shifted in Most Significant Bit (MSB) first on the rising edge of the data clock. The data clock is idle when low, and the interface runs up to speeds of 100Mhz.

Restore
September 05, 2006, at 01:23 PM by Heather Dewey-Hagborg -
Added lines 1-15:

# Controlling a Digital Potentiometer Using SPI

In this tutorial you will learn how to control the AD5206 digital potentiometer using Serial Peripheral Interface (SPI). For an explanation of SPI see the SPI EEPROM tutorial. Digital potentiometers are useful when you need to vary the resistance in a ciruit electronically rather than by hand. Example applications include LED dimming, audio signal conditioning and tone generation. In this example we will use a six channel digital potentiometer to control the brightness of six LEDs. The steps we will cover for implementing SPI communication can be modified for use with most other SPI devices.

## Introduction to the AD5206 Digital Potentiometer

PICTURE pins

PICTURE pin functions

The AD5206 is a 6 channel digital potentiometer. This means it has six variable resistors built in for individual electronic control. There are three pins on the chip for each of the six internal variable resistors, and they can be interfaced with just as you would use a mechanical potentiometer.

For example, in this tutorial we will be using each variable resistor as a voltage divider by pulling one side pin (pin B) high, pulling another side pin (pin A) low and taking the variable voltage output of the center pin (Wiper).

The AD5206is controlled using standard SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 8 bit operational codes (opcodes) and are shifted in on the rising edge of the data clock.

Restore

**Learning**   Examples | Foundations | Hacking | Links

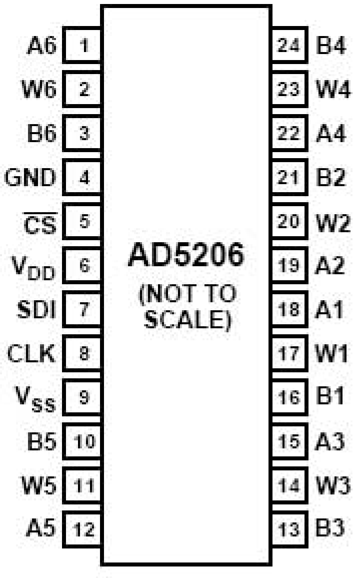
# Controlling a Digital Potentiometer Using SPI

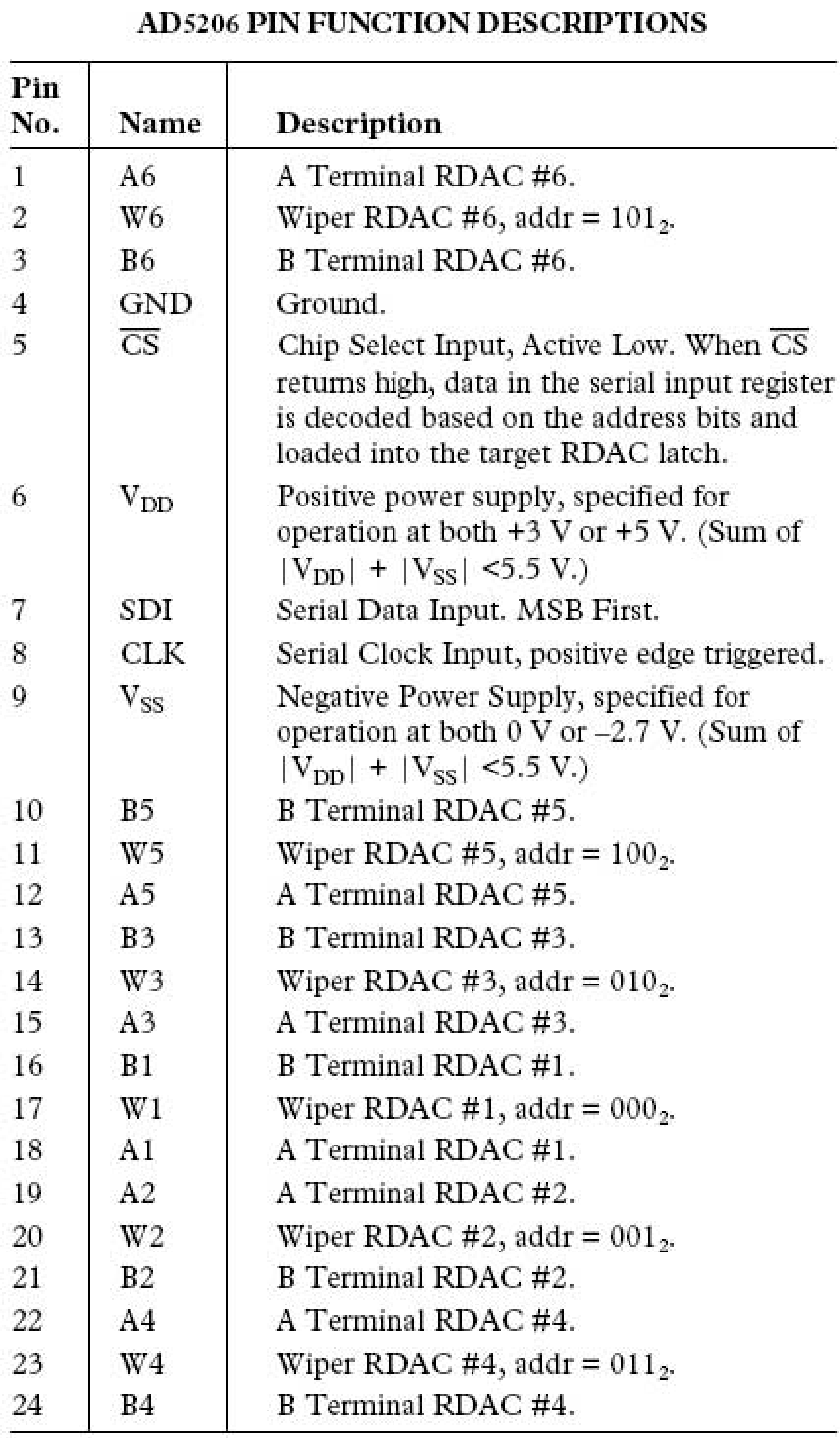
In this tutorial you will learn how to control the AD5206 digital potentiometer using Serial Peripheral Interface (SPI). For an explanation of SPI see the SPI EEPROM tutorial. Digital potentiometers are useful when you need to vary the resistance in a ciruit electronically rather than by hand. Example applications include LED dimming, audio signal conditioning and tone generation. In this example we will use a six channel digital potentiometer to control the brightness of six LEDs. The steps we will cover for implementing SPI communication can be modified for use with most other SPI devices.

Materials Needed:

- AD5206 Digital Potentiometer
- Arduino Microcontroller Module
- 6 Light Emitting Diodes (LEDs)
- Hookup Wire

## Introduction to the AD5206 Digital Potentiometer





**AD5206 PIN FUNCTION DESCRIPTIONS**

| Pin No. | Name | Description |
|---|---|---|
| 1 | A6 | A Terminal RDAC #6. |
| 2 | W6 | Wiper RDAC #6, addr = $101_2$. |
| 3 | B6 | B Terminal RDAC #6. |
| 4 | GND | Ground. |
| 5 | $\overline{CS}$ | Chip Select Input, Active Low. When $\overline{CS}$ returns high, data in the serial input register is decoded based on the address bits and loaded into the target RDAC latch. |
| 6 | $V_{DD}$ | Positive power supply, specified for operation at both +3 V or +5 V. (Sum of $\lvert V_{DD}\rvert + \lvert V_{SS}\rvert < 5.5$ V.) |
| 7 | SDI | Serial Data Input. MSB First. |
| 8 | CLK | Serial Clock Input, positive edge triggered. |
| 9 | $V_{SS}$ | Negative Power Supply, specified for operation at both 0 V or –2.7 V. (Sum of $\lvert V_{DD}\rvert + \lvert V_{SS}\rvert < 5.5$ V.) |
| 10 | B5 | B Terminal RDAC #5. |
| 11 | W5 | Wiper RDAC #5, addr = $100_2$. |
| 12 | A5 | A Terminal RDAC #5. |
| 13 | B3 | B Terminal RDAC #3. |
| 14 | W3 | Wiper RDAC #3, addr = $010_2$. |
| 15 | A3 | A Terminal RDAC #3. |
| 16 | B1 | B Terminal RDAC #1. |
| 17 | W1 | Wiper RDAC #1, addr = $000_2$. |
| 18 | A1 | A Terminal RDAC #1. |
| 19 | A2 | A Terminal RDAC #2. |
| 20 | W2 | Wiper RDAC #2, addr = $001_2$. |
| 21 | B2 | B Terminal RDAC #2. |
| 22 | A4 | A Terminal RDAC #4. |
| 23 | W4 | Wiper RDAC #4, addr = $011_2$. |
| 24 | B4 | B Terminal RDAC #4. |

The AD5206 is a 6 channel digital potentiometer. This means it has six variable resistors (potentiometers) built in for individual electronic control. There are three pins on the chip for each of the six internal variable resistors, and they can be interfaced with just as you would use a mechanical potentiometer. The individual variable resistor pins are labeled Ax, Bx and Wx, ie. A1, B1 and W1.

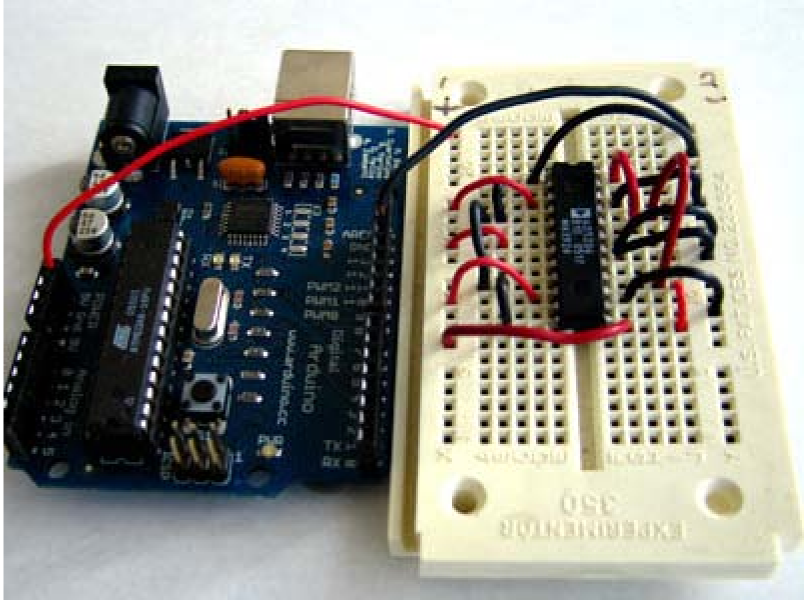
For example, in this tutorial we will be using each variable resistor as a voltage divider by pulling one side pin (pin B) high, pulling another side pin (pin A) low and taking the variable voltage output of the center pin (Wiper).

The AD5206 is digitally controlled using SPI. The device is enabled by pulling the Chip Select (CS) pin low. Instructions are sent as 11 bit operational codes (opcodes) with the three most significant bits (11-9) defining the address of which potentiometer to adjust and the eight least significant bits (8-1) defining what value to set that potentiometer to from 0-255. Data is shifted in Most Significant Bit (MSB) first on the rising edge of the data clock.

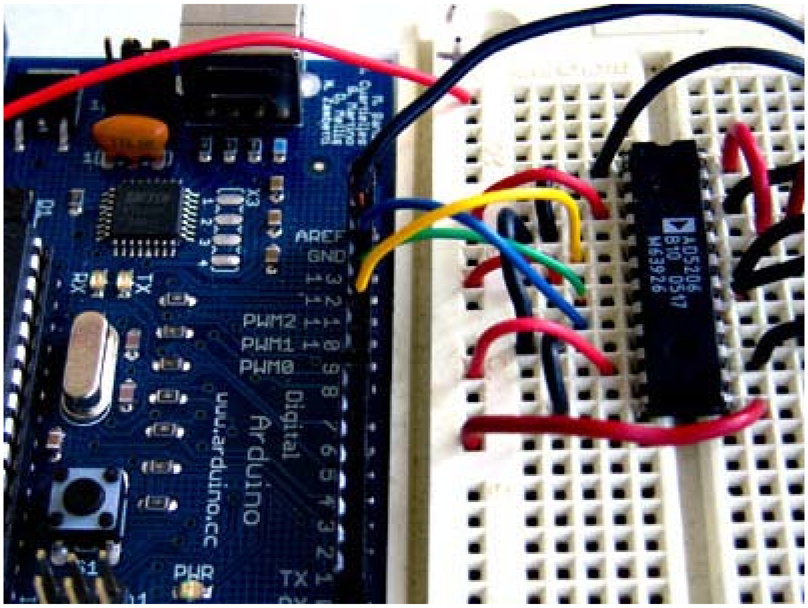
The data clock is idle when low, and the interface runs much faster than the Arduino, so we don't need to worry about pre-scaling to slow down the transmission.

## Prepare the Breadboard

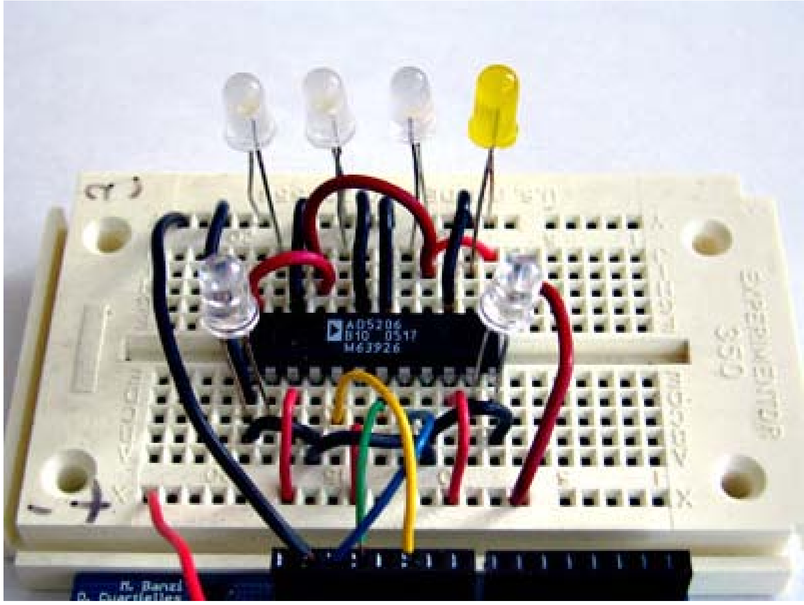
Insert the AD5206 chip into the breadboard. Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller. Connect AD5206 pins 3, 6, 10, 13, 16, 21 and 24 to 5v and pins 1, 4, 9, 12, 15, 18, 19, and 22 to ground. We are connecting all the A pins to ground and all of the B pins to 5v to create 6 voltage dividers.



Connect AD5206 pin 5 to Arduino pin 10 (Slave Select - SS), AD5206 pin 7 to Arduino pin 11 (Master Out Slave In - MOSI), and AD5206 pin 8 to Arduino pin 13 (Serial Clock - SCK).



Finally, connect an LED between each Wiper pin (AD5206 pins 2, 11, 14, 17, 20 and 23) and ground so that the long pin of the LED connects to the wiper and the short pin, or flat side of the LED connects to ground.



## Program the Arduino

Now we will write the code to enable SPI control of the AD5206. This program will sequentially pulse each LED on and then fade it out gradually. This is accomplished in the main loop of the program by individually changing the resistance of each potentiometer from full off to full on over its full range of 255 steps.

We will walk through the code in small sections.

We define the pins we will be using for our SPI connection, DATAOUT, DATAIN, SPICLOCK and SLAVESELECT. Although we are not reading any data back out of the AD5206 in this program, pin 12 is attached to the builtin SPI so it is best not to use it for other programming functions to avoid any possible errors:

```
#define DATAOUT 11//MOSI
#define DATAIN 12//MISO - not used, but part of builtin SPI
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss
```

Next we allocate variables to store resistance values and address values for the potentiometers:

```
byte pot=0;
byte resistance=0;
```

First we set our input and output pin modes and set the SLAVESELECT line high to start. This deselects the device and avoids any false transmission messages due to line noise:

```
void setup()
{
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN,  INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
```

Now we set the SPI Control register (SPCR) to the binary value 01010000. In the control register each bit sets a different functionality. The eighth bit disables the SPI interrupt, the seventh bit enables the SPI, the sixth bit chooses transmission with the most significant bit going first, the fifth bit puts the Arduino in Master mode, the fourth bit sets the data clock idle when it is low, the third bit sets the SPI to sample data on the rising edge of the data clock, and the second and first bits set the speed of the SPI to system speed / 4 (the fastest). After setting our control register up we read the SPI status register (SPSR) and data register (SPDR) in to the junk clr variable to clear out any spurious data from past runs:

```
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
```

We conclude the setup function by setting all the potentiometers to full on resistance states thereby turning the LEDs off:

```
  for (i=0;i<6;i++)
  {
    write_pot(i,255);
  }
}
```

In our main loop we iterate through each resistance value (0-255) for each potentiometer address (0-5). We pause for 10 milliseconds each iteration to make the steps visible. This causes the LEDs to sequentially flash on brightly and then fade out slowly:

```
void loop()
{
     write_pot(pot,resistance);
     delay(10);
     resistance++;
     if (resistance==255)
     {
       pot++;
     }
     if (pot==6)
     {
       pot=0;
     }
}
```

The spi_transfer function loads the output data into the data transmission register, thus starting the SPI transmission. It polls a bit to the SPI Status register (SPSR) to detect when the transmission is complete using a bit mask, SPIF. An explanation of bit masks can be found here. It then returns any data that has been shifted in to the data register by the EEPROM:

```
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}
```

The write_pot function allows us to control the individual potentiometers. We set the SLAVESELECT line low to enable the device. Then we transfer the address byte followed by the resistance value byte. Finally, we set the SLAVSELECT line high again to release the chip and signal the end of our data transfer.

```
byte write_pot(int address, int value)
{
  digitalWrite(SLAVESELECT,LOW);
  //2 byte opcode
  spi_transfer(address);
  spi_transfer(value);
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
}
```

LED video

For easy copy and pasting the full program text of this tutorial is below:

```
#define DATAOUT 11//MOSI
```

```
#define DATAIN 12//MISO - not used, but part of builtin SPI
#define SPICLOCK  13//sck
#define SLAVESELECT 10//ss

byte pot=0;
byte resistance=0;

char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // return the received byte
}

void setup()
{
  byte i;
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);
  for (i=0;i<6;i++)
  {
    write_pot(i,255);
  }
}

byte write_pot(int address, int value)
{
  digitalWrite(SLAVESELECT,LOW);
  //2 byte opcode
  spi_transfer(address);
  spi_transfer(value);
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
}

void loop()
{
     write_pot(pot,resistance);
     delay(10);
     resistance++;
     if (resistance==255)
     {
       pot++;
     }
     if (pot==6)
     {
       pot=0;
     }
}
```

*code, tutorial and photos by Heather Dewey-Hagborg*

**Learning** Examples | Foundations | Hacking | Links

Back to ShiftOut Tutorial

```
//************************************************************//
//  Name    : shiftOutCode, Hello World                                //
//  Author  : Carlyn Maw,Tom Igoe                          //
//  Date    : 25 Oct, 2006                                     //
//  Version : 1.0                                              //
//  Notes   : Code for using a 74HC595 Shift Register           //
//          : to count from 0 to 255                           //
//****************************************************************

//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
////Pin connected to DS of 74HC595
int dataPin = 11;



void setup() {
  //set pins to output because they are addressed in the main loop
  pinMode(latchPin, OUTPUT);

}

void loop() {
  //count up routine
  for (int j = 0; j < 256; j++) {
    //ground latchPin and hold low for as long as you are transmitting
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, j);
    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(latchPin, 1);
    delay(1000);
  }
}

void shiftOut(int myDataPin, int myClockPin, byte myDataOut) {
  // This shifts 8 bits out MSB first,
  //on the rising edge of the clock,
  //clock idles low

  //internal function setup
  int i=0;
  int pinState;
  pinMode(myClockPin, OUTPUT);
  pinMode(myDataPin, OUTPUT);

  //clear everything out just in case to
  //prepare shift register for bit shifting
  digitalWrite(myDataPin, 0);
  digitalWrite(myClockPin, 0);

  //for each bit in the byte myDataOut…
  //NOTICE THAT WE ARE COUNTING DOWN in our for loop
  //This means that %00000001 or "1" will go through such
  //that it will be pin Q0 that lights.
  for (i=7; i>=0; i--)  {
    digitalWrite(myClockPin, 0);

    //if the value passed to myDataOut and a bitmask result
    // true then... so if we are at i=6 and our value is
    // %11010100 it would the code compares it to %01000000
    // and proceeds to set pinState to 1.
    if ( myDataOut & (1<<i) ) {
      pinState= 1;
    }
    else {
      pinState= 0;
    }

    //Sets the pin to HIGH or LOW depending on pinState
    digitalWrite(myDataPin, pinState);
    //register shifts bits on upstroke of clock pin
    digitalWrite(myClockPin, 1);
    //zero the data pin after shift to prevent bleed through
    digitalWrite(myDataPin, 0);
  }

  //stop shifting
  digitalWrite(myClockPin, 0);
}
```

# Arduino

search

Buy | Download | Getting Started | Learning | Reference | Hardware | FAQ    Blog » | Forum » | Playground »

**Learning**   Examples | Foundations | Hacking | Links

Back to ShiftOut Tutorial

```
//**************************************************************//
//  Name    : shiftOutCode, One By One                         //
//  Author  : Carlyn Maw, Tom Igoe                          //
//  Date    : 25 Oct, 2006                                     //
//  Version : 1.0                                              //
//  Notes   : Code for using a 74HC595 Shift Register           //
//          : to count from 0 to 255                           //
//****************************************************************

//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
////Pin connected to DS of 74HC595
int dataPin = 11;

//holder for infromation you're going to pass to shifting function
byte data = 0;



void setup() {
  //set pins to output because they are addressed in the main loop
  pinMode(latchPin, OUTPUT);

}

void loop() {

  //function that blinks all the LEDs
  //gets passed the number of blinks and the pause time
  blinkAll(1,500);

 // light each pin one by one using a function A
  for (int j = 0; j < 8; j++) {
    lightShiftPinA(j);
    delay(1000);
  }

  blinkAll(2,500);

 // light each pin one by one using a function A
  for (int j = 0; j < 8; j++) {
    lightShiftPinB(j);
    delay(1000);
  }

}

//This function uses bitwise math to move the pins up
void lightShiftPinA(int p) {
  //defines a local variable
  int pin;

  //this is line uses a bitwise operator
  //shifting a bit left using << is the same
  //as multiplying the decimal number by two.
  pin = 1<< p;

  //ground latchPin and hold low for as long as you are transmitting
  digitalWrite(latchPin, 0);
  //move 'em out
  shiftOut(dataPin, clockPin, pin);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
  digitalWrite(latchPin, 1);

}

//This function uses that fact that each bit in a byte
//is 2 times greater than the one before it to
//shift the bits higher
void lightShiftPinB(int p) {
  //defines a local variable
  int pin;

  //start with the pin = 1 so that if 0 is passed to this
  //function pin 0 will light.
  pin = 1;

  for (int x = 0; x < p; x++) {
    pin = pin * 2;
  }

  //ground latchPin and hold low for as long as you are transmitting
  digitalWrite(latchPin, 0);
  //move 'em out
  shiftOut(dataPin, clockPin, pin);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
  digitalWrite(latchPin, 1);

}


// the heart of the program
void shiftOut(int myDataPin, int myClockPin, byte myDataOut) {
  // This shifts 8 bits out MSB first,
  //on the rising edge of the clock,
  //clock idles low

  //internal function setup
  int i=0;
  int pinState;
  pinMode(myClockPin, OUTPUT);
  pinMode(myDataPin, OUTPUT);

  //clear everything out just in case to
  //prepare shift register for bit shifting
  digitalWrite(myDataPin, 0);
  digitalWrite(myClockPin, 0);

  //for each bit in the byte myDataOut…
  //NOTICE THAT WE ARE COUNTING DOWN in our for loop
  //This means that %00000001 or "1" will go through such
  //that it will be pin Q0 that lights.
  for (i=7; i>=0; i--)  {
    digitalWrite(myClockPin, 0);

    //if the value passed to myDataOut and a bitmask result
    // true then... so if we are at i=6 and our value is
    // %11010100 it would the code compares it to %01000000
    // and proceeds to set pinState to 1.
    if ( myDataOut & (1<<i) ) {
      pinState= 1;
    }
    else {
      pinState= 0;
    }

    //Sets the pin to HIGH or LOW depending on pinState
    digitalWrite(myDataPin, pinState);
    //register shifts bits on upstroke of clock pin
    digitalWrite(myClockPin, 1);
    //zero the data pin after shift to prevent bleed through
    digitalWrite(myDataPin, 0);
  }

  //stop shifting
  digitalWrite(myClockPin, 0);
}


//blinks the whole register based on the number of times you want to
//blink "n" and the pause between them "d"
//starts with a moment of darkness to make sure the first blink
//has its full visual effect.
void blinkAll(int n, int d) {
  digitalWrite(latchPin, 0);
  shiftOut(dataPin, clockPin, 0);
  digitalWrite(latchPin, 1);
  delay(200);
  for (int x = 0; x < n; x++) {
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 255);
    digitalWrite(latchPin, 1);
    delay(d);
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 0);
    digitalWrite(latchPin, 1);
    delay(d);
  }
}
```

Back to ShiftOut Tutorial

```
//*************************************************************//
//  Name    : shiftOutCode, Predefined Array Style              //
//  Author  : Carlyn Maw, Tom Igoe                           //
//  Date    : 25 Oct, 2006                                      //
//  Version : 1.0                                              //
//  Notes   : Code for using a 74HC595 Shift Register           //
//          : to count from 0 to 255                            //
//***************************************************************

//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
////Pin connected to DS of 74HC595
int dataPin = 11;

//holders for infromation you're going to pass to shifting function
byte data;
byte dataArray[10];

void setup() {
  //set pins to output because they are addressed in the main loop
  pinMode(latchPin, OUTPUT);
  Serial.begin(9600);

  //Arduino doesn't seem to have a way to write binary straight into the code
  //so these values are in HEX.  Decimal would have been fine, too.
  dataArray[0] = 0xAA; //10101010
  dataArray[1] = 0x55; //01010101
  dataArray[2] = 0x81; //10000001
  dataArray[3] = 0xC3; //11000011
  dataArray[4] = 0xE7; //11100111
  dataArray[5] = 0xFF; //11111111
  dataArray[6] = 0x7E; //01111110
  dataArray[7] = 0x3C; //00111100
  dataArray[8] = 0x18; //00011000
  dataArray[9] = 0x00; //00000000

  //function that blinks all the LEDs
  //gets passed the number of blinks and the pause time
  blinkAll(2,500);
}

void loop() {


  for (int j = 0; j < 10; j++) {
    //load the light sequence you want from array
    data = dataArray[j];
    //ground latchPin and hold low for as long as you are transmitting
    digitalWrite(latchPin, 0);
    //move 'em out
    shiftOut(dataPin, clockPin, data);
    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(latchPin, 1);
    delay(1000);
  }
}



// the heart of the program
void shiftOut(int myDataPin, int myClockPin, byte myDataOut) {
  // This shifts 8 bits out MSB first,
  //on the rising edge of the clock,
  //clock idles low

  //internal function setup
  int i=0;
  int pinState;
  pinMode(myClockPin, OUTPUT);
  pinMode(myDataPin, OUTPUT);

  //clear everything out just in case to
  //prepare shift register for bit shifting
  digitalWrite(myDataPin, 0);
  digitalWrite(myClockPin, 0);

  //for each bit in the byte myDataOut…
  //NOTICE THAT WE ARE COUNTING DOWN in our for loop
  //This means that %00000001 or "1" will go through such
  //that it will be pin Q0 that lights.
  for (i=7; i>=0; i--)  {
    digitalWrite(myClockPin, 0);

    //if the value passed to myDataOut and a bitmask result
    // true then... so if we are at i=6 and our value is
    // %11010100 it would the code compares it to %01000000
    // and proceeds to set pinState to 1.
    if ( myDataOut & (1<<i) ) {
      pinState= 1;
    }
    else {
      pinState= 0;
    }

    //Sets the pin to HIGH or LOW depending on pinState
    digitalWrite(myDataPin, pinState);
    //register shifts bits on upstroke of clock pin
    digitalWrite(myClockPin, 1);
    //zero the data pin after shift to prevent bleed through
    digitalWrite(myDataPin, 0);
  }

  //stop shifting
  digitalWrite(myClockPin, 0);
}


//blinks the whole register based on the number of times you want to
//blink "n" and the pause between them "d"
//starts with a moment of darkness to make sure the first blink
//has its full visual effect.
void blinkAll(int n, int d) {
  digitalWrite(latchPin, 0);
  shiftOut(dataPin, clockPin, 0);
  digitalWrite(latchPin, 1);
  delay(200);
  for (int x = 0; x < n; x++) {
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 255);
    digitalWrite(latchPin, 1);
    delay(d);
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 0);
    digitalWrite(latchPin, 1);
    delay(d);
  }
}
```

Edit Page | Page History | Printable View | All Recent Site Changes

Back to ShiftOut Tutorial

```
//**************************************************************//
//  Name    : shiftOutCode, Dual Binary Counters                 //
//  Author  : Carlyn Maw, Tom Igoe                               //
//  Date    : 25 Oct, 2006                                       //
//  Version : 1.0                                                //
//  Notes   : Code for using a 74HC595 Shift Register            //
//          : to count from 0 to 255                             //
//**************************************************************//

//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
////Pin connected to DS of 74HC595
int dataPin = 11;



void setup() {
  //Start Serial for debuging purposes
  Serial.begin(9600);
  //set pins to output because they are addressed in the main loop
  pinMode(latchPin, OUTPUT);

}

void loop() {
  //count up routine
  for (int j = 0; j < 256; j++) {
    //ground latchPin and hold low for as long as you are transmitting
    digitalWrite(latchPin, 0);
    //count up on GREEN LEDs
    shiftOut(dataPin, clockPin, j);
    //count down on RED LEDs
    shiftOut(dataPin, clockPin, 255-j);
    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(latchPin, 1);
    delay(1000);
  }
}

void shiftOut(int myDataPin, int myClockPin, byte myDataOut) {
  // This shifts 8 bits out MSB first,
  //on the rising edge of the clock,
  //clock idles low

..//internal function setup
  int i=0;
  int pinState;
  pinMode(myClockPin, OUTPUT);
  pinMode(myDataPin, OUTPUT);

. //clear everything out just in case to
. //prepare shift register for bit shifting
  digitalWrite(myDataPin, 0);
  digitalWrite(myClockPin, 0);

  //for each bit in the byte myDataOut…
  //NOTICE THAT WE ARE COUNTING DOWN in our for loop
  //This means that %00000001 or "1" will go through such
  //that it will be pin Q0 that lights.
  for (i=7; i>=0; i--)  {
    digitalWrite(myClockPin, 0);

    //if the value passed to myDataOut and a bitmask result
    // true then... so if we are at i=6 and our value is
    // %11010100 it would the code compares it to %01000000
    // and proceeds to set pinState to 1.
    if ( myDataOut & (1<<i) ) {
      pinState= 1;
    }
    else {
      pinState= 0;
    }

    //Sets the pin to HIGH or LOW depending on pinState
    digitalWrite(myDataPin, pinState);
    //register shifts bits on upstroke of clock pin
    digitalWrite(myClockPin, 1);
    //zero the data pin after shift to prevent bleed through
    digitalWrite(myDataPin, 0);
  }

  //stop shifting
  digitalWrite(myClockPin, 0);
}
```

Back to ShiftOut Tutorial

```
//**************************************************************//
//  Name    : shiftOutCode, Dual One By One                           //
//  Author  : Carlyn Maw, Tom Igoe                               //
//  Date    : 25 Oct, 2006                                       //
//  Version : 1.0                                                //
//  Notes   : Code for using a 74HC595 Shift Register            //
//          : to count from 0 to 255                             //
//**************************************************************//

//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
////Pin connected to DS of 74HC595
int dataPin = 11;

//holder for infromation you're going to pass to shifting function
byte data = 0;



void setup() {
  //set pins to output because they are addressed in the main loop
  pinMode(latchPin, OUTPUT);

}

void loop() {

  //function that blinks all the LEDs
  //gets passed the number of blinks and the pause time
  blinkAll_2Bytes(1,500);

  // light each pin one by one using a function A
  for (int j = 0; j < 8; j++) {
    //ground latchPin and hold low for as long as you are transmitting
    digitalWrite(latchPin, 0);
    //red LEDs
    lightShiftPinA(7-j);
    //green LEDs
    lightShiftPinA(j);
    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(latchPin, 1);
    delay(1000);
  }

  // light each pin one by one using a function A
  for (int j = 0; j < 8; j++) {
    //ground latchPin and hold low for as long as you are transmitting
    digitalWrite(latchPin, 0);
    //red LEDs
    lightShiftPinB(j);
    //green LEDs
    lightShiftPinB(7-j);
    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(latchPin, 1);
    delay(1000);
  }

}

//This function uses bitwise math to move the pins up
void lightShiftPinA(int p) {
  //defines a local variable
  int pin;

  //this is line uses a bitwise operator
  //shifting a bit left using << is the same
  //as multiplying the decimal number by two.
  pin = 1<< p;

  //move 'em out
  shiftOut(dataPin, clockPin, pin);

}

//This function uses that fact that each bit in a byte
//is 2 times greater than the one before it to
//shift the bits higher
void lightShiftPinB(int p) {
  //defines a local variable
  int pin;

  //start with the pin = 1 so that if 0 is passed to this
  //function pin 0 will light.
  pin = 1;

  for (int x = 0; x < p; x++) {
    pin = pin * 2;
  }
  //move 'em out
  shiftOut(dataPin, clockPin, pin);
}


// the heart of the program
void shiftOut(int myDataPin, int myClockPin, byte myDataOut) {
  // This shifts 8 bits out MSB first,
  //on the rising edge of the clock,
  //clock idles low

  //internal function setup
  int i=0;
  int pinState;
  pinMode(myClockPin, OUTPUT);
  pinMode(myDataPin, OUTPUT);

  //clear everything out just in case to
  //prepare shift register for bit shifting
  digitalWrite(myDataPin, 0);
  digitalWrite(myClockPin, 0);

  //for each bit in the byte myDataOut…
  //NOTICE THAT WE ARE COUNTING DOWN in our for loop
  //This means that %00000001 or "1" will go through such
  //that it will be pin Q0 that lights.
  for (i=7; i>=0; i--)  {
    digitalWrite(myClockPin, 0);

    //if the value passed to myDataOut and a bitmask result
    // true then... so if we are at i=6 and our value is
    // %11010100 it would the code compares it to %01000000
    // and proceeds to set pinState to 1.
    if ( myDataOut & (1<<i) ) {
      pinState= 1;
    }
    else {
      pinState= 0;
    }

    //Sets the pin to HIGH or LOW depending on pinState
    digitalWrite(myDataPin, pinState);
    //register shifts bits on upstroke of clock pin
    digitalWrite(myClockPin, 1);
    //zero the data pin after shift to prevent bleed through
    digitalWrite(myDataPin, 0);
  }

  //stop shifting
  digitalWrite(myClockPin, 0);
}


//blinks both registers based on the number of times you want to
//blink "n" and the pause between them "d"
//starts with a moment of darkness to make sure the first blink
//has its full visual effect.
void blinkAll_2Bytes(int n, int d) {
  digitalWrite(latchPin, 0);
  shiftOut(dataPin, clockPin, 0);
  shiftOut(dataPin, clockPin, 0);
  digitalWrite(latchPin, 1);
  delay(200);
  for (int x = 0; x < n; x++) {
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 255);
    shiftOut(dataPin, clockPin, 255);
    digitalWrite(latchPin, 1);
    delay(d);
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 0);
    shiftOut(dataPin, clockPin, 0);
    digitalWrite(latchPin, 1);
    delay(d);
  }
}
```

Back to ShiftOut Tutorial

```
//**************************************************************//
//  Name    : shiftOutCode, Predefined Dual Array Style         //
//  Author  : Carlyn Maw, Tom Igoe                              //
//  Date    : 25 Oct, 2006                                      //
//  Version : 1.0                                              //
//  Notes   : Code for using a 74HC595 Shift Register           //
//          : to count from 0 to 255                            //
//****************************************************************

//Pin connected to ST_CP of 74HC595
int latchPin = 8;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;
////Pin connected to DS of 74HC595
int dataPin = 11;

//holders for infromation you're going to pass to shifting function
byte dataRED;
byte dataGREEN;
byte dataArrayRED[10];
byte dataArrayGREEN[10];

void setup() {
  //set pins to output because they are addressed in the main loop
  pinMode(latchPin, OUTPUT);
  Serial.begin(9600);

  //Arduino doesn't seem to have a way to write binary straight into the code
  //so these values are in HEX.  Decimal would have been fine, too.
  dataArrayRED[0] = 0xFF; //11111111
  dataArrayRED[1] = 0xFE; //11111110
  dataArrayRED[2] = 0xFC; //11111100
  dataArrayRED[3] = 0xF8; //11111000
  dataArrayRED[4] = 0xF0; //11110000
  dataArrayRED[5] = 0xE0; //11100000
  dataArrayRED[6] = 0xC0; //11000000
  dataArrayRED[7] = 0x80; //10000000
  dataArrayRED[8] = 0x00; //00000000
  dataArrayRED[9] = 0xE0; //11100000

  //Arduino doesn't seem to have a way to write binary straight into the code
  //so these values are in HEX.  Decimal would have been fine, too.
  dataArrayGREEN[0] = 0xFF; //11111111
  dataArrayGREEN[1] = 0x7F; //01111111
  dataArrayGREEN[2] = 0x3F; //00111111
  dataArrayGREEN[3] = 0x1F; //00011111
  dataArrayGREEN[4] = 0x0F; //00001111
  dataArrayGREEN[5] = 0x07; //00000111
  dataArrayGREEN[6] = 0x03; //00000011
  dataArrayGREEN[7] = 0x01; //00000001
  dataArrayGREEN[8] = 0x00; //00000000
  dataArrayGREEN[9] = 0x07; //00000111

  //function that blinks all the LEDs
  //gets passed the number of blinks and the pause time
  blinkAll_2Bytes(2,500);
}

void loop() {


  for (int j = 0; j < 10; j++) {
    //load the light sequence you want from array
    dataRED = dataArrayRED[j];
    dataGREEN = dataArrayGREEN[j];
    //ground latchPin and hold low for as long as you are transmitting
    digitalWrite(latchPin, 0);
    //move 'em out
    shiftOut(dataPin, clockPin, dataGREEN);
    shiftOut(dataPin, clockPin, dataRED);
    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(latchPin, 1);
    delay(300);
  }
}



// the heart of the program
void shiftOut(int myDataPin, int myClockPin, byte myDataOut) {
  // This shifts 8 bits out MSB first,
  //on the rising edge of the clock,
  //clock idles low

  //internal function setup
  int i=0;
  int pinState;
  pinMode(myClockPin, OUTPUT);
  pinMode(myDataPin, OUTPUT);

  //clear everything out just in case to
  //prepare shift register for bit shifting
  digitalWrite(myDataPin, 0);
  digitalWrite(myClockPin, 0);

  //for each bit in the byte myDataOut…
  //NOTICE THAT WE ARE COUNTING DOWN in our for loop
  //This means that %00000001 or "1" will go through such
  //that it will be pin Q0 that lights.
  for (i=7; i>=0; i--)  {
    digitalWrite(myClockPin, 0);

    //if the value passed to myDataOut and a bitmask result
    // true then... so if we are at i=6 and our value is
    // %11010100 it would the code compares it to %01000000
    // and proceeds to set pinState to 1.
    if ( myDataOut & (1<<i) ) {
      pinState= 1;
    }
    else {
      pinState= 0;
    }

    //Sets the pin to HIGH or LOW depending on pinState
    digitalWrite(myDataPin, pinState);
    //register shifts bits on upstroke of clock pin
    digitalWrite(myClockPin, 1);
    //zero the data pin after shift to prevent bleed through
    digitalWrite(myDataPin, 0);
  }

  //stop shifting
  digitalWrite(myClockPin, 0);
}


//blinks the whole register based on the number of times you want to
//blink "n" and the pause between them "d"
//starts with a moment of darkness to make sure the first blink
//has its full visual effect.
void blinkAll_2Bytes(int n, int d) {
  digitalWrite(latchPin, 0);
  shiftOut(dataPin, clockPin, 0);
  shiftOut(dataPin, clockPin, 0);
  digitalWrite(latchPin, 1);
  delay(200);
  for (int x = 0; x < n; x++) {
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 255);
    shiftOut(dataPin, clockPin, 255);
    digitalWrite(latchPin, 1);
    delay(d);
    digitalWrite(latchPin, 0);
    shiftOut(dataPin, clockPin, 0);
    shiftOut(dataPin, clockPin, 0);
    digitalWrite(latchPin, 1);
    delay(d);
  }
}
```

## Tutorial.ShiftOut History

Hide minor edits - Show changes to markup

May 23, 2007, at 11:26 AM by Paul Badger -
Changed lines 7-10 from:

At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more.

Users may also wish to search for other driver chips with "595" or "596" in their part numbers, there are many. The STP16C596 for example will drive 16 LED's and eliminates the series resistors with built-in constant current sources.

to:

At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more. (Users may also wish to search for other driver chips with "595" or "596" in their part numbers, there are many. The STP16C596 for example will drive 16 LED's and eliminates the series resistors with built-in constant current sources.)

Restore
May 23, 2007, at 11:23 AM by Paul Badger -
Changed lines 7-8 from:

At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more.

to:

At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more.

Users may also wish to search for other driver chips with "595" or "596" in their part numbers, there are many. The STP16C596 for example will drive 16 LED's and eliminates the series resistors with built-in constant current sources.

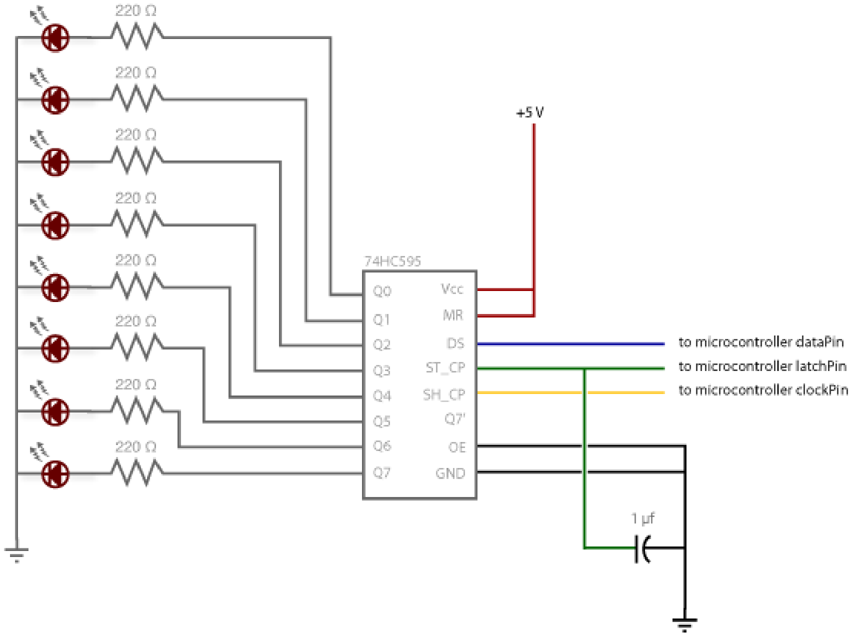
Restore
December 07, 2006, at 05:20 PM by Carlyn Maw -
Changed line 21 from:

(:cell:) Ground, Vss

to:

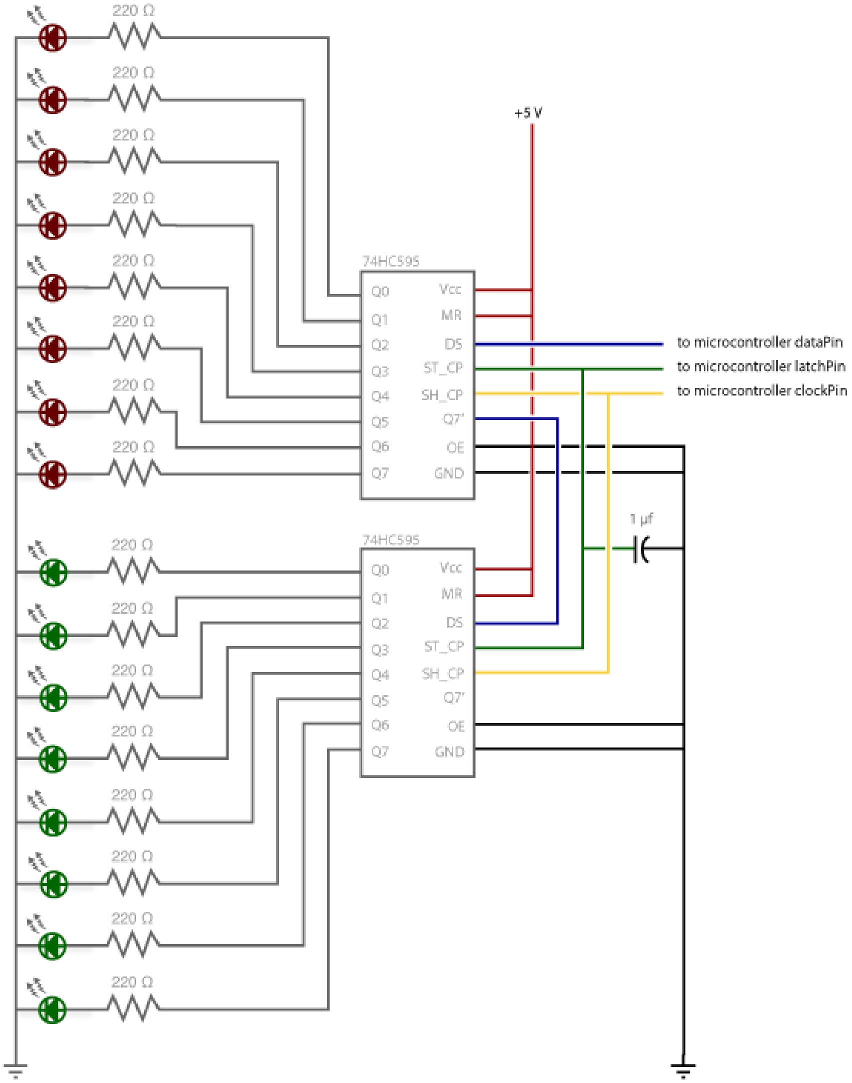
(:cell:) Output Pins

Restore
November 13, 2006, at 05:18 PM by Carlyn Maw -
Changed lines 83-84 from:
to:



Added lines 124-127:

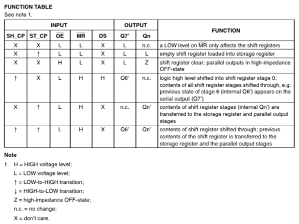
### Circuit Diagram


220 Ω
+5 V
74HC595
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Vcc
MR
DS
ST_CP
SH_CP
Q7'
OE
GND
to microcontroller dataPin
to microcontroller latchPin
to microcontroller clockPin
1 µf

Restore

November 09, 2006, at 04:25 PM by Carlyn Maw -

Changed lines 89-91 from:



FUNCTION TABLE

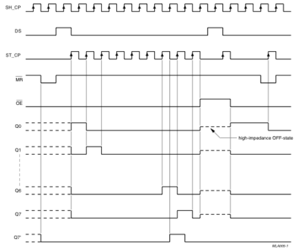
See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{\text{OE}}$ | $\overline{\text{MR}}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{\text{MR}}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

Note

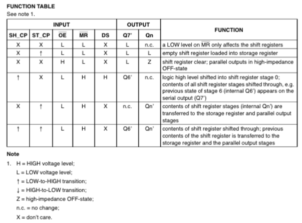
1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



595 Timing Diagram The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.

to:



**FUNCTION TABLE**

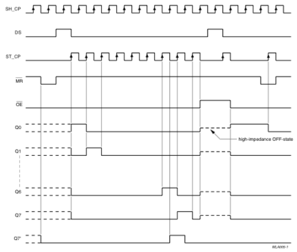
See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{OE}$ | $\overline{MR}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{MR}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



595 Timing Diagram

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.

Restore
November 09, 2006, at 04:22 PM by Carlyn Maw -
Changed lines 77-78 from:

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Some shift registers won't supply power, they will only ground. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

to:

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Using the shift register to supply power like this is called *sourcing current*. Some shift registers can't source current, they can only do what is called *sinking current.* If you have one of those it means you will have to flip the direction of the LEDs, putting the anodes directly to power and the cathodes (ground pins) to the shift register outputs. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

Added lines 100-101:

In this example you'll add a second shift register, doubling the number of output pins you have while still using the same number of pins from the Arduino.

Restore
November 09, 2006, at 04:10 PM by Carlyn Maw -
Changed lines 13-14 from:

“3 states” refers to the fact that you can set the output pins as either high, low or “high impedance.” Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Niether example takes advantage of this feature and you won’t usually need to worry about getting a chip that has it.

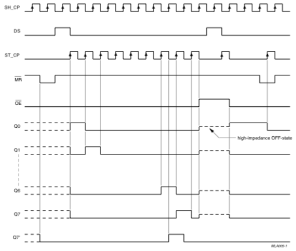
to:

“3 states” refers to the fact that you can set the output pins as either high, low or “high impedance.” Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Neither example takes advantage of this feature and you won’t usually need to worry about getting a chip that has it.

Restore
November 09, 2006, at 03:07 PM by Carlyn Maw -
Restore
November 09, 2006, at 03:07 PM by Carlyn Maw -
Added lines 87-88:

Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

Changed lines 90-97 from:

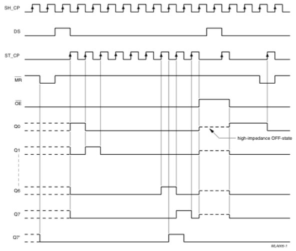


595 Timing Diagram

Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.

to:

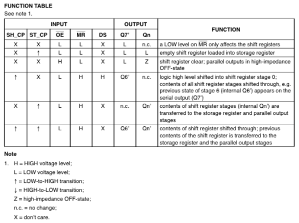


595 Timing Diagram The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.
Added lines 96-97:

Restore

November 09, 2006, at 03:04 PM by Carlyn Maw -

Added lines 87-89:



**FUNCTION TABLE**

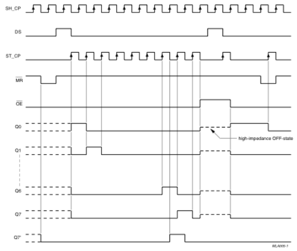
See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{OE}$ | $\overline{MR}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{MR}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

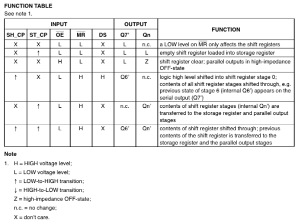
1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



595 Timing Diagram

Deleted lines 93-95:



**FUNCTION TABLE**

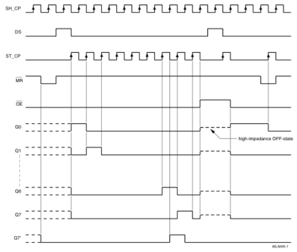
See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{OE}$ | $\overline{MR}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{MR}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



595 Timing Diagram

Restore

November 09, 2006, at 03:02 PM by Carlyn Maw -

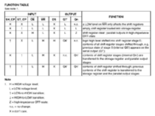
Changed lines 87-88 from:

http://www.arduino.cc/en/uploads/Tutorial/595_logic_table.png Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

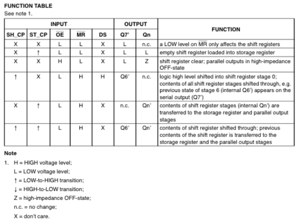
to:

Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

Changed lines 91-97 from:



595 Logic Table



**FUNCTION TABLE**

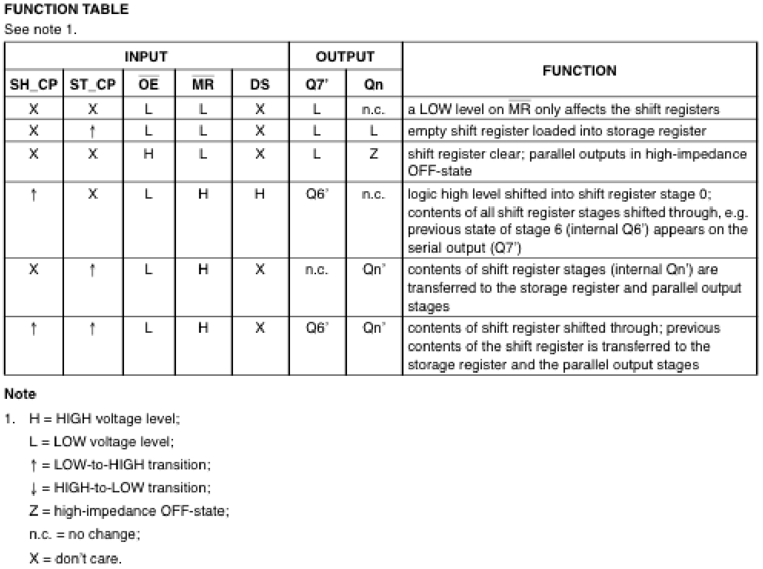
See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{\text{OE}}$ | $\overline{\text{MR}}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{\text{MR}}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



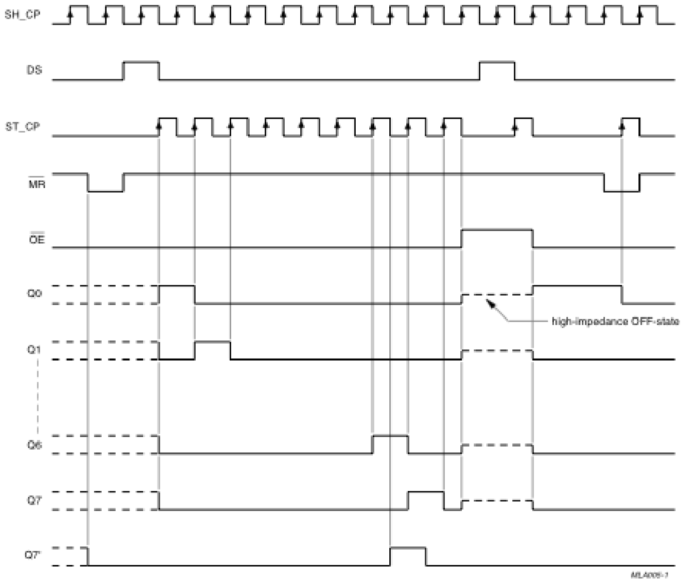
**FUNCTION TABLE**

See note 1.

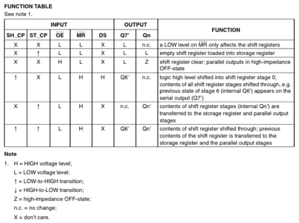
| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{\text{OE}}$ | $\overline{\text{MR}}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{\text{MR}}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.



to:



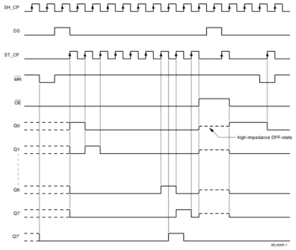
**FUNCTION TABLE**
See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{OE}$ | $\overline{MR}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{MR}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



595 Timing Diagram

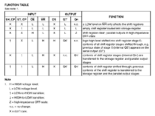
Restore
November 09, 2006, at 03:00 PM by Carlyn Maw -
Changed lines 87-88 from:

Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

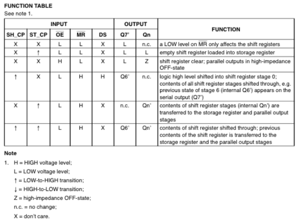
to:

http://www.arduino.cc/en/uploads/Tutorial/595_logic_table.png Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

Added lines 91-94:



595 Logic Table



**FUNCTION TABLE**

See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{OE}$ | $\overline{MR}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{MR}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

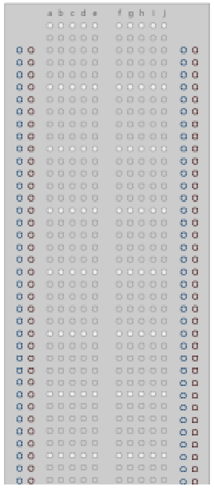
1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

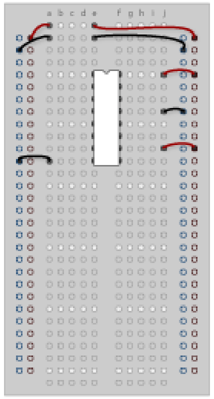
595 Logic Table

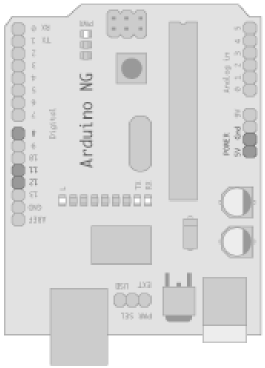
Restore

November 09, 2006, at 02:46 PM by Carlyn Maw -

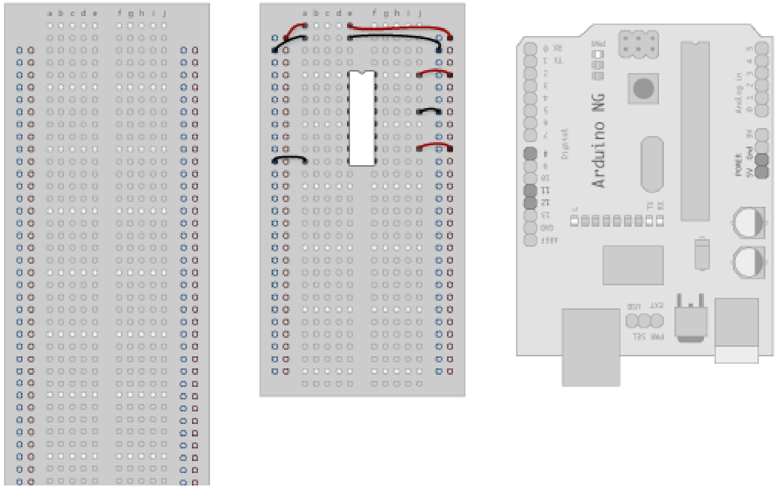
Deleted lines 54-55:



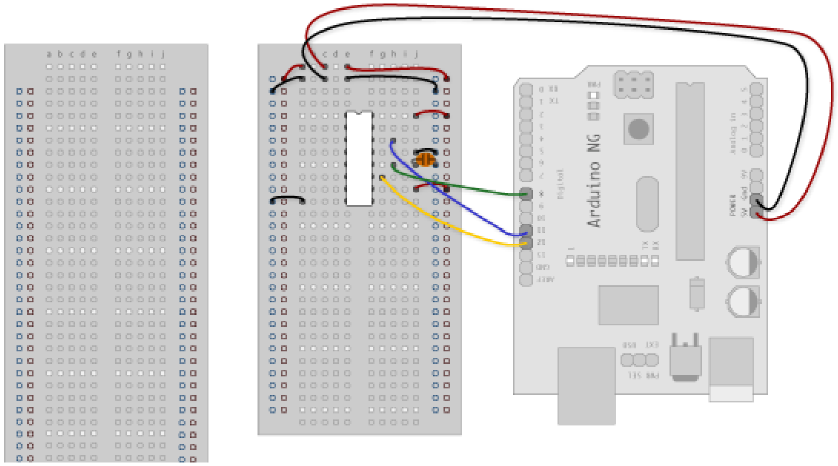




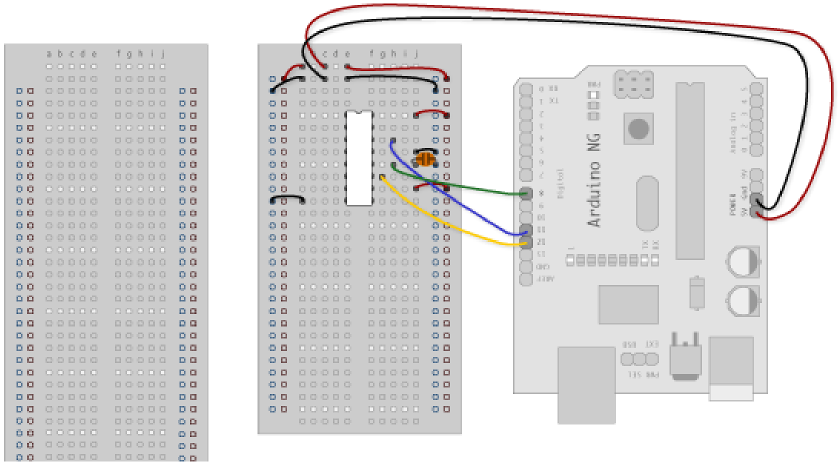
Added lines 63-64:



Deleted lines 66-67:



Added lines 73-74:



Added lines 77-78:

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Some shift registers won't supply power, they will only ground. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

Deleted lines 80-81:

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Some shift registers won't supply power, they will only ground. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

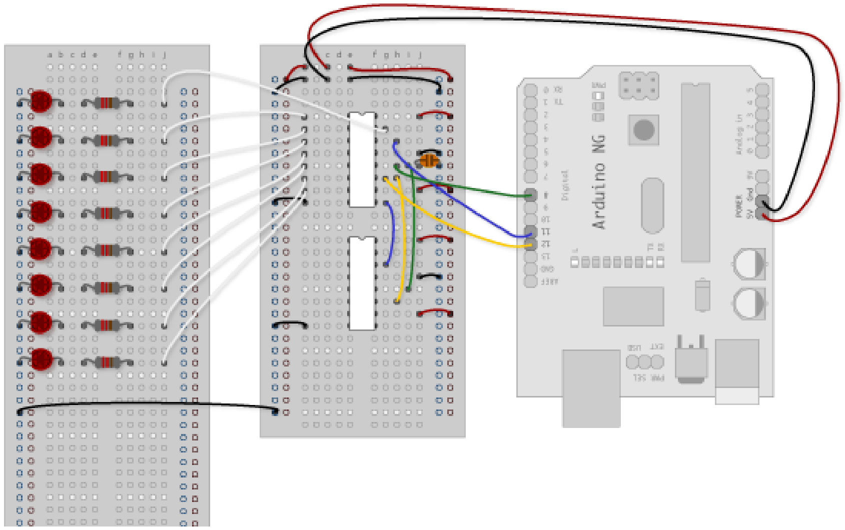
Added lines 104-105:

Starting from the previous example, you should put a second shift register on the board. It should have the same leads to power and ground.

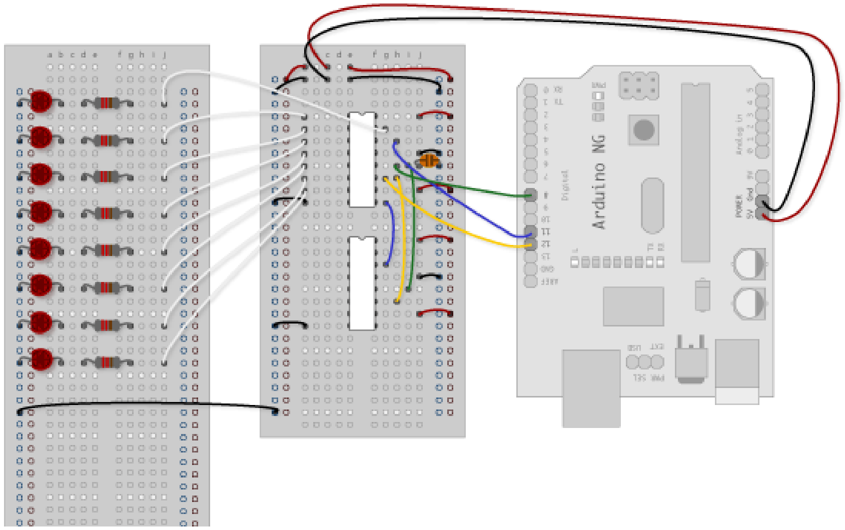
Deleted lines 107-108:

Starting from the previous example, you should put a second shift register on the board. It should have the same leads to power and ground.

Deleted lines 108-109:



Added lines 112-113:



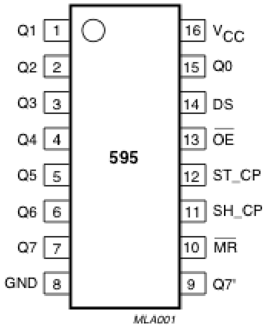
Added lines 116-117:

In this case I added green ones so when reading the code it is clear which byte is going to which set of LEDs

Deleted lines 119-120:

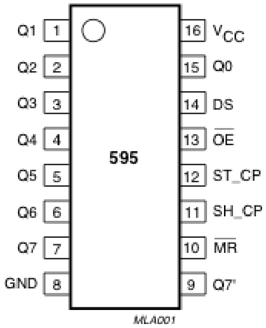
In this case I added green ones so when reading the code it is clear which byte is going to which set of LEDs

Restore
November 09, 2006, at 02:41 PM by Carlyn Maw -
Restore
November 09, 2006, at 02:41 PM by Carlyn Maw -
Changed line 18 from:



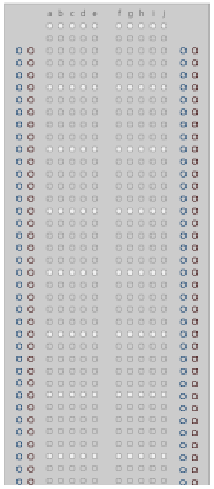
(:cell rowspan=9 :)

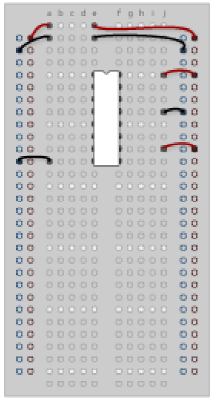
to:

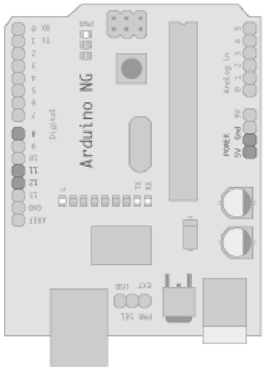


(:cell rowspan=9 :)

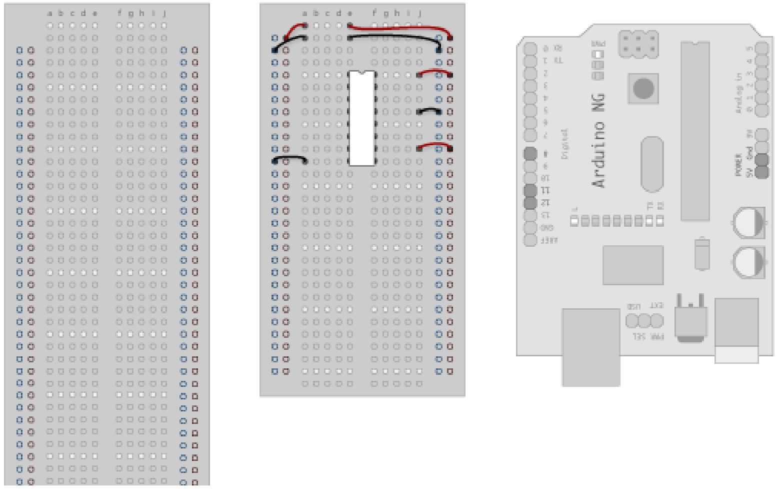
Changed lines 55-56 from:



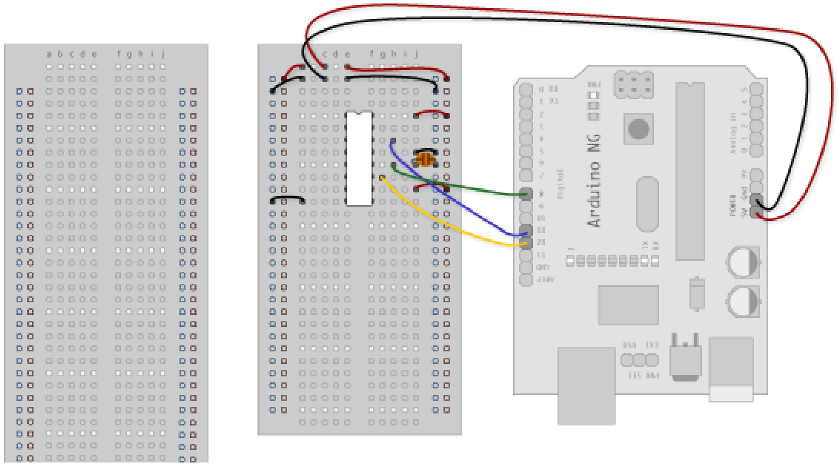




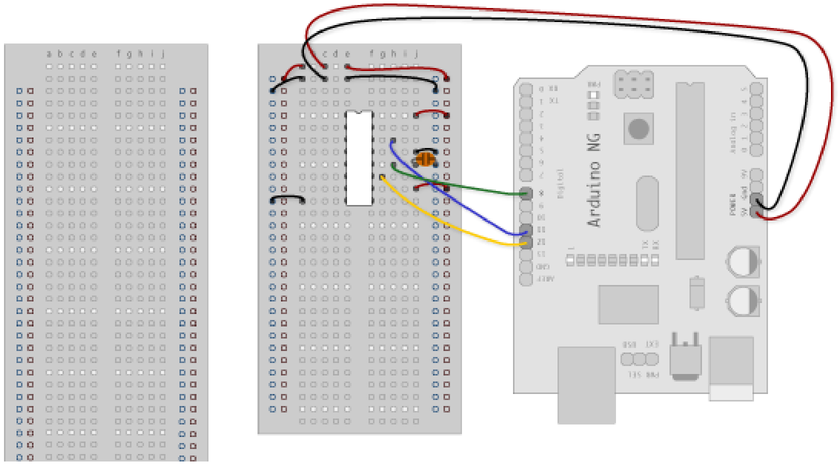
to:



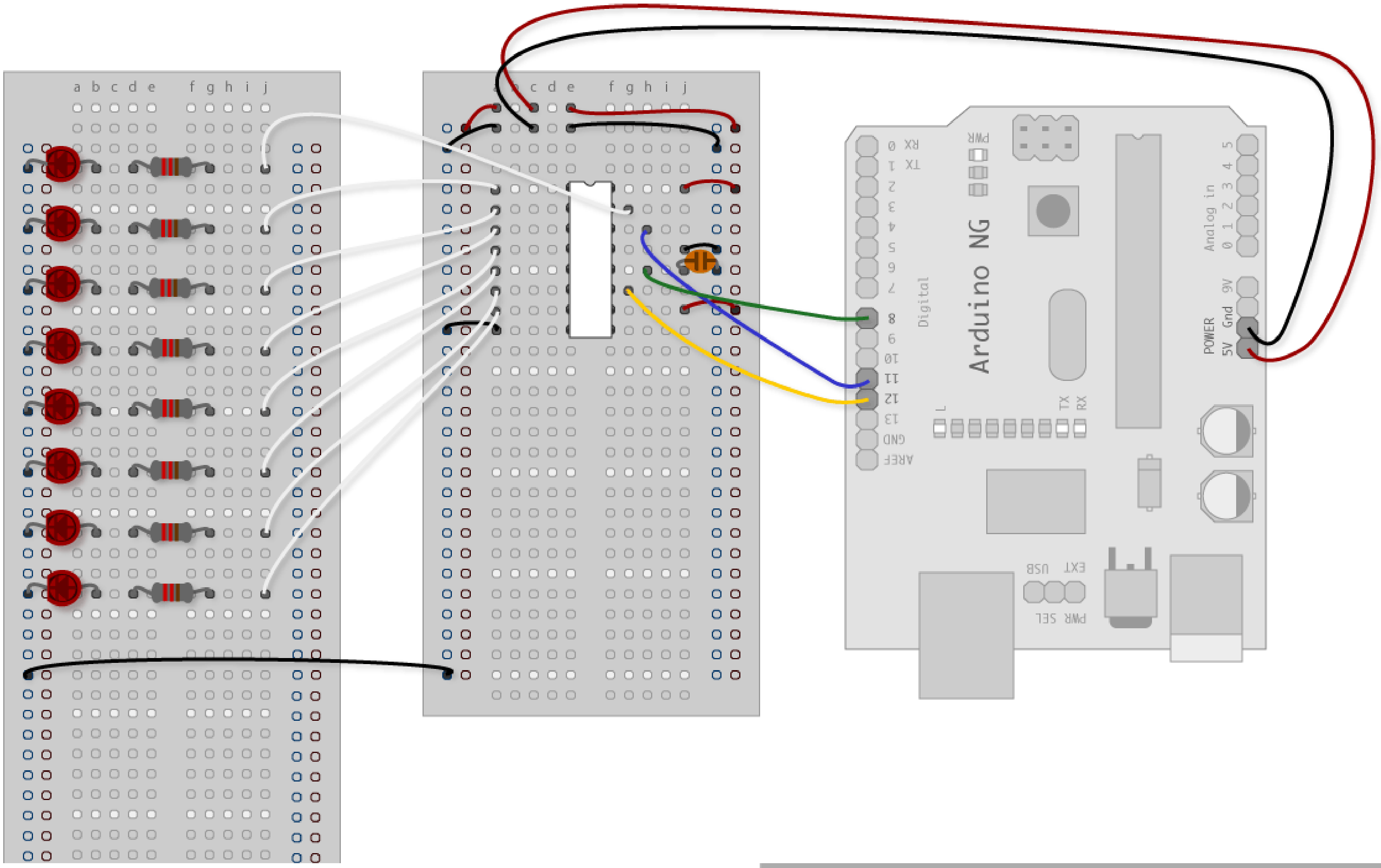
Changed lines 67-68 from:



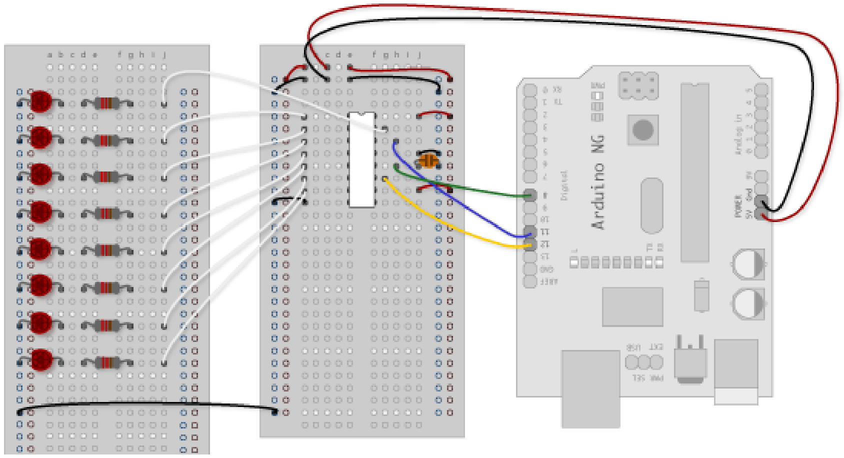
to:



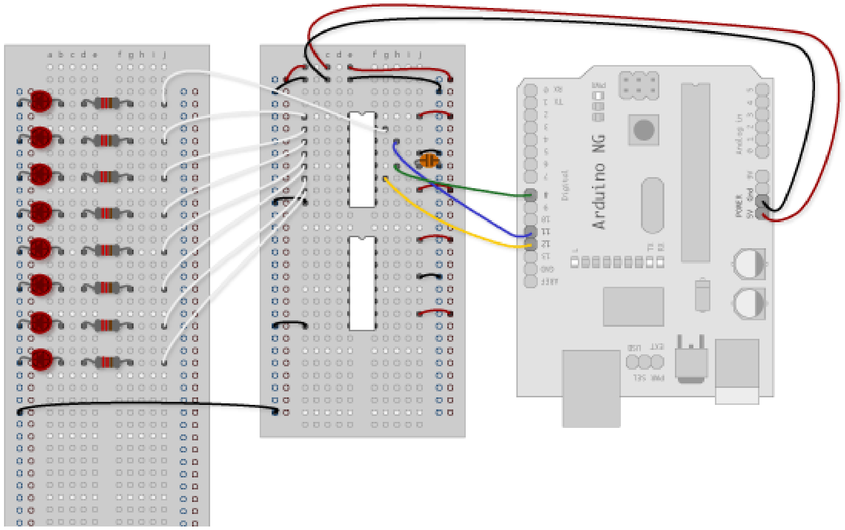
Changed lines 77-78 from:



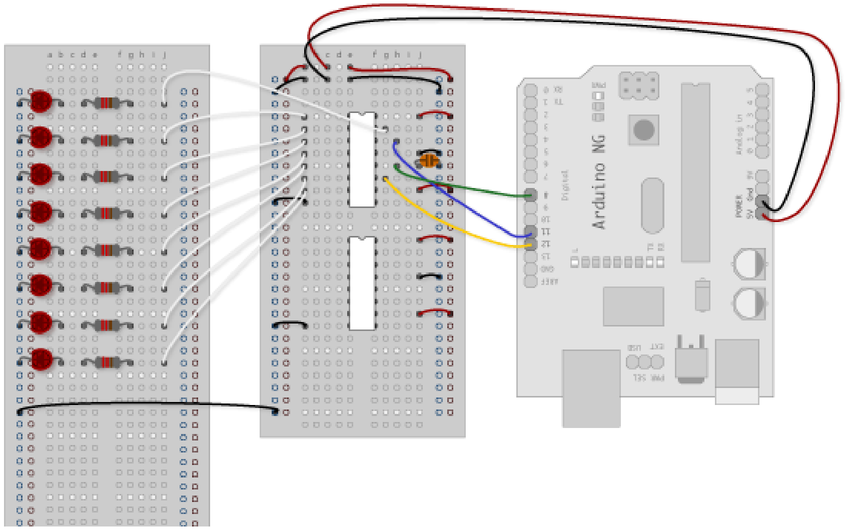
to:



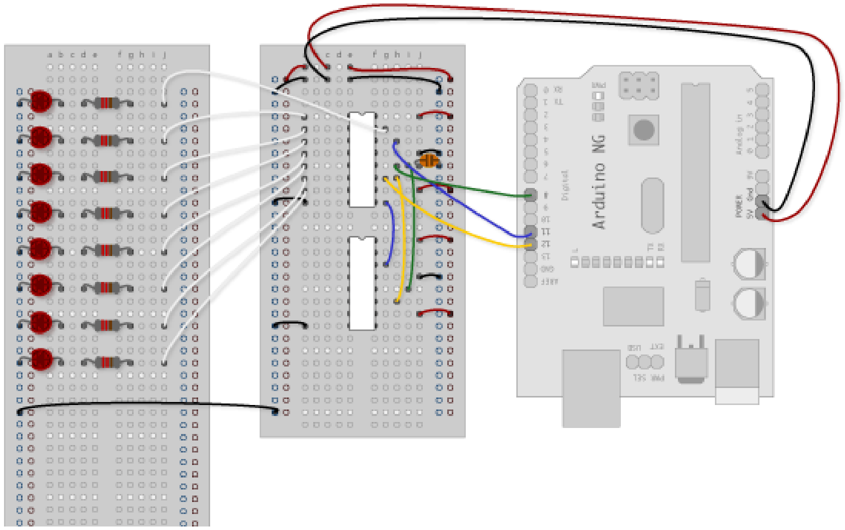
Changed lines 104-105 from:



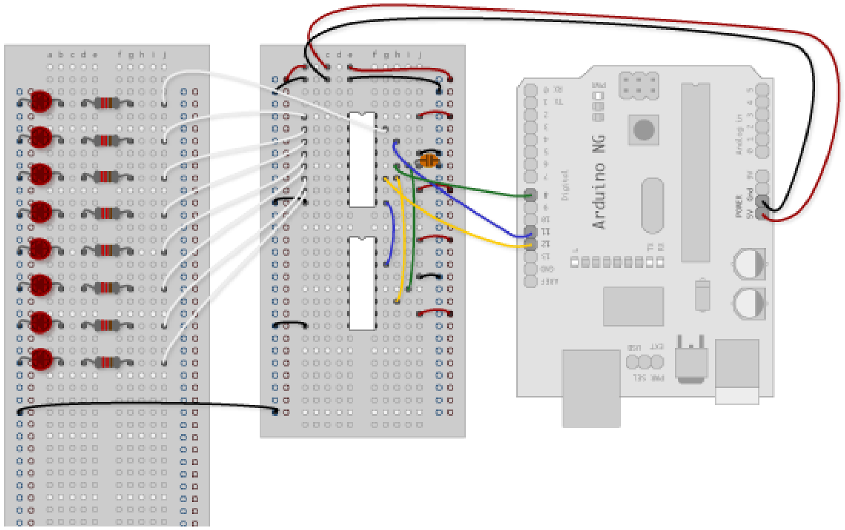
to:



Changed line 110 from:



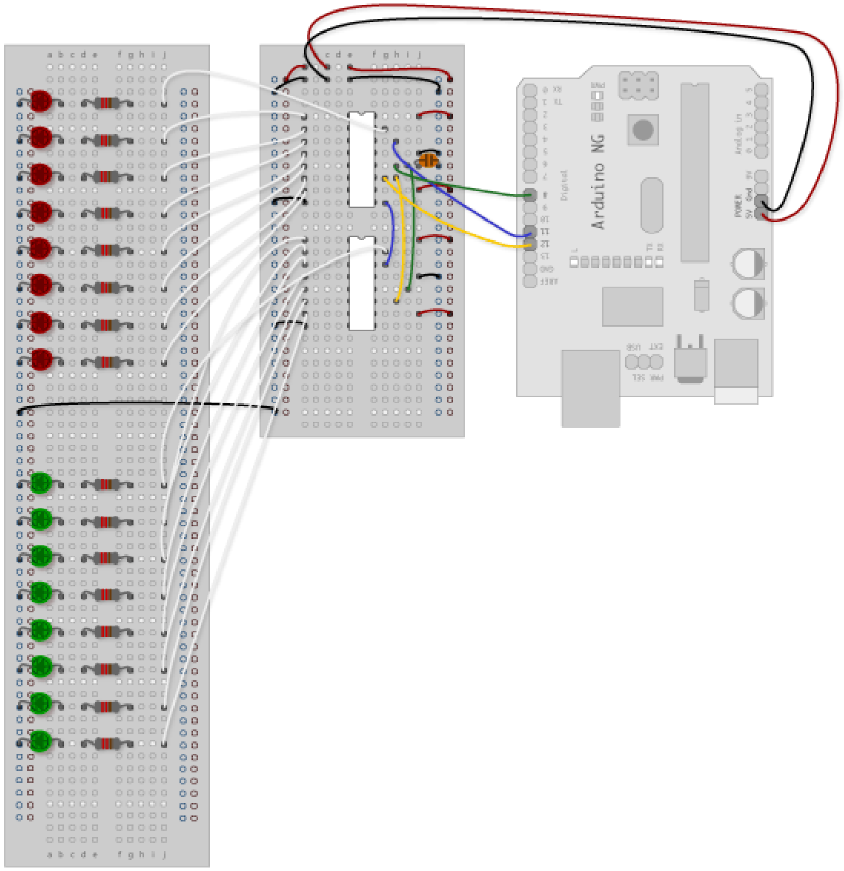
to:



Changed lines 116-117 from:

Attach:Exmp2_3.gif Δ

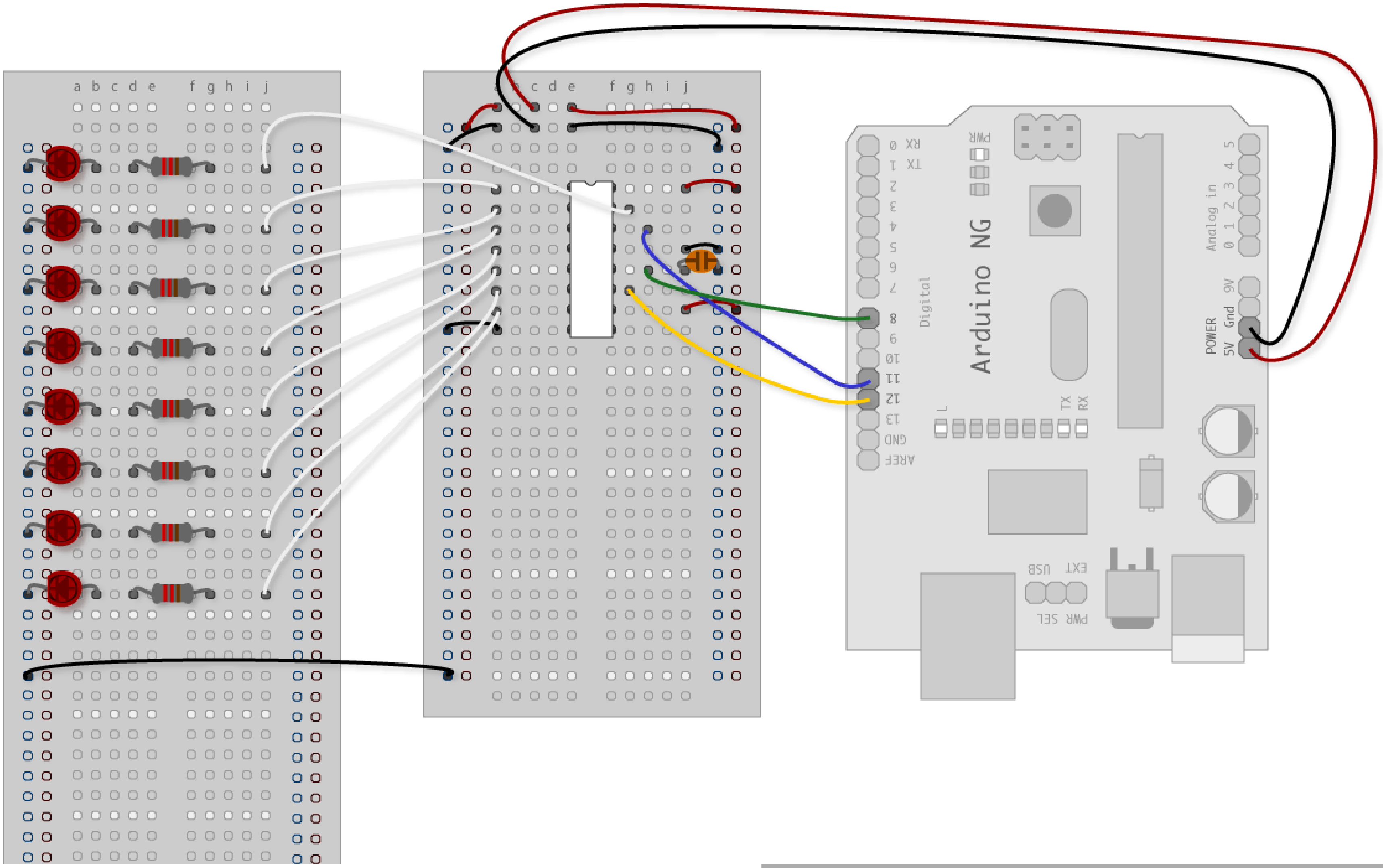
to:



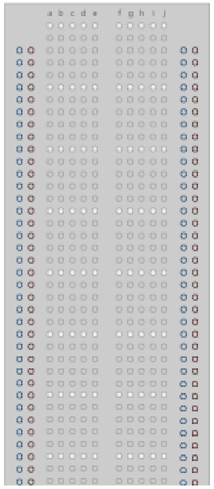
Restore
November 09, 2006, at 02:22 PM by Carlyn Maw -
Changed lines 77-78 from:

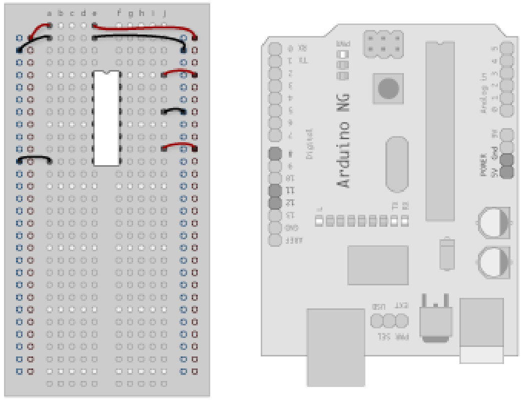
Attach:Exmp1_.gif Δ

to:

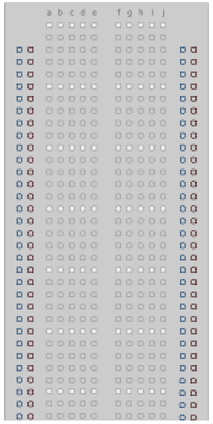


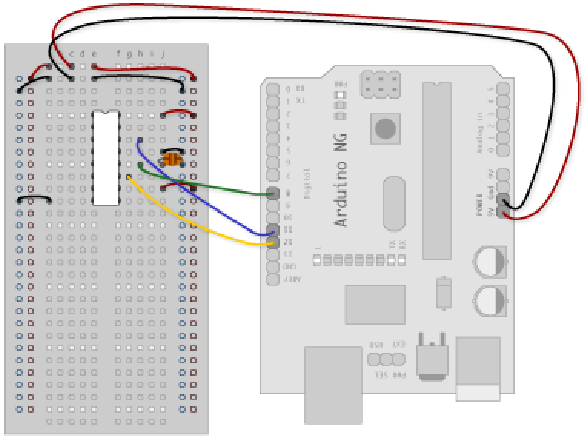
Restore
November 09, 2006, at 02:19 PM by Carlyn Maw -
Added lines 55-56:





Added lines 67-68:





Changed lines 75-76 from:

**Add 8 LEDs.**

to:

**3. Add 8 LEDs.**

Attach:Exmp1_.gif Δ

Added lines 83-84:
Added lines 91-93:

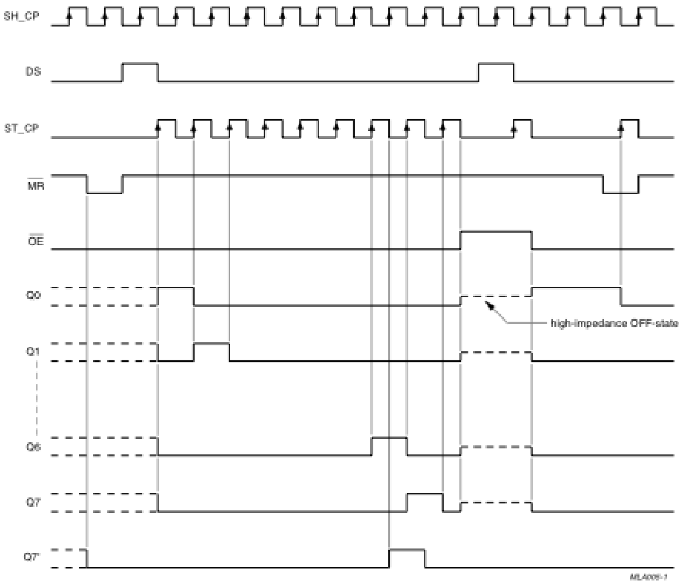
**FUNCTION TABLE**

See note 1.

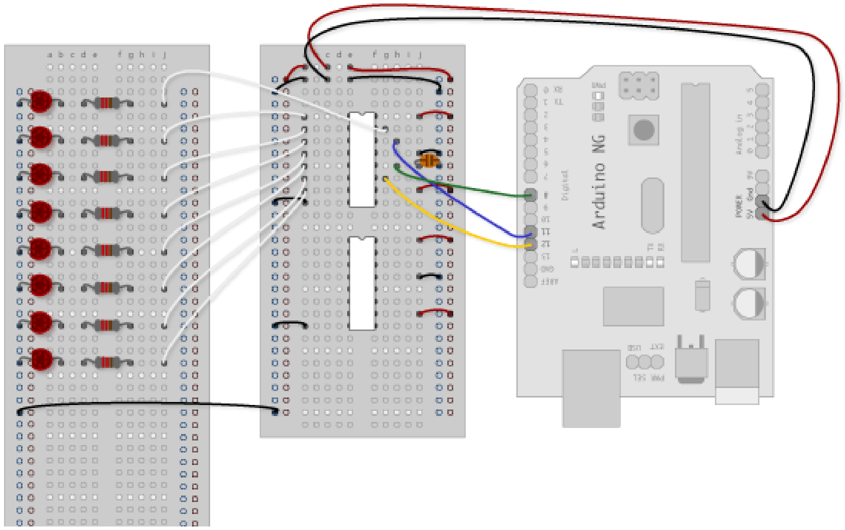
| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{\text{OE}}$ | $\overline{\text{MR}}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{\text{MR}}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

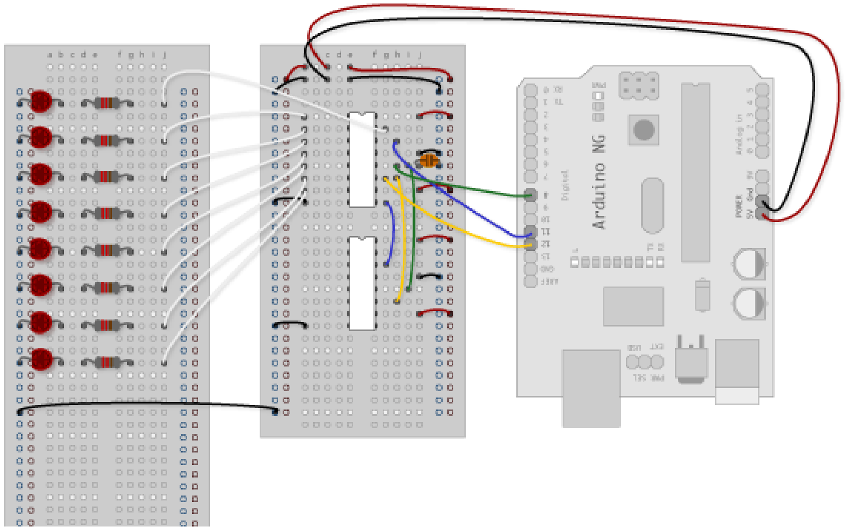
1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.



Added lines 104-105:



Added lines 109-110:



Added lines 115-117:

Attach:Exmp2_3.gif Δ

Restore
November 09, 2006, at 02:08 PM by Carlyn Maw -
Changed lines 15-17 from:

Here is a table explaining the pin-outs adapted from the datasheet.

(:table border=1 cellpadding=5 cellspacing=0:)

to:

**Here is a table explaining the pin-outs adapted from the Phillip's datasheet.**

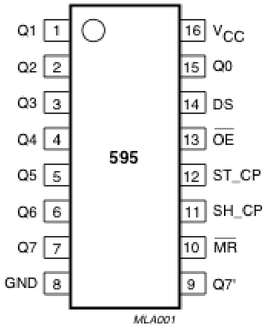
(:table border=1 bordercolor=#CCCCCC cellpadding=5 cellspacing=0:)

Restore
November 09, 2006, at 02:01 PM by Carlyn Maw -
Changed lines 18-48 from:

(:cell rowspan=9 :) a1 (:cell:) b1 (:cell:) c1 (:cell:) d1 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2

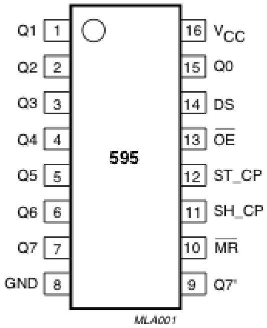
(:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2

to:



(:cell rowspan=9 :) (:cell:) PINS 1-7, 15 (:cell:) Q0 – Q7 (:cell:) Ground, Vss (:cellnr:) PIN 8 (:cell:) GND (:cell:) Ground, Vss (:cellnr:) PIN 9 (:cell:) Q7’ (:cell:) Serial Out (:cellnr:) PIN 10 (:cell:) MR (:cell:) Master Reclear, active low (:cellnr:) PIN 11 (:cell:) SH_CP (:cell:) Shift register clock pin (:cellnr:) PIN 12 (:cell:) ST_CP (:cell:) Storage register clock pin (latch pin) (:cellnr:) PIN 13 (:cell:) OE (:cell:) Output enable, active low (:cellnr:) PIN 14 (:cell:) DS (:cell:) Serial data input (:cellnr:) PIN 16 (:cell:) Vcc (:cell:) Positive supply voltage

Deleted lines 47-48:



Restore
November 09, 2006, at 01:53 PM by Carlyn Maw -
Added lines 17-50:

(:table border=1 cellpadding=5 cellspacing=0:) (:cell rowspan=9 :) a1 (:cell:) b1 (:cell:) c1 (:cell:) d1 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:cellnr:) b2 (:cell:) c2 (:cell:) d2 (:tableend:)

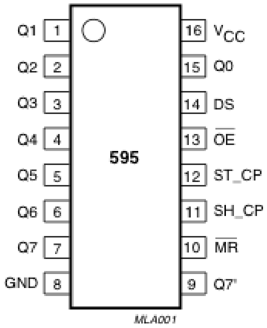
Restore
November 09, 2006, at 01:49 PM by Carlyn Maw -
Changed lines 13-14 from:

“3 states” refers to the fact that you can set the output pins as either high, low or “high impedance.” Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Niether example takes advantage of this feature and you won’t usually need to worry about getting a chip that has it.

to:

“3 states” refers to the fact that you can set the output pins as either high, low or “high impedance.” Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Niether example takes advantage of this feature and you won't usually need to worry about getting a chip that has it.

Here is a table explaining the pin-outs adapted from the datasheet.



Restore
November 09, 2006, at 01:45 PM by Carlyn Maw -
Changed lines 13-14 from:

"3 states" refers to the fact that you can set the output pins as either high, low or “high impedance.” When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn't actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can't set pins to their high impedance state individually, you can only set the whole chip together. You'd do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature as it is a pretty specialized thing to do, so you don't need to spend a lot of time on it now.

to:

“3 states” refers to the fact that you can set the output pins as either high, low or “high impedance.” Unlike the HIGH and LOW states, you can't set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Niether example takes advantage of this feature and you won't usually need to worry about getting a chip that has it.

Restore
November 01, 2006, at 07:38 PM by Carlyn Maw -
Changed lines 9-10 from:

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. It's by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be independently set to an agreed upon specified data rate. Once the whole byte gets to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

to:

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. It's by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast to using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be set independently to an agreed upon specified data rate. Once the whole byte is transmitted to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

Changed lines 13-14 from:

"3 states" refers to the fact that you can set the output pins as either high, low or “high impedance.” When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn't actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can't set pins to their high impedance state individually, you can only set the whole chip together. You'd do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature, and it is a pretty specialized thing to do, so you don't need to spend a lot of time on it now.

to:

"3 states" refers to the fact that you can set the output pins as either high, low or "high impedance." When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn't actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can't set pins to their high impedance state individually, you can only set the whole chip together. You'd do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature as it is a pretty specialized thing to do, so you don't need to spend a lot of time on it now.

Changed lines 28-29 from:

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. You can get around this by also controlling the MR and OE pins from your Arduino board, but this will work and leave you with more open pins.

to:

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. You can get around this by controlling the MR and OE pins from your Arduino board too, but this way will work and leave you with more open pins.

Changed lines 36-37 from:

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin, if you notice some flicker every time the latch pin pulses you can use a capacitor to even it out.

to:

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin, if you have some flicker when the latch pin pulses you can use a capacitor to even it out.

Changed lines 40-41 from:

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each to its respective shift register output pin. Some shift registers won't supply power, they will only ground. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

to:

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Some shift registers won't supply power, they will only ground. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

Changed lines 48-49 from:

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register on the shift register. When the latchPin goes from low to high the sent data gets moved from the aforementioned memory register into the output pins, lighting the LEDs.

to:

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.

Restore
November 01, 2006, at 07:28 PM by Carlyn Maw -
Changed lines 9-10 from:

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. Its by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be independently set to an agreed upon specified data rate. Once the whole byte gets to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

to:

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. It's by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be independently set to an agreed upon specified data rate. Once the whole byte gets to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

Changed lines 13-14 from:

The 3 states refers to the fact that you can set the output pins as either high, low or “high impedance.” When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn’t actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually, you can only set the whole chip together. You’d do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature, and it is a pretty specialized thing to do, so you don’t need to spend a lot of time on it now.

to:

"3 states" refers to the fact that you can set the output pins as either high, low or “high impedance.” When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn’t actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually, you can only set the whole chip together. You’d do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature, and it is a pretty specialized thing to do, so you don’t need to spend a lot of time on it now.

Restore
November 01, 2006, at 06:51 PM by Carlyn Maw -
Restore
November 01, 2006, at 06:51 PM by Carlyn Maw -
Changed lines 80-81 from:

2.3 also takes advantage of the new blinkAll_2bytes() function. Its big difference from 1.3 is only that instead of just a single variable called “data” and a single array called “dataArray” you have to have a dataRED, a dataGREEN, dataArrayRED, dataArrayGREEN defined up front. This means that later on line

to:

Like sample 2.2, sample 2.3 also takes advantage of the new blinkAll_2bytes() function. 2.3's big difference from sample 1.3 is only that instead of just a single variable called “data” and a single array called “dataArray” you have to have a dataRED, a dataGREEN, dataArrayRED, dataArrayGREEN defined up front. This means that line

Restore
November 01, 2006, at 06:46 PM by Carlyn Maw -
Changed line 79 from:

Code Sample 2.3 - Dual Defined Arrays

to:

Code Sample 2.3 - Dual Defined Arrays\\

Restore

November 01, 2006, at 06:45 PM by Carlyn Maw -
Changed lines 3-4 from:

Carlyn Maw, Tom Igoe

to:

Started by Carlyn Maw and Tom Igoe Nov, 06

Changed line 73 from:

Code Sample 2.1 – Dual Binary Counters

to:

Code Sample 2.1 – Dual Binary Counters\\

Added lines 75-97:

Code Sample 2.2 – 2 Byte One By One
Comparing this code to the similar code from Example 1 you see that a little bit more has had to change. The blinkAll() function has been changed to the blinkAll_2Bytes() function to reflect the fact that now there are 16 LEDs to control. Also, in version 1 the pulsings of the latchPin were situated inside the subfunctions lightShiftPinA and lightShiftPinB(). Here they need to be moved back into the main loop to accommodate needing to run each subfunction twice in a row, once for the green LEDs and once for the red ones.

Code Sample 2.3 - Dual Defined Arrays 2.3 also takes advantage of the new blinkAll_2bytes() function. Its big difference from 1.3 is only that instead of just a single variable called “data” and a single array called “dataArray” you have to have a dataRED, a dataGREEN, dataArrayRED, dataArrayGREEN defined up front. This means that later on line

```
data = dataArray[j];
```

becomes

```
dataRED = dataArrayRED[j];
dataGREEN = dataArrayGREEN[j];
```

and

```
shiftOut(dataPin, clockPin, data);
```

becomes

```
shiftOut(dataPin, clockPin, dataGREEN);
shiftOut(dataPin, clockPin, dataRED);
```

Restore
November 01, 2006, at 06:41 PM by Carlyn Maw -
Added lines 53-74:

## Example 2

### The Circuit

**1. Add a second shift register.**

Starting from the previous example, you should put a second shift register on the board. It should have the same leads to power and ground.

**2. Connect the 2 registers.**

Two of these connections simply extend the same clock and latch signal from the Arduino to the second shift register (yellow and green wires). The blue wire is going from the serial out pin (pin 9) of the first shift register to the serial data input (pin 14) of the second register.

**3. Add a second set of LEDs.**

In this case I added green ones so when reading the code it is clear which byte is going to which set of LEDs

### The Code

Here again are three code samples. If you are curious, you might want to try the samples from the first example with this circuit set up just to see what happens.

Code Sample 2.1 – Dual Binary Counters There is only one extra line of code compared to the first code sample from Example 1. It sends out a second byte. This forces the first shift register, the one directly attached to the Arduino, to pass the first byte sent through to the second register, lighting the green LEDs. The second byte will then show up on the red LEDs.

Restore
November 01, 2006, at 06:34 PM by Carlyn Maw -
Changed lines 50-51 from:

Code Sample 1.1 – Hello World Code Sample 1.2 – One by One

to:

Code Sample 1.1 – Hello World
Code Sample 1.2 – One by One
Code Sample 1.3 – from Defined Array\\

Restore
November 01, 2006, at 06:32 PM by Carlyn Maw -
Added line 51:

Code Sample 1.2 – One by One

Restore
November 01, 2006, at 06:27 PM by Carlyn Maw -
Added lines 43-50:

### The Code

Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register on the shift register. When the latchPin goes from low to high the sent data gets moved from the aforementioned memory register into the output pins, lighting the LEDs.

Code Sample 1.1 – Hello World

Restore
November 01, 2006, at 06:25 PM by Carlyn Maw -
Changed lines 36-42 from:

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin. I was getting some flicker every time the latch pin pulsed so I used the capacitor to even it out.

to:

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin, if you notice some flicker every time the latch pin pulses you can use a capacitor to even it out.

**Add 8 LEDs.**

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each to its respective shift register output pin. Some shift registers won't supply power, they will only ground. You should check the your specific datasheet if you aren’t using a 595 series chip. Don’t forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

**Circuit Diagram**

Restore
November 01, 2006, at 06:19 PM by Carlyn Maw -
Changed lines 3-4 from:

by Carlyn Maw

to:

Carlyn Maw, Tom Igoe

Changed line 36 from:

I’m going to refer to them from now on as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin. I was getting some flicker every time the latch pin pulsed so I used the capacitor to even it out.

to:

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin. I was getting some flicker every time the latch pin pulsed so I used the capacitor to even it out.

Restore
November 01, 2006, at 06:18 PM by Carlyn Maw -
Deleted lines 27-49:

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. ======= At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more.

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. Its by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be independently set to an agreed upon specified data rate. Once the whole byte gets to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

The “serial output” part of this component comes from its extra pin which can pass the serial information received from the microcontroller out again unchanged. This means you can transmit 16 bits in a row (2 bytes) and the first 8 will flow through the first register into the second register and be expressed there. You can learn to do that from the second example.

The 3 states refers to the fact that you can set the output pins as either high, low or “high impedance.” When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn’t actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually, you can only set the whole chip together. You’d do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature, and it is a pretty specialized thing to do, so you don’t need to spend a lot of time on it now.

## Example 1: One Shift Register

The first step is to extend your Arduino with one shift register.

### The Circuit

#### 1. Turning it on

Make the following connections:

- GND (pin 8) to ground,
- Vcc (pin 16) to 5V
- OE (pin 13) to ground
- MR (pin 10) to 5V

Changed line 36 from:

I’m going to refer to them from now on as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin. I was getting some flicker every time the latch pin pulsed so I used the capacitor to even it out.

to:

I’m going to refer to them from now on as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin. I was getting some flicker every time the latch pin pulsed so I used the capacitor to even it out.

Restore
November 01, 2006, at 06:17 PM by Carlyn Maw -
Changed lines 51-52 from:

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. >>>>>>>

to:

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. You can get around this by also controlling the MR and OE pins from your Arduino board, but this will work and leave you with more open pins.

**2. Connect to Arduino**

- DS (pin 14) to Ardunio DigitalPin 11 (blue wire)
- SH_CP (pin 11) to to Ardunio DigitalPin 12 (yellow wire)
- ST_CP (pin 12) to Ardunio DigitalPin 8 (green wire)

I'm going to refer to them from now on as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin. I was getting some flicker every time the latch pin pulsed so I used the capacitor to even it out.

Restore
November 01, 2006, at 06:13 PM by Carlyn Maw -
Added lines 2-4:

by Carlyn Maw

Deleted line 6:

<<<<<<<

Restore
November 01, 2006, at 06:13 PM by Carlyn Maw -
Added line 4:

<<<<<<<

Changed lines 26-50 from:

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run.

to:

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. ======= At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an "8-bit serial-in, serial or parallel-out shift register with output latches; 3-state." In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more.

How this all works is through something called "synchronous serial communication," i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. Its by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast using the "asynchronous serial communication" of the Serial.begin() function which relies on the sender and the receiver to be independently set to an agreed upon specified data rate. Once the whole byte gets to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the "parallel output" part, having all the pins do what you want them to do all at once.

The "serial output" part of this component comes from its extra pin which can pass the serial information received from the microcontroller out again unchanged. This means you can transmit 16 bits in a row (2 bytes) and the first 8 will flow through the first register into the second register and be expressed there. You can learn to do that from the second example.

The 3 states refers to the fact that you can set the output pins as either high, low or "high impedance." When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn't actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can't set pins to their high impedance state individually, you can only set the whole chip together. You'd do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature, and it is a pretty specialized thing to do, so you don't need to spend a lot of time on it now.

## Example 1: One Shift Register

The first step is to extend your Arduino with one shift register.

### The Circuit

**1. Turning it on**

Make the following connections:

- GND (pin 8) to ground,
- Vcc (pin 16) to 5V
- OE (pin 13) to ground
- MR (pin 10) to 5V

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. >>>>>>>

Restore
November 01, 2006, at 06:13 PM by Carlyn Maw -
Added lines 11-25:

## Example 1: One Shift Register

The first step is to extend your Arduino with one shift register.

### The Circuit

**1. Turning it on**

Make the following connections:

- GND (pin 8) to ground,
- Vcc (pin 16) to 5V
- OE (pin 13) to ground
- MR (pin 10) to 5V

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you might end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run.

Restore
November 01, 2006, at 06:09 PM by Carlyn Maw -
Added lines 1-10:

# Serial to Parallel Shifting-Out with a 74HC595

## Shifting Out & the 595 chip

At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more.

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. Its by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be independently set to an agreed upon specified data rate. Once the whole byte gets to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

The “serial output” part of this component comes from its extra pin which can pass the serial information received from the microcontroller out again unchanged. This means you can transmit 16 bits in a row (2 bytes) and the first 8 will flow through the first register into the second register and be expressed there. You can learn to do that from the second example.

The 3 states refers to the fact that you can set the output pins as either high, low or “high impedance.” When you set a pin set high by sending a 1 bit to that address, it will output whatever voltage you have connected to the Vcc pin. When you set low it will output zero volts (Vss). When a pin is in a high impedance state, the shift register isn’t actively set to either a high or low voltage. High impedance is meant to be a type of blank state so if you wanted to have the outputs attached to one register controlled by, for example, a second register attached in parallel to the original circuit you could do so without competition. Think of an LED array that might need to be controlled by different microcontrollers depending on a mode built into your project. Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually, you can only set the whole chip together. You’d do this by setting the Output-Enable (pin 13) HIGH and the Master-Reclear (pin) LOW. Neither example takes advantage of this feature, and it is a pretty specialized thing to do, so you don’t need to spend a lot of time on it now.

Restore

# Serial to Parallel Shifting-Out with a 74HC595

Started by Carlyn Maw and Tom Igoe Nov, 06

## Shifting Out & the 595 chip

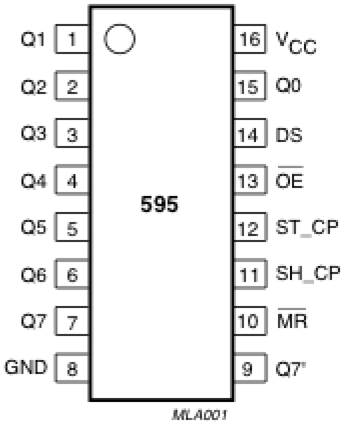
At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an “8-bit serial-in, serial or parallel-out shift register with output latches; 3-state.” In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more. (Users may also wish to search for other driver chips with "595" or "596" in their part numbers, there are many. The STP16C596 for example will drive 16 LED's and eliminates the series resistors with built-in constant current sources.)

How this all works is through something called “synchronous serial communication,” i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. It's by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast to using the “asynchronous serial communication” of the Serial.begin() function which relies on the sender and the receiver to be set independently to an agreed upon specified data rate. Once the whole byte is transmitted to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the “parallel output” part, having all the pins do what you want them to do all at once.

The “serial output” part of this component comes from its extra pin which can pass the serial information received from the microcontroller out again unchanged. This means you can transmit 16 bits in a row (2 bytes) and the first 8 will flow through the first register into the second register and be expressed there. You can learn to do that from the second example.

“3 states” refers to the fact that you can set the output pins as either high, low or “high impedance.” Unlike the HIGH and LOW states, you can’t set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Neither example takes advantage of this feature and you won’t usually need to worry about getting a chip that has it.

**Here is a table explaining the pin-outs adapted from the** Phillip's datasheet**.**

| | | | |
|---|---|---|---|
|  Q1<br>Q2<br>Q3<br>Q4<br>Q5<br>Q6<br>Q7<br>GND<br>1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>16<br>15<br>14<br>13<br>12<br>11<br>10<br>9<br>VCC<br>Q0<br>DS<br>OE<br>ST_CP<br>SH_CP<br>MR<br>Q7'<br>595<br>MLA001 | PINS 1-7, 15 | Q0 – Q7 | Output Pins |
| | PIN 8 | GND | Ground, Vss |
| | PIN 9 | Q7’ | Serial Out |
| | PIN 10 | MR | Master Reclear, active low |
| | PIN 11 | SH_CP | Shift register clock pin |
| | PIN 12 | ST_CP | Storage register clock pin (latch pin) |
| | PIN 13 | OE | Output enable, active low |
| | PIN 14 | DS | Serial data input |
| | PIN 16 | Vcc | Positive supply voltage |

# Example 1: One Shift Register

The first step is to extend your Arduino with one shift register.

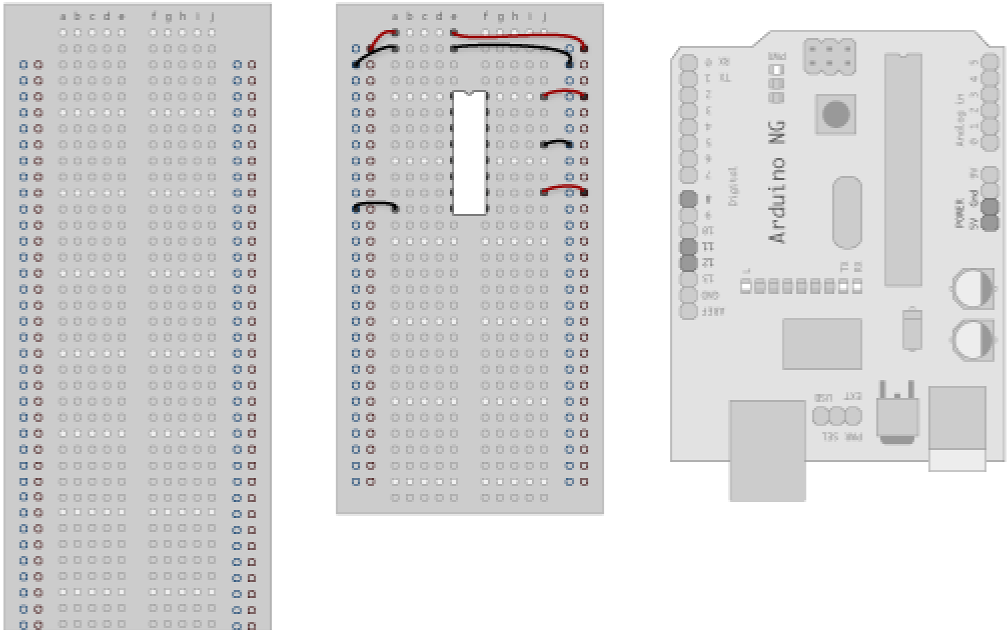
## The Circuit

### 1. Turning it on

Make the following connections:

- GND (pin 8) to ground,
- Vcc (pin 16) to 5V
- OE (pin 13) to ground
- MR (pin 10) to 5V

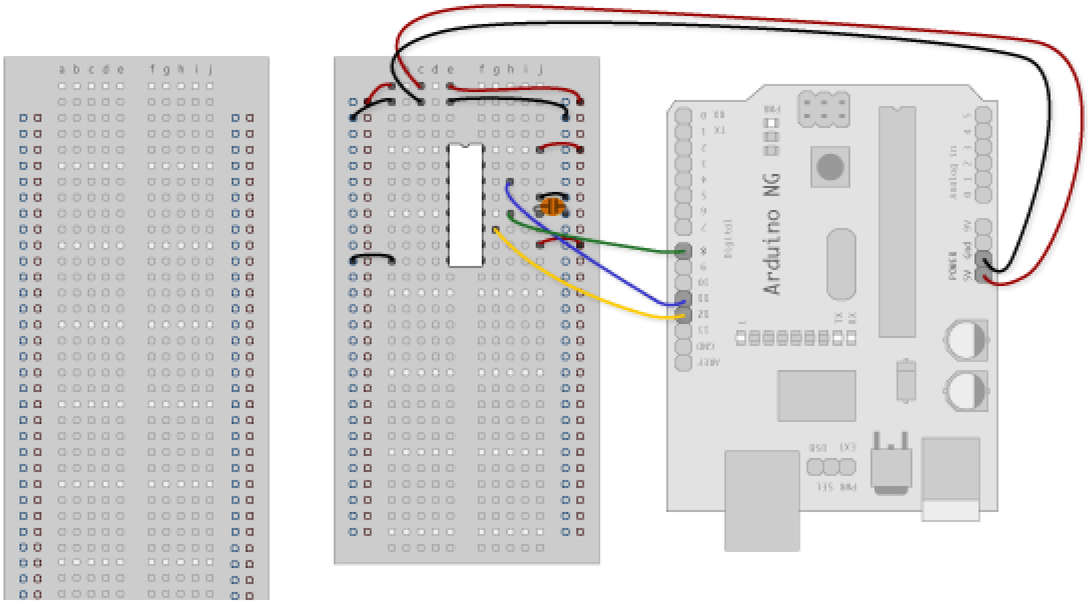
This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. You can get around this by controlling the MR and OE pins from your Arduino board too, but this way will work and leave you with more open pins.



### 2. Connect to Arduino

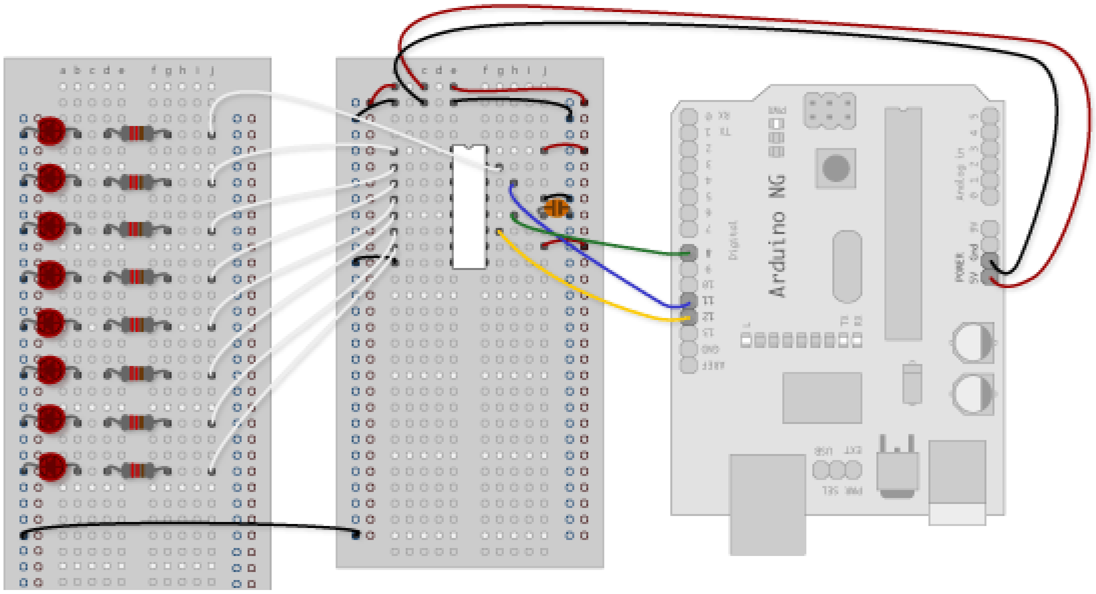
- DS (pin 14) to Ardunio DigitalPin 11 (blue wire)
- SH_CP (pin 11) to to Ardunio DigitalPin 12 (yellow wire)
- ST_CP (pin 12) to Ardunio DigitalPin 8 (green wire)

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1µf capacitor on the latchPin, if you have some flicker when the latch pin pulses you can use a capacitor to even it out.

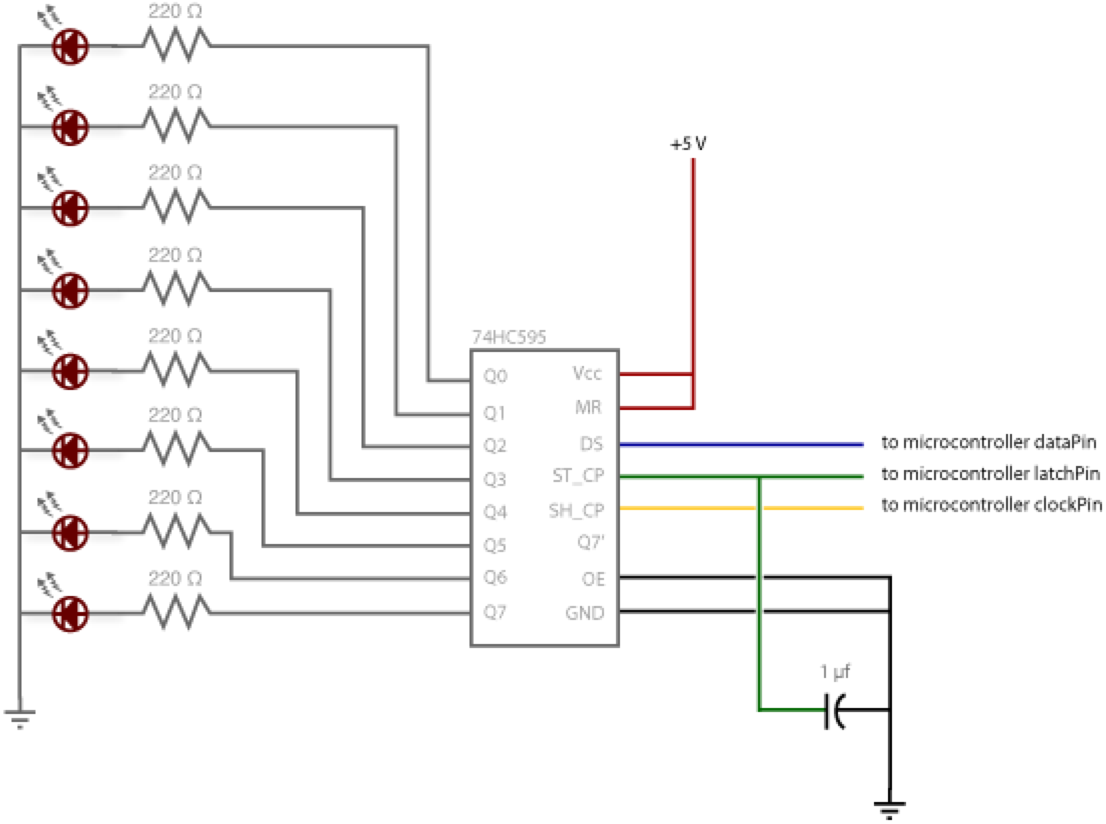


### 3. Add 8 LEDs.

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Using the shift register to supply power like this is called *sourcing current.* Some shift registers can't source current, they can only do what is called *sinking current.* If you have one of those it means you will have to flip the direction of the LEDs, putting the anodes directly to power and the cathodes (ground pins) to the shift register outputs. You should check the your specific datasheet if you aren’t using a 595 series chip. Don’t forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.



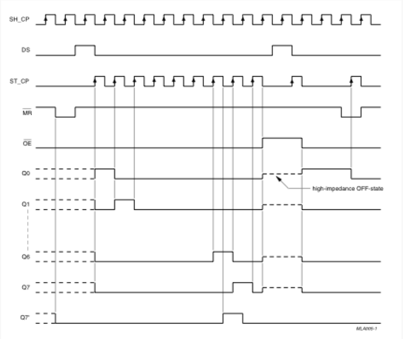
### Circuit Diagram


220 Ω
+5 V
74HC595
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Vcc
MR
DS
ST_CP
SH_CP
Q7'
OE
GND
to microcontroller dataPin
to microcontroller latchPin
to microcontroller clockPin
1 µf

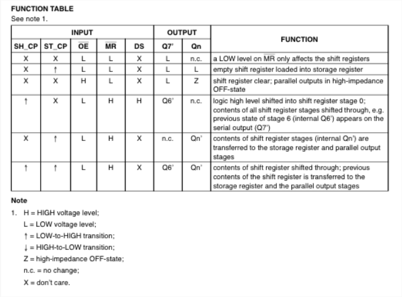
## The Code

Here are three code examples. The first is just some “hello world” code that simply outputs a byte value from 0 to 255. The second program lights one LED at a time. The third cycles through an array.

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.



595 Timing Diagram



**FUNCTION TABLE**

See note 1.

| INPUT | | | | | OUTPUT | | FUNCTION |
|---|---|---|---|---|---|---|---|
| SH_CP | ST_CP | $\overline{OE}$ | $\overline{MR}$ | DS | Q7' | Qn | |
| X | X | L | L | X | L | n.c. | a LOW level on $\overline{MR}$ only affects the shift registers |
| X | ↑ | L | L | X | L | L | empty shift register loaded into storage register |
| X | X | H | L | X | L | Z | shift register clear; parallel outputs in high-impedance OFF-state |
| ↑ | X | L | H | H | Q6' | n.c. | logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7') |
| X | ↑ | L | H | X | n.c. | Qn' | contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages |
| ↑ | ↑ | L | H | X | Q6' | Qn' | contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages |

**Note**

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table

Code Sample 1.1 – Hello World
Code Sample 1.2 – One by One
Code Sample 1.3 – from Defined Array

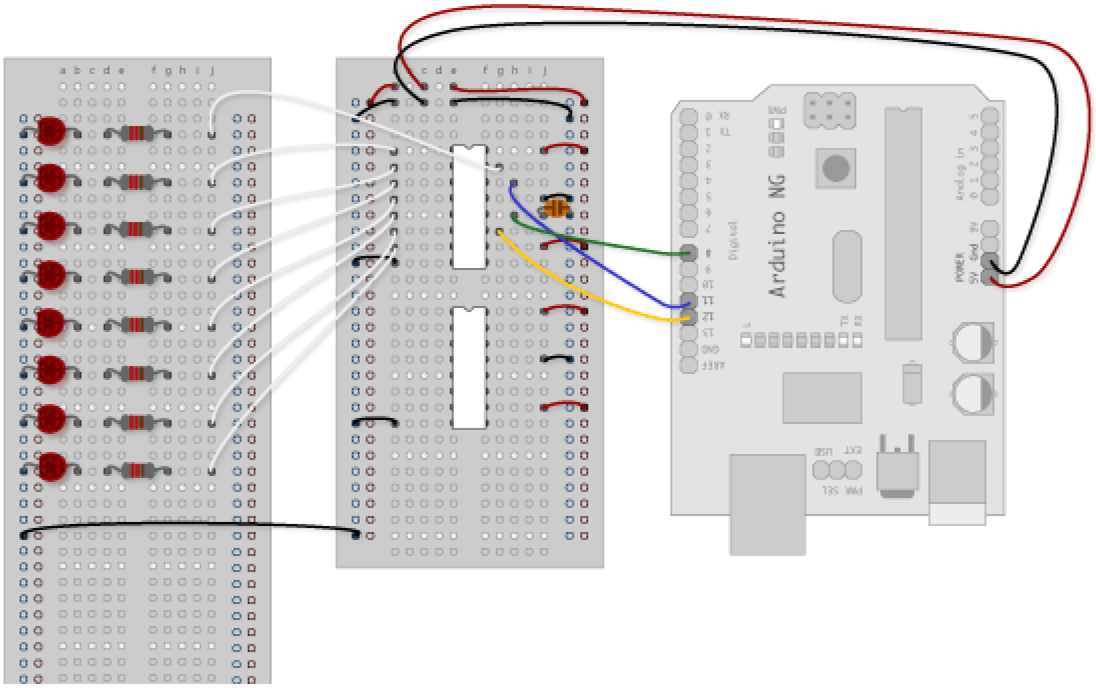
# Example 2

In this example you’ll add a second shift register, doubling the number of output pins you have while still using the same number of pins from the Arduino.

## The Circuit

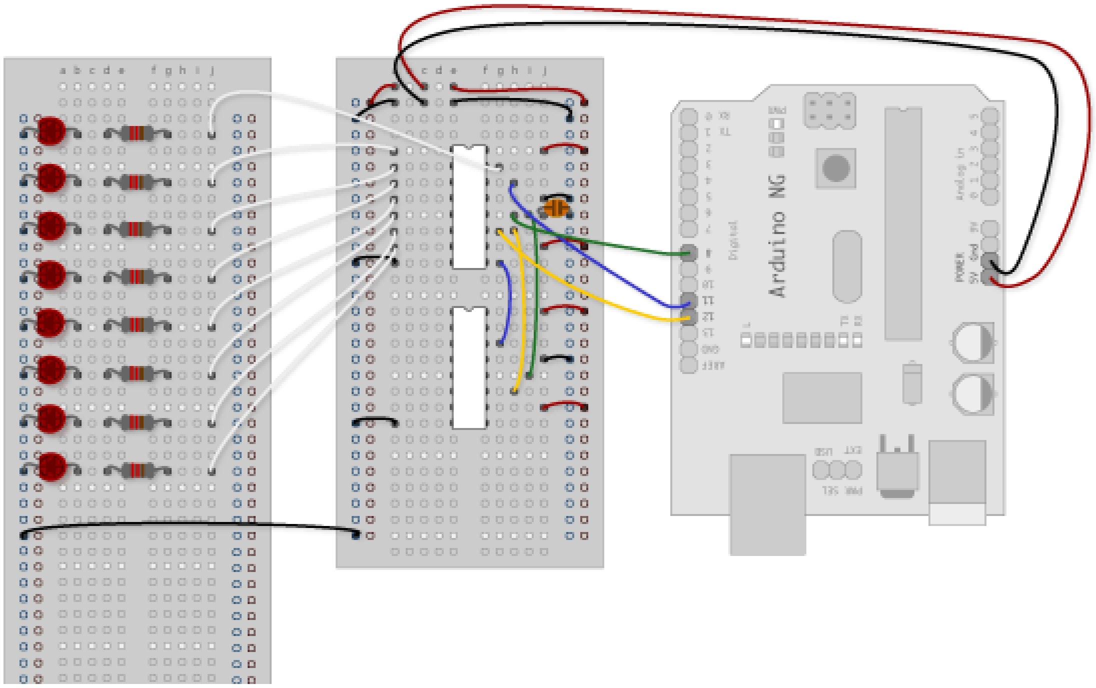
### 1. Add a second shift register.

Starting from the previous example, you should put a second shift register on the board. It should have the same leads to power and ground.



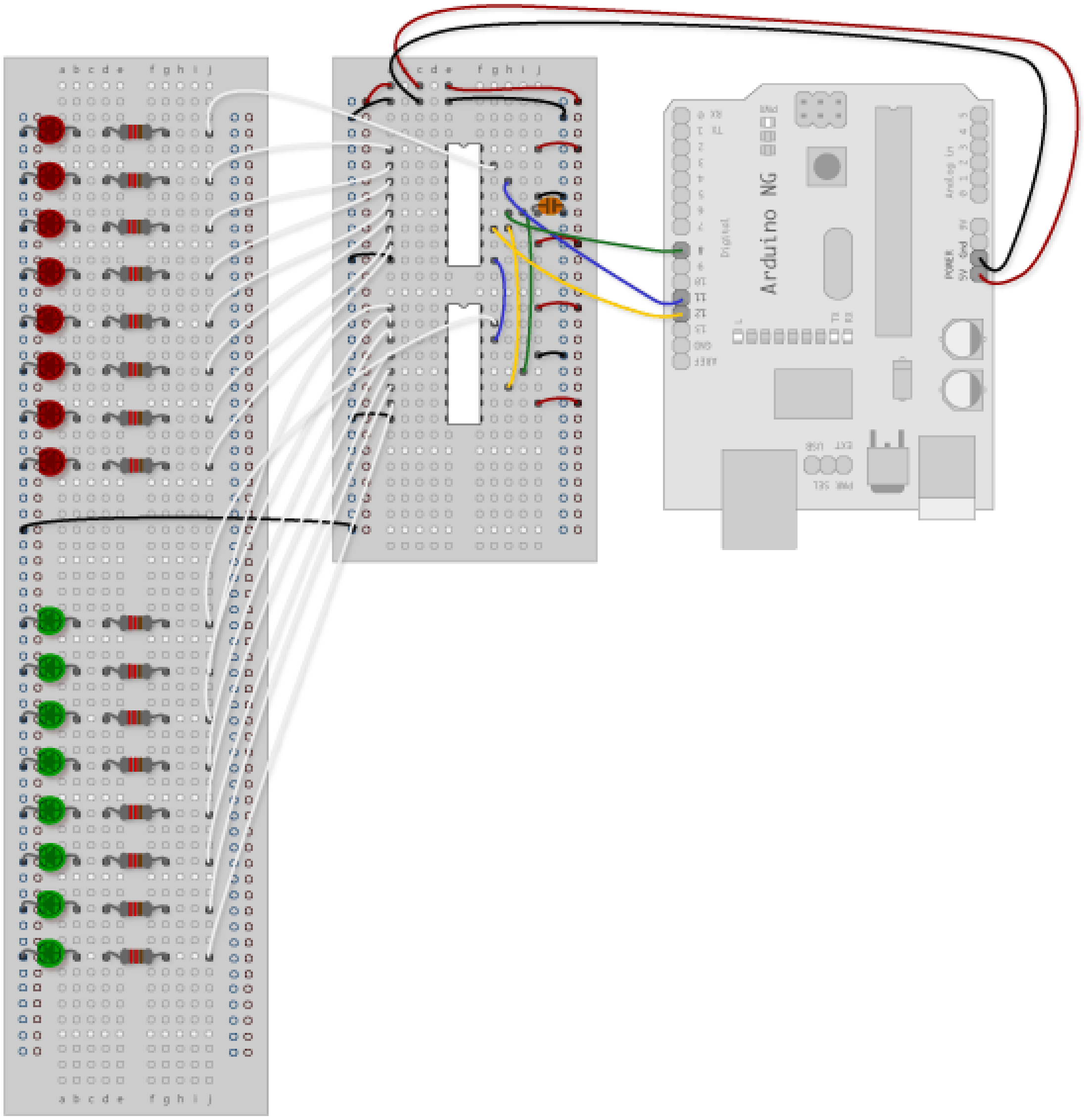
## 2. Connect the 2 registers.

Two of these connections simply extend the same clock and latch signal from the Arduino to the second shift register (yellow and green wires). The blue wire is going from the serial out pin (pin 9) of the first shift register to the serial data input (pin 14) of the second register.

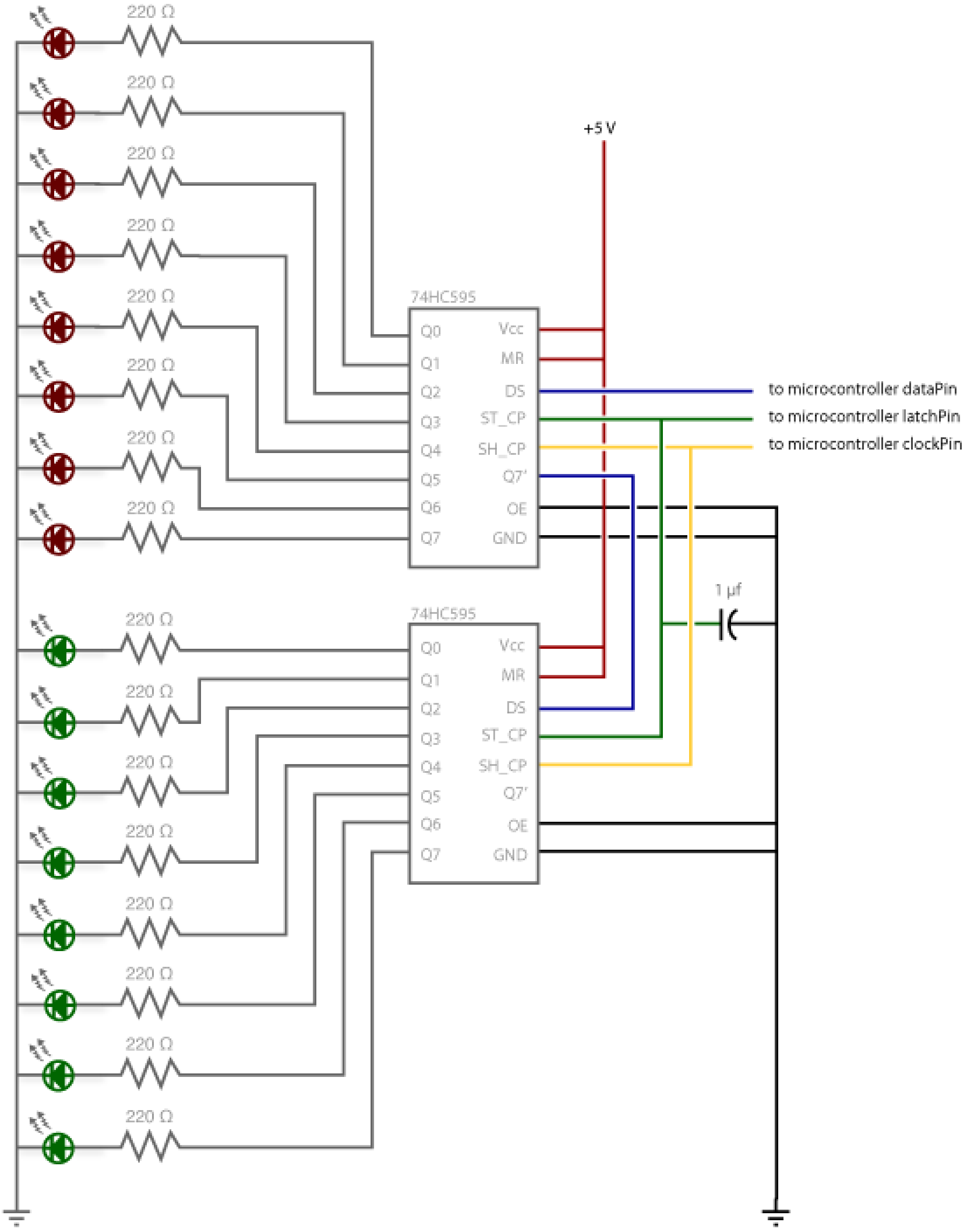


## 3. Add a second set of LEDs.

In this case I added green ones so when reading the code it is clear which byte is going to which set of LEDs



## Circuit Diagram


220 Ω
+5 V
74HC595
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Vcc
MR
DS
ST_CP
SH_CP
Q7'
OE
GND
to microcontroller dataPin
to microcontroller latchPin
to microcontroller clockPin
1 µf
74HC595

## The Code

Here again are three code samples. If you are curious, you might want to try the samples from the first example with this circuit set up just to see what happens.

Code Sample 2.1 – Dual Binary Counters
There is only one extra line of code compared to the first code sample from Example 1. It sends out a second byte. This forces the first shift register, the one directly attached to the Arduino, to pass the first byte sent through to the second register, lighting the green LEDs. The second byte will then show up on the red LEDs.

Code Sample 2.2 – 2 Byte One By One
Comparing this code to the similar code from Example 1 you see that a little bit more has had to change. The blinkAll() function has been changed to the blinkAll_2Bytes() function to reflect the fact that now there are 16 LEDs to control. Also, in version 1 the pulsings of the latchPin were situated inside the subfunctions lightShiftPinA and lightShiftPinB(). Here they need to be moved back into the main loop to accommodate needing to run each

subfunction twice in a row, once for the green LEDs and once for the red ones.

### Code Sample 2.3 - Dual Defined Arrays

Like sample 2.2, sample 2.3 also takes advantage of the new blinkAll_2bytes() function. 2.3's big difference from sample 1.3 is only that instead of just a single variable called “data” and a single array called “dataArray” you have to have a dataRED, a dataGREEN, dataArrayRED, dataArrayGREEN defined up front. This means that line

```
data = dataArray[j];
```

becomes

```
dataRED = dataArrayRED[j];
dataGREEN = dataArrayGREEN[j];
```

and

```
shiftOut(dataPin, clockPin, data);
```

becomes

```
shiftOut(dataPin, clockPin, dataGREEN);
shiftOut(dataPin, clockPin, dataRED);
```

(Printable View of http://www.arduino.cc/en/Tutorial/ShiftOut)

## Tutorial.X10 History

Hide minor edits - Show changes to markup

June 20, 2007, at 10:23 AM by Tom Igoe -
Changed lines 19-22 from:

**X10(int strLength)** - initialize an instance of the X10 library on two digital pins. e.g.

```
X10 myHouse = X10(9, 10); // initializes X10 on pins 9 (zero crossing pin) and 10 (data pin)
```

to:

**x10(int strLength)** - initialize an instance of the X10 library on two digital pins. e.g.

```
x10 myHouse = x10(9, 10); // initializes X10 on pins 9 (zero crossing pin) and 10 (data pin)
```

Restore
June 20, 2007, at 10:22 AM by Tom Igoe -
Added lines 29-34:

**version(void)** - get the library version. Since there will be more functions added, printing the version is a useful debugging tool when you get an error from a given function. Perhaps you're using an earlier version that doesn't feature the version you need! e.g.

```
Serial.println(myHouse.version());   // prints the version of the library
```

Deleted lines 56-61:

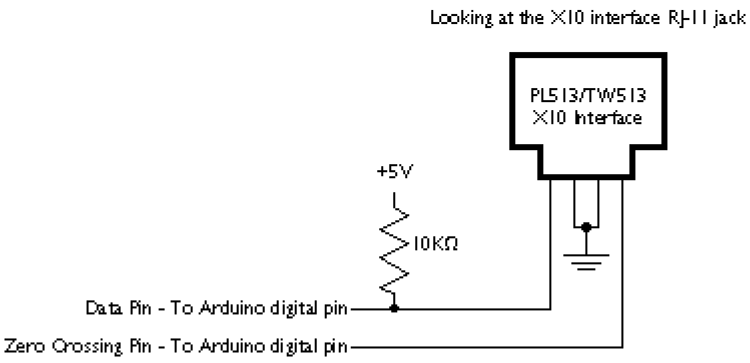
**version(void)** - get the library version. Since there will be more functions added, printing the version is a useful debugging tool when you get an error from a given function. Perhaps you're using an earlier version that doesn't feature the version you need! e.g.

```
Serial.println(myHouse.version());   // prints the version of the library
```

Restore
June 20, 2007, at 10:21 AM by Tom Igoe -
Changed lines 9-10 from:

Attach: X10-schematic.jpg

to:


Looking at the X10 interface RJ-11 jack
PL513/TW513
X10 Interface
+5V
10KΩ
Data Pin - To Arduino digital pin
Zero Crossing Pin - To Arduino digital pin

Restore

June 20, 2007, at 10:21 AM by Tom Igoe -
Changed lines 49-50 from:

For a full explanation of X10 and these codes, see this technote

to:

For a full explanation of X10 and these codes, see this technote

Restore
June 20, 2007, at 10:20 AM by Tom Igoe -
Changed lines 9-10 from:

Attach: X10.png

to:

Attach: X10-schematic.jpg

Restore
June 20, 2007, at 10:19 AM by Tom Igoe -
Changed lines 49-50 from:

For a full explanation of X10 and these codes, see

to:

For a full explanation of X10 and these codes, see this technote

Restore
June 20, 2007, at 10:18 AM by Tom Igoe -
Changed lines 3-4 from:

This library enables you to send and receive X10 commands from an Arduino module.

to:

This library enables you to send and receive X10 commands from an Arduino module. X10 is a synchronous serial protocol that travels over AC power lines, sending a bit every time the AC power crosses zero volts. It's used in home automation. You can find X10 controllers and devices at http://www.x10.com, http://www.smarthome.com, and more.

This library has been tested using the PL513 one-way X10 controller, and the TW523 two-way X10 controller. Both of these are essentially X10 modems, converting the 5V output of the Arduino into AC signals on the zero crossing.

To connect an Arduino to one of these modules, get a phone cable with an RJ-11 connector, and cut one end off. Then wire the pins as follows:

Attach: X10.png

Changed lines 49-50 from:
to:

For a full explanation of X10 and these codes, see

Restore
June 20, 2007, at 09:59 AM by Tom Igoe -
Added lines 1-52:

## X10 Library

This library enables you to send and receive X10 commands from an Arduino module.

Download: X10.zip

To use, unzip it and copy the resulting folder, called TextString, into the lib/targets/libraries directory of your arduino application folder. Then re-start the Arduino application.

When you restart, you'll see a few warning messages in the debugger pane at the bottom of the program. You can ignore them.

As of version 0.2, here's what you can do:

**X10(int strLength)** - initialize an instance of the X10 library on two digital pins. e.g.

```
X10 myHouse = X10(9, 10); // initializes X10 on pins 9 (zero crossing pin) and 10 (data pin)
```

**void write(byte houseCode, byte numberCode, int numRepeats)** - Send an X10 message, e.g.

```
myHouse.write(A, ALL_LIGHTS_ON, 1);     // Turns on all lights in house code A
```

There are a number of constants added to make X10 easier. They are as follows:

- A through F: house code values.
- UNIT_1 through UNIT_16: unit code values
- ALL_UNITS_OFF
- ALL_LIGHTS_ON
- ON
- OFF
- DIM
- BRIGHT
- ALL_LIGHTS_OFF
- EXTENDED_CODE
- HAIL_REQUEST
- HAIL_ACKNOWLEDGE
- PRE_SET_DIM
- EXTENDED_DATA
- STATUS_ON
- STATUS_OFF
- STATUS_REQUEST

**version(void)** - get the library version. Since there will be more functions added, printing the version is a useful debugging tool when you get an error from a given function. Perhaps you're using an earlier version that doesn't feature the version you need! e.g.

```
Serial.println(myHouse.version());   // prints the version of the library
```

---

If anyone's interested in helping to develop this library further, please contact me at tom.igoe at gmail.com

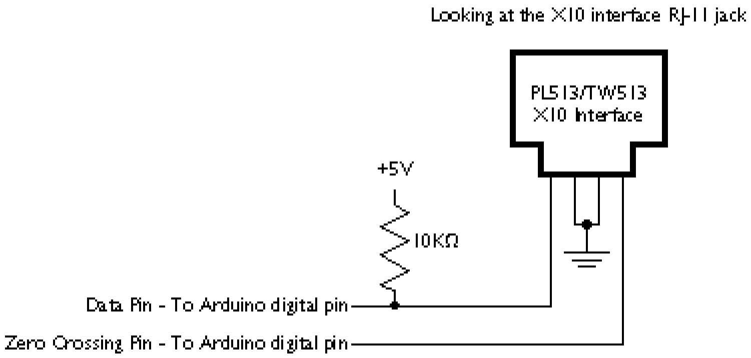
Restore

# X10 Library

This library enables you to send and receive X10 commands from an Arduino module. X10 is a synchronous serial protocol that travels over AC power lines, sending a bit every time the AC power crosses zero volts. It's used in home automation. You can find X10 controllers and devices at http://www.x10.com, http://www.smarthome.com, and more.

This library has been tested using the PL513 one-way X10 controller, and the TW523 two-way X10 controller. Both of these are essentially X10 modems, converting the 5V output of the Arduino into AC signals on the zero crossing.

To connect an Arduino to one of these modules, get a phone cable with an RJ-11 connector, and cut one end off. Then wire the pins as follows:


Looking at the X10 interface RJ-11 jack
PL513/TW513
X10 Interface
+5V
10KΩ
Data Pin - To Arduino digital pin
Zero Crossing Pin - To Arduino digital pin

Download: X10.zip

To use, unzip it and copy the resulting folder, called TextString, into the lib/targets/libraries directory of your arduino application folder. Then re-start the Arduino application.

When you restart, you'll see a few warning messages in the debugger pane at the bottom of the program. You can ignore them.

As of version 0.2, here's what you can do:

**x10(int strLength)** - initialize an instance of the X10 library on two digital pins. e.g.

```
x10 myHouse = x10(9, 10); // initializes X10 on pins 9 (zero crossing pin) and 10 (data pin)
```

**void write(byte houseCode, byte numberCode, int numRepeats)** - Send an X10 message, e.g.

```
myHouse.write(A, ALL_LIGHTS_ON, 1);     // Turns on all lights in house code A
```

**version(void)** - get the library version. Since there will be more functions added, printing the version is a useful debugging tool when you get an error from a given function. Perhaps you're using an earlier version that doesn't feature the version you need! e.g.

```
Serial.println(myHouse.version());   // prints the version of the library
```

There are a number of constants added to make X10 easier. They are as follows:

A through F: house code values.

- UNIT_1 through UNIT_16: unit code values
- ALL_UNITS_OFF
- ALL_LIGHTS_ON
- ON
- OFF
- DIM
- BRIGHT
- ALL_LIGHTS_OFF
- EXTENDED_CODE
- HAIL_REQUEST
- HAIL_ACKNOWLEDGE
- PRE_SET_DIM
- EXTENDED_DATA
- STATUS_ON
- STATUS_OFF
- STATUS_REQUEST

For a full explanation of X10 and these codes, see this technote

If anyone's interested in helping to develop this library further, please contact me at tom.igoe at gmail.com

## Tutorial.EEPROMClear History

Hide minor edits - Show changes to markup

May 21, 2008, at 09:32 PM by David A. Mellis -
Changed lines 29-31 from:

- Example: EEPROM Read
- Example: EEPROM Write
- Reference: EEPROM library

to:

- EEPROM Read example
- EEPROM Write example
- EEPROM library reference

Restore
May 21, 2008, at 09:32 PM by David A. Mellis -
Changed lines 25-31 from:

@]

to:

@]

### See also

- Example: EEPROM Read
- Example: EEPROM Write
- Reference: EEPROM library

Restore
May 21, 2008, at 09:27 PM by David A. Mellis -
Added lines 1-25:

*Examples > EEPROM Library*

## EEPROM Clear

Sets all of the bytes of the EEPROM to 0.

### Code

```
#include <EEPROM.h>

void setup()
{
  // write a 0 to all 512 bytes of the EEPROM
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, 0);

  // turn the LED on when we're done
  digitalWrite(13, HIGH);
}

void loop()
{
}
```

Restore

*Examples > EEPROM Library*

# EEPROM Clear

Sets all of the bytes of the EEPROM to 0.

## Code

```
#include <EEPROM.h>

void setup()
{
  // write a 0 to all 512 bytes of the EEPROM
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, 0);

  // turn the LED on when we're done
  digitalWrite(13, HIGH);
}

void loop()
{
}
```

## See also

- EEPROM Read example
- EEPROM Write example
- EEPROM library reference

## Tutorial.EEPROMRead History

Hide minor edits - Show changes to markup

May 21, 2008, at 09:33 PM by David A. Mellis -
Changed lines 41-47 from:

@]

to:

@]

### See also

- EEPROM Clear example
- EEPROM Write example
- EEPROM library reference

Restore
May 21, 2008, at 09:28 PM by David A. Mellis -
Added lines 1-41:

*Examples > EEPROM Library*

## EEPROM Read

Reads the value of each byte of the EEPROM and prints it to the computer.

### Code

```
#include <EEPROM.h>

// start reading from the first byte (address 0) of the EEPROM
int address = 0;
byte value;

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

void loop()
{
  // read a byte from the current address of the EEPROM
  value = EEPROM.read(address);

  Serial.print(address);
  Serial.print("\t");
  Serial.print(value, DEC);
  Serial.println();

  // advance to the next address of the EEPROM
  address = address + 1;

  // there are only 512 bytes of EEPROM, from 0 to 511, so if we're
  // on address 512, wrap around to address 0
  if (address == 512)
```

```
    address = 0;

  delay(500);
}
```

Restore

*Examples > EEPROM Library*

# EEPROM Read

Reads the value of each byte of the EEPROM and prints it to the computer.

## Code

```
#include <EEPROM.h>

// start reading from the first byte (address 0) of the EEPROM
int address = 0;
byte value;

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

void loop()
{
  // read a byte from the current address of the EEPROM
  value = EEPROM.read(address);

  Serial.print(address);
  Serial.print("\t");
  Serial.print(value, DEC);
  Serial.println();

  // advance to the next address of the EEPROM
  address = address + 1;

  // there are only 512 bytes of EEPROM, from 0 to 511, so if we're
  // on address 512, wrap around to address 0
  if (address == 512)
    address = 0;

  delay(500);
}
```

## See also

- EEPROM Clear example
- EEPROM Write example
- EEPROM library reference

## Tutorial.EEPROMWrite History

Hide minor edits - Show changes to markup

May 21, 2008, at 09:33 PM by David A. Mellis -
Changed lines 40-46 from:

@]

to:

@]

### See also

- EEPROM Clear example
- EEPROM Read example
- EEPROM library reference

Restore
May 21, 2008, at 09:30 PM by David A. Mellis -
Added lines 1-40:

*Examples > EEPROM Library*

## EEPROM Write

Stores values read from analog input 0 into the EEPROM. These values will stay in the EEPROM when the board is turned off and may be retrieved later by another sketch.

### Code

```
#include <EEPROM.h>

// the current address in the EEPROM (i.e. which byte
// we're going to write to next)
int addr = 0;

void setup()
{
}

void loop()
{
  // need to divide by 4 because analog inputs range from
  // 0 to 1023 and each byte of the EEPROM can only hold a
  // value from 0 to 255.
  int val = analogRead(0) / 4;

  // write the value to the appropriate byte of the EEPROM.
  // these values will remain there when the board is
  // turned off.
  EEPROM.write(addr, val);

  // advance to the next address.  there are 512 bytes in
  // the EEPROM, so go back to 0 when we hit 512.
  addr = addr + 1;
  if (addr == 512)
```

```
    addr = 0;

  delay(100);
}
```

Restore

Edit Page | Page History | Printable View | All Recent Site Changes

*Examples > EEPROM Library*

# EEPROM Write

Stores values read from analog input 0 into the EEPROM. These values will stay in the EEPROM when the board is turned off and may be retrieved later by another sketch.

## Code

```
#include <EEPROM.h>

// the current address in the EEPROM (i.e. which byte
// we're going to write to next)
int addr = 0;

void setup()
{
}

void loop()
{
  // need to divide by 4 because analog inputs range from
  // 0 to 1023 and each byte of the EEPROM can only hold a
  // value from 0 to 255.
  int val = analogRead(0) / 4;

  // write the value to the appropriate byte of the EEPROM.
  // these values will remain there when the board is
  // turned off.
  EEPROM.write(addr, val);

  // advance to the next address.  there are 512 bytes in
  // the EEPROM, so go back to 0 when we hit 512.
  addr = addr + 1;
  if (addr == 512)
    addr = 0;

  delay(100);
}
```

## See also

- EEPROM Clear example
- EEPROM Read example
- EEPROM library reference

## Tutorial.MotorKnob History

Hide minor edits - Show changes to markup

May 21, 2008, at 09:40 PM by David A. Mellis -
Changed lines 45-46 from:

**See Also**

to:

**See also**

Restore
May 21, 2008, at 09:40 PM by David A. Mellis -
Changed lines 43-47 from:

@]

to:

@]

**See Also**

- Stepper library reference

Restore
May 21, 2008, at 09:37 PM by David A. Mellis -
Changed lines 7-8 from:

A stepper motor follows the turns of a potentiometer (or other sensor) on analog input 0. The unipolar? or bipolar? stepper is controlled with pins 8, 9, 10, and 11, using one of the circuits on the linked pages.

to:

A stepper motor follows the turns of a potentiometer (or other sensor) on analog input 0. The unipolar or bipolar stepper is controlled with pins 8, 9, 10, and 11, using one of the circuits on the linked pages.

Restore
May 21, 2008, at 09:36 PM by David A. Mellis -
Added lines 1-43:

*Examples > Stepper Library*

## Motor Knob

**Description**

A stepper motor follows the turns of a potentiometer (or other sensor) on analog input 0. The unipolar? or bipolar? stepper is controlled with pins 8, 9, 10, and 11, using one of the circuits on the linked pages.

**Code**

```
#include <Stepper.h>

// change this to the number of steps on your motor
#define STEPS 100
```

```
// create an instance of the stepper class, specifying
// the number of steps of the motor and the pins it's
// attached to
Stepper stepper(STEPS, 8, 9, 10, 11);

// the previous reading from the analog input
int previous = 0;

void setup()
{
  // set the speed of the motor to 30 RPMs
  stepper.setSpeed(30);
}

void loop()
{
  // get the sensor value
  int val = analogRead(0);

  // move a number of steps equal to the change in the
  // sensor reading
  stepper.step(val - previous);

  // remember the previous value of the sensor
  previous = val;
}
```

Restore

**Learning**   Examples | Foundations | Hacking | Links

*Examples > Stepper Library*

# Motor Knob

### Description

A stepper motor follows the turns of a potentiometer (or other sensor) on analog input 0. The unipolar or bipolar stepper is controlled with pins 8, 9, 10, and 11, using one of the circuits on the linked pages.

### Code

```
#include <Stepper.h>

// change this to the number of steps on your motor
#define STEPS 100

// create an instance of the stepper class, specifying
// the number of steps of the motor and the pins it's
// attached to
Stepper stepper(STEPS, 8, 9, 10, 11);

// the previous reading from the analog input
int previous = 0;

void setup()
{
  // set the speed of the motor to 30 RPMs
  stepper.setSpeed(30);
}

void loop()
{
  // get the sensor value
  int val = analogRead(0);

  // move a number of steps equal to the change in the
  // sensor reading
  stepper.step(val - previous);

  // remember the previous value of the sensor
  previous = val;
}
```

### See also

- Stepper library reference

## Tutorial.HomePage History

Show minor edits - Show changes to markup

July 02, 2008, at 03:11 PM by David A. Mellis -
Changed lines 2-3 from:

## Arduino Examples

to:

## Examples

Restore
July 02, 2008, at 03:11 PM by David A. Mellis -
Changed lines 4-5 from:

*See the **foundations page** for in-depth description of core concepts of the Arduino hardware and software, and the **links page** for other documentation.*

to:

*See the **foundations page** for in-depth description of core concepts of the Arduino hardware and software; the **hacking page** for information on extending and modifying the Arduino hardware and software; and the **links page** for other documentation.*

Restore
July 02, 2008, at 02:07 PM by David A. Mellis -
Added line 63:

- Read an ADXL3xx accelerometer

Restore
May 21, 2008, at 09:44 PM by David A. Mellis -
Deleted lines 42-45:

**Matrix Library**

- Hello Matrix?: blinks a smiley face on the LED matrix.

Restore
May 21, 2008, at 09:43 PM by David A. Mellis -
Added lines 43-46:

**Matrix Library**

- Hello Matrix?: blinks a smiley face on the LED matrix.

Restore
May 21, 2008, at 09:36 PM by David A. Mellis -
Added lines 43-46:

**Stepper Library**

- Motor Knob: control a stepper motor with a potentiometer.

Restore
May 21, 2008, at 09:25 PM by David A. Mellis - adding EEPROM examples.
Added lines 37-42:

**EEPROM Library**

- EEPROM Clear: clear the bytes in the EEPROM.
- EEPROM Read: read the EEPROM and send its values to the computer.
- EEPROM Write: stores values from an analog input to the EEPROM.

Restore
May 21, 2008, at 09:22 PM by David A. Mellis -
Changed line 15 from:

- BlinkWithoutDelay: blinking an LED without using the delay() function.

to:

- Blink Without Delay: blinking an LED without using the delay() function.

Restore
April 29, 2008, at 06:55 PM by David A. Mellis - moving the resources to the links page.
Changed lines 2-5 from:

## Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino Getting Started.

to:

## Arduino Examples

*See the **foundations page** for in-depth description of core concepts of the Arduino hardware and software, and the **links page** for other documentation.*

Added line 15:

- BlinkWithoutDelay: blinking an LED without using the delay() function.

Changed lines 37-42 from:

### Timing & Millis

- Blinking an LED without using the delay() function
- Stopwatch

(:if false:)

- TimeSinceStart:

(:ifend:)

to:

(:cell width=50%:)

Changed lines 41-42 from:

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

to:

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included on the page.

Deleted lines 43-44:
Added lines 49-51:

### Timing & Millis

- Stopwatch

Deleted lines 75-125:

(:cell width=50%:)

### Foundations

See the foundations page for explanations of the concepts involved in the Arduino hardware and software.

### Tutorials

Tutorials created by the Arduino community. Hosted on the publicly-editable playground wiki.

Board Setup and Configuration: Information about the components and usage of Arduino hardware.

Interfacing With Hardware: Code, circuits, and instructions for using various electronic components with an Arduino board.

- Output
- Input
- Interaction
- Storage
- Communication

Interfacing with Software: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

Code Library and Tutorials: Arduino functions for performing specific tasks and other programming tutorials.

Electronics Techniques: tutorials on soldering and other electronics resources.

### Manuals, Curricula, and Other Resources

Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Examples from Tom Igoe

Examples from Jeff Gray

Restore
April 23, 2008, at 10:29 PM by David A. Mellis -
Changed line 6 from:

(:table width=90% border=0 cellpadding=5 cellspacing=0:)

to:

(:table width=100% border=0 cellpadding=5 cellspacing=0:)

Restore
April 22, 2008, at 05:59 PM by Paul Badger -
Changed line 39 from:
to:

(:if false:)

Changed line 41 from:
to:

(:ifend:)

Restore
April 22, 2008, at 05:56 PM by Paul Badger -
Added lines 40-41:

- TimeSinceStart:

Restore
April 18, 2008, at 07:22 AM by Paul Badger -
Added lines 36-39:

### Timing & Millis

- Blinking an LED without using the delay() function
- Stopwatch

Changed line 46 from:

- Blinking an LED without using the delay() function

to:
Restore
April 08, 2008, at 08:22 PM by David A. Mellis - moving TwoSwitchesOnePin to "other examples" since it's not (yet) in the distribution.
Changed lines 18-19 from:

- TwoSwitchesOnePin: Read two switches with one I/O pin

to:
Added line 43:

- * TwoSwitchesOnePin: Read two switches with one I/O pin

Restore
April 08, 2008, at 07:41 PM by Paul Badger -
Changed lines 18-19 from:
to:

- TwoSwitchesOnePin: Read two switches with one I/O pin

Restore
March 09, 2008, at 07:20 PM by David A. Mellis -
Changed lines 73-78 from:

- Foundations has moved here

- Bootloader: A small program pre-loaded on the Arduino board to allow uploading sketches.

to:

See the foundations page for explanations of the concepts involved in the Arduino hardware and software.

Restore
March 07, 2008, at 09:26 PM by Paul Badger -
Changed lines 73-75 from:

to:

- Foundations has moved here

Restore
March 07, 2008, at 09:24 PM by Paul Badger -
Changed lines 74-107 from:

- Memory: The various types of memory available on the Arduino board.

- Digital Pins: How the pins work and what it means for them to be configured as inputs or outputs.

- Analog Input Pins: Details about the analog-to-digital conversion and other uses of the pins.
- Foundations

(:if false:)

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.
- Communication?: An overview of the various ways in which an Arduino board can communicate with other devices (serial, I2C, SPI, Midi, etc.)
- Serial Communication?: How to send serial data from an Arduino board to a computer or other device (including via the USB connection).
- Interrupts?: Code that interrupts other code under certain conditions.
- Numbers?: The various types of numbers available and how to use them.
- Variables: How to define and use variables.
- Arrays?: How to store multiple values of the same type.
- Pointers?:
- Functions?: How to write and call functions.
- Optimization?: What to do when your program runs too slowly.
- Debugging?: Figuring out what's wrong with your hardware or software and how to fix it.

(:ifend:)

to:
Restore
March 07, 2008, at 09:09 PM by Paul Badger -
Added lines 80-81:

- Foundations

Restore
February 15, 2008, at 06:00 PM by David A. Mellis -
Changed lines 72-73 from:

### Tutorials

to:

### Foundations

Changed lines 108-109 from:

### More Tutorials

to:

### Tutorials

Restore
February 13, 2008, at 10:42 PM by Paul Badger -
Changed lines 4-5 from:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide.

to:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino Getting Started.

Restore
February 13, 2008, at 10:06 PM by David A. Mellis -

Restore
February 13, 2008, at 09:58 PM by David A. Mellis -
Added lines 100-103:

- Optimization?: What to do when your program runs too slowly.
- Debugging?: Figuring out what's wrong with your hardware or software and how to fix it.

Restore
February 13, 2008, at 09:41 PM by David A. Mellis -
Added lines 90-99:

- Numbers?: The various types of numbers available and how to use them.
- Variables: How to define and use variables.
- Arrays?: How to store multiple values of the same type.
- Pointers?:
- Functions?: How to write and call functions.

Restore
February 13, 2008, at 09:38 PM by David A. Mellis -
Changed lines 86-87 from:

- Serial Communication?: How to send serial data from an Arduino board to a computer or other device.

to:

- Serial Communication?: How to send serial data from an Arduino board to a computer or other device (including via the USB connection).
- Interrupts?: Code that interrupts other code under certain conditions.

Restore
February 13, 2008, at 09:36 PM by David A. Mellis -
Added lines 80-81:

(:if false:)

Added lines 84-89:

- Communication?: An overview of the various ways in which an Arduino board can communicate with other devices (serial, I2C, SPI, Midi, etc.)
- Serial Communication?: How to send serial data from an Arduino board to a computer or other device.

(:ifend:)

Restore
February 13, 2008, at 09:31 PM by David A. Mellis -
Changed lines 80-81 from:

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.

to:

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.

Restore
February 13, 2008, at 09:30 PM by David A. Mellis -
Added lines 80-81:

- PWM (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.

Restore
February 13, 2008, at 09:22 PM by David A. Mellis -
Added lines 80-81:

- Bootloader: A small program pre-loaded on the Arduino board to allow uploading sketches.

Restore
February 13, 2008, at 09:12 PM by David A. Mellis -

Added lines 74-81:

- Memory: The various types of memory available on the Arduino board.
- Digital Pins: How the pins work and what it means for them to be configured as inputs or outputs.
- Analog Input Pins: Details about the analog-to-digital conversion and other uses of the pins.

### More Tutorials

Restore
January 11, 2008, at 11:31 AM by David A. Mellis - linking to board setup and configuration on the playground.
Added lines 76-77:

Board Setup and Configuration: Information about the components and usage of Arduino hardware.

Restore
December 19, 2007, at 11:54 PM by David A. Mellis - adding links to other pages: the tutorial parts of the playground, ladyada's tutorials, todbot, etc.
Changed lines 36-42 from:

(:cell width=50%:)

### Tutorials

These are more complex tutorials for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

to:

### Other Examples

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

Changed lines 71-78 from:

#### Other Arduino Tutorials

- Tutorials from the Arduino playground
- Example labs from ITP
- Spooky Arduino and more from Todbot
- Examples from Tom Igoe
- Examples from Jeff Gray

to:

(:cell width=50%:)

### Tutorials

Tutorials created by the Arduino community. Hosted on the publicly-editable playground wiki.

Interfacing With Hardware: Code, circuits, and instructions for using various electronic components with an Arduino board.

- Output
- Input
- Interaction
- Storage
- Communication

Interfacing with Software: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

Code Library and Tutorials: Arduino functions for performing specific tasks and other programming tutorials.

Electronics Techniques: tutorials on soldering and other electronics resources.

### Manuals, Curricula, and Other Resources

Arduino Booklet (pdf): an illustrated guide to the philosophy and practice of Arduino.

Learn electronics using Arduino: an introduction to programming, input / output, communication, etc. using Arduino. By ladyada.

- Lesson 0: Pre-flight check...Is your Arduino and computer ready?
- Lesson 1: The "Hello World!" of electronics, a simple blinking light
- Lesson 2: Sketches, variables, procedures and hacking code
- Lesson 3: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
- Lesson 4: The serial library and binary data - getting chatty with Arduino and crunching numbers
- Lesson 5: Buttons & switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

Example labs from ITP

Spooky Arduino: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:

- class 1 (getting started)
- class 2 (input and sensors)
- class 3 (communication, servos, and pwm)
- class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Bionic Arduino: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

Examples from Tom Igoe

Examples from Jeff Gray

Restore
December 13, 2007, at 11:08 PM by David A. Mellis - adding debounce example.
Added line 16:

- Debounce: read a pushbutton, filtering noise.

Restore
August 28, 2007, at 11:15 PM by Tom Igoe -
Changed lines 71-72 from:
to:

- X10 output control devices over AC powerlines using X10

Restore
June 15, 2007, at 05:04 PM by David A. Mellis - adding link to Processing (for the communication examples)
Added lines 27-28:

*These examples include code that allows the Arduino to talk to Processing sketches running on the computer. For more information or to download Processing, see processing.org.*

Restore
June 12, 2007, at 08:57 AM by David A. Mellis - removing link to obsolete joystick example.
Deleted line 43:

- Interfacing a Joystick

Restore
June 11, 2007, at 11:14 PM by David A. Mellis -
Changed lines 10-11 from:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.)

Restore
June 11, 2007, at 11:13 PM by David A. Mellis -
Changed lines 10-11 from:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder.

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder. (If you're looking for an older example, check the Arduino 0007 tutorials page.

Restore
June 11, 2007, at 11:10 PM by David A. Mellis - updating to 0008 examples
Changed lines 10-11 from:

### Digital Output

- Blinking LED

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the **examples** folder.

### Digital I/O

- Blink: turn an LED on and off.
- Button: use a pushbutton to control an LED.
- Loop: controlling multiple LEDs with a loop and an array.

### Analog I/O

- Analog Input: use a potentiometer to control the blinking of an LED.
- Fading: uses an analog output (PWM pin) to fade an LED.
- Knock: detect knocks with a piezo element.
- Smoothing: smooth multiple readings of an analog input.

### Communication

- ASCII Table: demonstrates Arduino's advanced serial output functions.
- Dimmer: move the mouse to change the brightness of an LED.
- Graph: sending data to the computer and graphing it in Processing.
- Physical Pixel: turning on and off an LED by sending data from Processing.
- Virtual Color Mixer: sending multiple variables from Arduino to the computer and reading them in Processing.

(:cell width=50%:)

## Tutorials

These are more complex tutorials for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

### Miscellaneous

Deleted lines 42-51:

- Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- More complex dimming/color crossfader
- Knight Rider example
- Shooting star
- PWM all of the digital pins in a sinewave pattern

### Digital Input

- Digital Input and Output (from ITP physcomp labs)
- Read a Pushbutton
- Using a pushbutton as a switch

Deleted lines 43-45:

### Analog Input

- Read a Potentiometer

Deleted lines 45-46:

- Read a Piezo Sensor
- 3 LED cross-fades with a potentiometer

Changed lines 52-53 from:

- Use two Arduino pins as a capacitive sensor

to:
Deleted line 54:

- More sound ideas

Added line 64:

- Build your own DMX Master device

Changed lines 70-72 from:

- Multiple digital inputs with a CD4021 Shift Register

## Other Arduino Examples

to:

### Other Arduino Tutorials

- Tutorials from the Arduino playground

Added line 75:

- Spooky Arduino and more from Todbot

Deleted lines 78-105:

(:cell width=50%:)

## Interfacing with Other Software

- Introduction to Serial Communication (from ITP physcomp labs)
- Arduino + Flash
- Arduino + Processing
- Arduino + PD
- Arduino + MaxMSP
- Arduino + VVVV
- Arduino + Director
- Arduino + Ruby
- Arduino + C

## Tech Notes (from the forums or playground)

- Software serial (serial on pins besides 0 and 1)
- L297 motor driver
- Hex inverter
- Analog multiplexer
- Power supplies
- The components on the Arduino board
- Arduino build process
- AVRISP mkII on the Mac
- Non-volatile memory (EEPROM)
- Bluetooth
- Zigbee
- LED as light sensor (en Francais)
- Arduino and the Asuro robot
- Using Arduino from the command line

Restore
May 11, 2007, at 06:06 AM by Paul Badger -
Changed lines 17-18 from:
to:

- PWM all of the digital pins in a sinewave pattern

Restore
May 10, 2007, at 07:07 PM by Paul Badger -
Changed lines 36-37 from:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 |Use a couple of Arduino pins as a capacitive sensor]]

to:

- Use two Arduino pins as a capacitive sensor

Restore
May 10, 2007, at 07:05 PM by Paul Badger -
Changed lines 36-37 from:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 Use a couple of Arduino pins as a capacitive sensor

to:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 |Use a couple of Arduino pins as a capacitive sensor]]

Restore
May 10, 2007, at 07:04 PM by Paul Badger -
Changed lines 36-37 from:
to:

- http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 Use a couple of Arduino pins as a capacitive sensor

Restore
May 10, 2007, at 06:59 PM by Paul Badger -
Added line 39:

- More sound ideas

Restore
April 24, 2007, at 03:40 PM by Clay Shirky -
Changed lines 13-14 from:

- Dimming 3 LEDs with Pulse-Width Modulation (PWM)

to:

- Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- More complex dimming/color crossfader

Restore
February 08, 2007, at 12:02 PM by Carlyn Maw -
Changed lines 52-53 from:
to:

- Multiple digital inputs with a CD4021 Shift Register

Restore
February 06, 2007, at 02:52 PM by Carlyn Maw -
Changed lines 52-54 from:

- Multiple digital ins with a CD4021 Shift Register

to:
Restore
February 06, 2007, at 02:51 PM by Carlyn Maw -
Changed lines 52-53 from:
to:

- Multiple digital ins with a CD4021 Shift Register

Restore
January 30, 2007, at 03:37 PM by David A. Mellis -
Deleted line 46:

- Build your own DMX Master device

Restore
December 25, 2006, at 11:57 PM by David A. Mellis -
Added line 20:

- Using a pushbutton as a switch

Restore
December 07, 2006, at 06:04 AM by David A. Mellis - adding link to todbot's C serial port code.
Changed lines 69-70 from:
to:

- Arduino + C

Restore
December 02, 2006, at 10:43 AM by David A. Mellis -
Added line 1:

(:title Tutorials:)

Restore
November 21, 2006, at 10:13 AM by David A. Mellis -
Added line 64:

- Arduino + MaxMSP

Changed lines 67-68 from:
to:

- Arduino + Ruby

Restore
November 18, 2006, at 02:42 AM by David A. Mellis -
Changed lines 20-21 from:

- Controlling an LED circle with a joystick

to:
Added line 24:

- Controlling an LED circle with a joystick

Restore
November 09, 2006, at 03:10 PM by Carlyn Maw -
Changed lines 50-51 from:
to:

- Multiple digital outs with a 595 Shift Register

Restore
November 06, 2006, at 10:49 AM by David A. Mellis -
Changed lines 37-38 from:

- MIDI Output (from ITP physcomp labs)

to:

- MIDI Output (from ITP physcomp labs) and from Spooky Arduino

Restore
November 04, 2006, at 12:25 PM by David A. Mellis -
Deleted line 53:
Deleted line 54:
Restore
November 04, 2006, at 12:24 PM by David A. Mellis -
Added lines 51-58:

### Other Arduino Examples

- Example labs from ITP

- Examples from Tom Igoe
- Examples from Jeff Gray

Deleted lines 83-89:

## Other Arduino Examples

- Example labs from ITP
- Examples from Tom Igoe.
- Examples from Jeff Gray.

Restore
November 04, 2006, at 12:24 PM by David A. Mellis -
Changed lines 50-51 from:

### Example labs from ITP

to:
Changed lines 77-78 from:

Also, see the examples from Tom Igoe and those from Jeff Gray.

to:

- Example labs from ITP
- Examples from Tom Igoe.
- Examples from Jeff Gray.

Restore
November 04, 2006, at 12:23 PM by David A. Mellis -
Changed line 77 from:

## Other Arduino Sites

to:

## Other Arduino Examples

Deleted lines 79-81:

## Do you need extra help?

Is there a sensor you would like to see characterized for Arduino, or is there something you would like to see published in this site? Refer to the forum for further help.

Restore
November 04, 2006, at 10:38 AM by David A. Mellis -
Changed lines 3-4 from:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide?.

to:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide.

Restore
November 04, 2006, at 10:37 AM by David A. Mellis - lots of content moved to the new guide.
Deleted lines 52-67:

## The Arduino board

This guide to the Arduino board explains the functions of the various parts of the board.

## The Arduino environment

This guide to the Arduino IDE (integrated development environment) explains the functions of the various buttons and menus.

The libraries page explains how to use libraries in your sketches and how to make your own.

### Video Lectures by Tom Igoe

Watch Tom introduce Arduino. Thanks to Pollie Barden for the great videos.

### Course Guides

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm), class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Deleted lines 82-87:

### External Resources

Instant Soup is an introduction to electronics through a series of beautifully-documented fun projects.

Make magazine has some great links in its electronics archive.

hack a day has links to interesting hacks and how-to articles on various topics.

Restore
November 04, 2006, at 10:17 AM by David A. Mellis -
Changed lines 3-4 from:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Howto.

to:

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino guide?.

Restore
November 01, 2006, at 06:54 PM by Carlyn Maw -
Deleted line 49:

- Extend your digital outs with 74HC595 shift registers

Restore
November 01, 2006, at 06:06 PM by Carlyn Maw -
Added line 50:

- Extend your digital outs with 74HC595 shift registers

Restore
October 31, 2006, at 10:47 AM by Tod E. Kurt -
Changed lines 67-68 from:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm).

to:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm), class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)

Restore
October 22, 2006, at 12:52 PM by David A. Mellis -
Changed lines 1-4 from:

## Learning to use Arduino

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto.

to:

# Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Howto.

Restore
October 22, 2006, at 12:51 PM by David A. Mellis -
Changed lines 67-68 from:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors).

to:

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors), class 3 (communication, servos, and pwm).

Restore
October 21, 2006, at 04:25 PM by David A. Mellis - adding links to todbot's class notes.
Added lines 66-68:

### Course Guides

todbot has some very detailed, illustrated tutorials from his Spooky Projects course: class 1 (getting started), class 2 (input and sensors).

Restore
October 08, 2006, at 05:46 PM by David A. Mellis -
Changed lines 59-62 from:

This guide to the Arduino IDE? (integrated development environment) explains the functions of the various buttons and menus.

The libraries? page explains how to use libraries in your sketches and how to make your own.

to:

This guide to the Arduino IDE (integrated development environment) explains the functions of the various buttons and menus.

The libraries page explains how to use libraries in your sketches and how to make your own.

Restore
October 08, 2006, at 05:45 PM by David A. Mellis -
Changed lines 3-4 from:

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto?.

to:

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto.

Restore
October 08, 2006, at 05:38 PM by David A. Mellis -
Added lines 1-102:

# Learning to use Arduino

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the Howto?.

(:table width=90% border=0 cellpadding=5 cellspacing=0:) (:cell width=50%:)

### Examples

#### Digital Output

- Blinking LED

- Blinking an LED without using the delay() function
- Dimming 3 LEDs with Pulse-Width Modulation (PWM)
- Knight Rider example
- Shooting star

**Digital Input**

- Digital Input and Output (from ITP physcomp labs)
- Read a Pushbutton
- Read a Tilt Sensor
- Controlling an LED circle with a joystick

**Analog Input**

- Read a Potentiometer
- Interfacing a Joystick
- Read a Piezo Sensor
- 3 LED cross-fades with a potentiometer
- 3 LED color mixer with 3 potentiometers

**Complex Sensors**

- Read an Accelerometer
- Read an Ultrasonic Range Finder (ultrasound sensor)
- Reading the qprox qt401 linear touch sensor

**Sound**

- Play Melodies with a Piezo Speaker
- Play Tones from the Serial Connection
- MIDI Output (from ITP physcomp labs)

**Interfacing w/ Hardware**

- Multiply the Amount of Outputs with an LED Driver
- Interfacing an LCD display with 8 bits
    - LCD interface library
- Driving a DC Motor with an L293 (from ITP physcomp labs).
- Driving a Unipolar Stepper Motor
- Build your own DMX Master device
- Implement a software serial connection
    - RS-232 computer interface
- Interface with a serial EEPROM using SPI
- Control a digital potentiometer using SPI

**Example labs from ITP**

(:cell width=50%:)

### The Arduino board

This guide to the Arduino board explains the functions of the various parts of the board.

### The Arduino environment

This guide to the Arduino IDE? (integrated development environment) explains the functions of the various buttons and menus.

The libraries? page explains how to use libraries in your sketches and how to make your own.

### Video Lectures by Tom Igoe

Watch Tom introduce Arduino. Thanks to Pollie Barden for the great videos.

### Interfacing with Other Software

- Introduction to Serial Communication (from ITP physcomp labs)
- Arduino + Flash
- Arduino + Processing

- Arduino + PD
- Arduino + VVVV
- Arduino + Director

## Tech Notes (from the forums or playground)

- Software serial (serial on pins besides 0 and 1)
- L297 motor driver
- Hex inverter
- Analog multiplexer
- Power supplies
- The components on the Arduino board
- Arduino build process
- AVRISP mkII on the Mac
- Non-volatile memory (EEPROM)
- Bluetooth
- Zigbee
- LED as light sensor (en Francais)
- Arduino and the Asuro robot
- Using Arduino from the command line

## Other Arduino Sites

Also, see the examples from Tom Igoe and those from Jeff Gray.

## Do you need extra help?

Is there a sensor you would like to see characterized for Arduino, or is there something you would like to see published in this site? Refer to the forum for further help.

## External Resources

Instant Soup is an introduction to electronics through a series of beautifully-documented fun projects.

Make magazine has some great links in its electronics archive.

hack a day has links to interesting hacks and how-to articles on various topics. (:tableend:)

Restore

## Tutorial.HomePage History

Hide minor edits - Show changes to output

July 02, 2008, at 03:11 PM by David A. Mellis -
Changed lines 2-3 from:
!!Arduino Examples
to:
!!Examples
Restore
July 02, 2008, at 03:11 PM by David A. Mellis -
Changed lines 4-5 from:
''See the '''[[Tutorial/Foundations | foundations page]]''' for in-depth description of core concepts of the Arduino hardware and software, and the '''[[Tutorial/Links | links page]]''' for other documentation.''
to:
''See the '''[[Tutorial/Foundations | foundations page]]''' for in-depth description of core concepts of the Arduino hardware and software; the '''[[Hacking/HomePage | hacking page]]''' for information on extending and modifying the Arduino hardware and software; and the '''[[Tutorial/Links | links page]]''' for other documentation.''
Restore
July 02, 2008, at 02:07 PM by David A. Mellis -
Added line 63:
* [[ Tutorial/ADXL3xx | Read an ADXL3xx accelerometer]]
Restore
May 21, 2008, at 09:44 PM by David A. Mellis -
Deleted lines 42-45:
!!!! Matrix Library

* [[HelloMatrix | Hello Matrix]]: blinks a smiley face on the LED matrix.
Restore
May 21, 2008, at 09:43 PM by David A. Mellis -
Added lines 43-46:
!!!! Matrix Library

* [[HelloMatrix | Hello Matrix]]: blinks a smiley face on the LED matrix.
Restore
May 21, 2008, at 09:36 PM by David A. Mellis -
Added lines 43-46:
!!!! Stepper Library

* [[MotorKnob | Motor Knob]]: control a stepper motor with a potentiometer.
Restore
May 21, 2008, at 09:25 PM by David A. Mellis - adding EEPROM examples.
Added lines 37-42:
!!!! EEPROM Library

* [[EEPROMClear | EEPROM Clear]]: clear the bytes in the EEPROM.
* [[EEPROMRead | EEPROM Read]]: read the EEPROM and send its values to the computer.
* [[EEPROMWrite | EEPROM Write]]: stores values from an analog input to the EEPROM.
Restore
May 21, 2008, at 09:22 PM by David A. Mellis -
Changed line 15 from:
* [[BlinkWithoutDelay | BlinkWithoutDelay]]: blinking an LED without using the delay() function.
to:
* [[BlinkWithoutDelay | Blink Without Delay]]: blinking an LED without using the delay() function.

Restore
April 29, 2008, at 06:55 PM by David A. Mellis - moving the resources to the links page.
Changed lines 2-5 from:
!!Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino [[Guide/HomePage | Getting Started]].
to:
!!Arduino Examples

''See the '''[[Tutorial/Foundations | foundations page]]''' for in-depth description of core concepts of the Arduino hardware and software, and the '''[[Tutorial/Links | links page]]''' for other documentation.''
Added line 15:
* [[BlinkWithoutDelay | BlinkWithoutDelay]]: blinking an LED without using the delay() function.
Changed lines 37-42 from:
!!!Timing & Millis
* [[ Tutorial/BlinkWithoutDelay | Blinking an LED without using the delay() function]]
* [[ Tutorial/stopwatch | Stopwatch ]]
(:if false:)
* [[TimeSinceStart]]:
(:ifend:)
to:
(:cell width=50%:)
Changed lines 41-42 from:
These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.
to:
These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included on the page.
Deleted lines 43-44:

Added lines 49-51:
!!!Timing & Millis
* [[ Tutorial/stopwatch | Stopwatch ]]
Deleted lines 75-125:
(:cell width=50%:)
!!!Foundations

See the [[Foundations| foundations page]] for explanations of the concepts involved in the Arduino hardware and software.

!!!Tutorials

Tutorials created by the Arduino community. Hosted on the publicly-editable [[http://www.arduino.cc/playground/ | playground wiki]].

[[http://www.arduino.cc/playground/Main/ArduinoCoreHardware | Board Setup and Configuration]]: Information about the components and usage of Arduino hardware.

[[http://www.arduino.cc/playground/Main/InterfacingWithHardware | Interfacing With Hardware]]: Code, circuits, and instructions for using various electronic components with an Arduino board.
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Output | Output]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Input | Input]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Interaction | Interaction]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Storage | Storage]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Communication | Communication]]

[[http://www.arduino.cc/playground/Main/InterfacingWithSoftware | Interfacing with Software]]: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

[[http://www.arduino.cc/playground/Main/GeneralCodeLibrary | Code Library and Tutorials]]: Arduino functions for performing specific tasks and other programming tutorials.

[[http://www.arduino.cc/playground/Main/ElectroInfoResources | Electronics Techniques]]: tutorials on soldering and other electronics resources.

!!!Manuals, Curricula, and Other Resources

[[http://www.tinker.it/en/uploads/v3_arduino_small.pdf | Arduino Booklet (pdf)]]: an illustrated guide to the philosophy and practice of Arduino.

[[http://www.ladyada.net/learn/arduino/index.html | Learn electronics using Arduino]]: an introduction to programming, input / output, communication, etc. using Arduino. By [[http://www.ladyada.net/ | ladyada]].

* [[http://www.ladyada.net/learn/arduino/lesson0.html | Lesson 0]]: Pre-flight check...Is your Arduino and computer ready?
* [[http://www.ladyada.net/learn/arduino/lesson1.html | Lesson 1]]: The &quot;Hello World!&quot; of electronics, a simple blinking light
* [[http://www.ladyada.net/learn/arduino/lesson2.html | Lesson 2]]: Sketches, variables, procedures and hacking code
* [[http://www.ladyada.net/learn/arduino/lesson3.html | Lesson 3]]: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
* [[http://www.ladyada.net/learn/arduino/lesson4.html | Lesson 4]]: The serial library and binary data - getting chatty with Arduino and crunching numbers
* [[http://www.ladyada.net/learn/arduino/lesson5.html | Lesson 5]]: Buttons &amp; switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

[[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]

[[http://todbot.com/blog/spookyarduino/ | Spooky Arduino]]: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]]
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]]
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class3.pdf | class 3 (communication, servos, and pwm)]]
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class4.pdf | class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)]]

[[http://todbot.com/blog/bionicarduino/ | Bionic Arduino]]: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

[[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | Examples from Tom Igoe]]

[[http://www.grayfuse.com/blog/?p=15 | Examples from Jeff Gray]]
Restore
April 23, 2008, at 10:29 PM by David A. Mellis -
Changed line 6 from:
(:table width=90% border=0 cellpadding=5 cellspacing=0:)
to:
(:table width=100% border=0 cellpadding=5 cellspacing=0:)
Restore
April 22, 2008, at 05:59 PM by Paul Badger -
Changed line 39 from:
to:
(:if false:)
Changed line 41 from:
to:
(:ifend:)
Restore
April 22, 2008, at 05:56 PM by Paul Badger -
Added lines 40-41:
* [[TimeSinceStart]]:
Restore
April 18, 2008, at 07:22 AM by Paul Badger -
Added lines 36-39:
!!!Timing & Millis
* [[ Tutorial/BlinkWithoutDelay | Blinking an LED without using the delay() function]]
* [[ Tutorial/stopwatch | Stopwatch ]]
Changed line 46 from:
* [[ Tutorial/BlinkWithoutDelay | Blinking an LED without using the delay() function]]
to:

Restore
April 08, 2008, at 08:23 PM by David A. Mellis -
Changed line 43 from:
* * [[TwoSwitchesOnePin]]: Read two switches with one I/O pin
to:
* [[TwoSwitchesOnePin]]: Read two switches with one I/O pin
Restore
April 08, 2008, at 08:22 PM by David A. Mellis - moving TwoSwitchesOnePin to "other examples" since it's not (yet) in the distribution.
Changed lines 18-19 from:
* [[TwoSwitchesOnePin]]: Read two switches with one I/O pin
to:
Added line 43:
* * [[TwoSwitchesOnePin]]: Read two switches with one I/O pin
Restore
April 08, 2008, at 07:41 PM by Paul Badger -
Changed lines 18-19 from:
to:
* [[TwoSwitchesOnePin]]: Read two switches with one I/O pin
Restore
March 09, 2008, at 07:20 PM by David A. Mellis -
Changed lines 73-78 from:
* [[Foundations| Foundations has moved here]]

* [[Bootloader]]: A small program pre-loaded on the Arduino board to allow uploading sketches.
to:

See the [[Foundations| foundations page]] for explanations of the concepts involved in the Arduino hardware and software.
Restore
March 07, 2008, at 09:26 PM by Paul Badger -
Changed lines 73-75 from:

to:
* [[Foundations| Foundations has moved here]]

Restore
March 07, 2008, at 09:24 PM by Paul Badger -
Changed lines 74-107 from:
* [[Memory]]: The various types of memory available on the Arduino board.

* [[Digital Pins]]: How the pins work and what it means for them to be configured as inputs or outputs.

* [[Analog Input Pins]]: Details about the analog-to-digital conversion and other uses of the pins.

* [[Foundations]]

(:if false:)

* [[PWM | PWM (Pulse-Width Modulation)]]: The method used by analogWrite() to simulate an analog output with digital pins.

* [[Communication]]: An overview of the various ways in which an Arduino board can communicate with other devices (serial, I2C, SPI, Midi, etc.)

* [[Serial | Serial Communication]]: How to send serial data from an Arduino board to a computer or other device (including via the USB connection).

* [[Interrupts]]: Code that interrupts other code under certain conditions.

* [[Numbers]]: The various types of numbers available and how to use them.

* [[Variables]]: How to define and use variables.

* [[Arrays]]: How to store multiple values of the same type.

* [[Pointers]]:

* [[Functions]]: How to write and call functions.

* [[Optimization]]: What to do when your program runs too slowly.

* [[Debugging]]: Figuring out what's wrong with your hardware or software and how to fix it.

(:ifend:)
to:

Restore
March 07, 2008, at 09:09 PM by Paul Badger -
Added lines 80-81:
* [[Foundations]]
Restore
February 15, 2008, at 06:00 PM by David A. Mellis -
Changed lines 72-73 from:
!!!Tutorials
to:
!!!Foundations
Changed lines 108-109 from:
!!!More Tutorials
to:
!!!Tutorials
Restore
February 13, 2008, at 10:42 PM by Paul Badger -
Changed lines 4-5 from:
Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino [[Guide/HomePage | guide]].
to:
Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino [[Guide/HomePage | Getting Started]].
Restore
February 13, 2008, at 10:06 PM by David A. Mellis -
Restore
February 13, 2008, at 09:58 PM by David A. Mellis -
Added lines 100-103:
* [[Optimization]]: What to do when your program runs too slowly.

* [[Debugging]]: Figuring out what's wrong with your hardware or software and how to fix it.
Restore
February 13, 2008, at 09:41 PM by David A. Mellis -
Added lines 90-99:
* [[Numbers]]: The various types of numbers available and how to use them.

* [[Variables]]: How to define and use variables.

* [[Arrays]]: How to store multiple values of the same type.

* [[Pointers]]:

* [[Functions]]: How to write and call functions.
Restore
February 13, 2008, at 09:38 PM by David A. Mellis -
Changed lines 86-87 from:
* [[Serial | Serial Communication]]: How to send serial data from an Arduino board to a computer or other device.
to:

* [[Serial | Serial Communication]]: How to send serial data from an Arduino board to a computer or other device (including via the USB connection).

* [[Interrupts]]: Code that interrupts other code under certain conditions.
Restore
February 13, 2008, at 09:36 PM by David A. Mellis -
Added lines 80-81:
(:if false:)
Added lines 84-89:
* [[Communication]]: An overview of the various ways in which an Arduino board can communicate with other devices (serial, I2C, SPI, Midi, etc.)

* [[Serial | Serial Communication]]: How to send serial data from an Arduino board to a computer or other device.

(:ifend:)
Restore
February 13, 2008, at 09:31 PM by David A. Mellis -
Changed lines 80-81 from:
* [[PWM]] (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.
to:
* [[PWM | PWM (Pulse-Width Modulation)]]: The method used by analogWrite() to simulate an analog output with digital pins.
Restore
February 13, 2008, at 09:30 PM by David A. Mellis -
Added lines 80-81:
* [[PWM]] (Pulse-Width Modulation): The method used by analogWrite() to simulate an analog output with digital pins.
Restore
February 13, 2008, at 09:22 PM by David A. Mellis -
Added lines 80-81:
* [[Bootloader]]: A small program pre-loaded on the Arduino board to allow uploading sketches.
Restore
February 13, 2008, at 09:12 PM by David A. Mellis -
Added lines 74-81:
* [[Memory]]: The various types of memory available on the Arduino board.

* [[Digital Pins]]: How the pins work and what it means for them to be configured as inputs or outputs.

* [[Analog Input Pins]]: Details about the analog-to-digital conversion and other uses of the pins.

!!!More Tutorials
Restore
January 11, 2008, at 11:31 AM by David A. Mellis - linking to board setup and configuration on the playground.
Added lines 76-77:
[[http://www.arduino.cc/playground/Main/ArduinoCoreHardware | Board Setup and Configuration]]: Information about the components and usage of Arduino hardware.
Restore
December 19, 2007, at 11:54 PM by David A. Mellis - adding links to other pages: the tutorial parts of the playground, ladyada's tutorials, todbot, etc.
Changed lines 36-42 from:

(:cell width=50%:)
!!!Tutorials

These are more complex tutorials for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.
to:
!!!Other Examples

These are more complex examples for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.
Changed lines 71-78 from:

!!!!Other Arduino Tutorials
* [[http://www.arduino.cc/playground/Learning/Tutorials | Tutorials from the Arduino playground]]

* [[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]
* [[http://todbot.com/blog/category/arduino/ | Spooky Arduino and more from Todbot]]
* [[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | Examples from Tom Igoe]]
* [[http://www.grayfuse.com/blog/?p=15 | Examples from Jeff Gray]]
to:
(:cell width=50%:)
!!!Tutorials

Tutorials created by the Arduino community. Hosted on the publicly-editable [[http://www.arduino.cc/playground/ | playground wiki]].

[[http://www.arduino.cc/playground/Main/InterfacingWithHardware | Interfacing With Hardware]]: Code, circuits, and instructions for using various electronic components with an Arduino board.
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Output | Output]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Input | Input]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Interaction | Interaction]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Storage | Storage]]
* [[http://www.arduino.cc/playground/Main/InterfacingWithHardware#Communication | Communication]]

[[http://www.arduino.cc/playground/Main/InterfacingWithSoftware | Interfacing with Software]]: how to get an Arduino board talking to software running on the computer (e.g. Processing, PD, Flash, Max/MSP).

[[http://www.arduino.cc/playground/Main/GeneralCodeLibrary | Code Library and Tutorials]]: Arduino functions for performing specific tasks and other programming tutorials.

[[http://www.arduino.cc/playground/Main/ElectroInfoResources | Electronics Techniques]]: tutorials on soldering and other electronics resources.

!!!Manuals, Curricula, and Other Resources

[[http://www.tinker.it/en/uploads/v3_arduino_small.pdf | Arduino Booklet (pdf)]]: an illustrated guide to the philosophy and practice of Arduino.

[[http://www.ladyada.net/learn/arduino/index.html | Learn electronics using Arduino]]: an introduction to programming, input / output, communication, etc. using Arduino. By [[http://www.ladyada.net/ | ladyada]].

* [[http://www.ladyada.net/learn/arduino/lesson0.html | Lesson 0]]: Pre-flight check...Is your Arduino and computer ready?
* [[http://www.ladyada.net/learn/arduino/lesson1.html | Lesson 1]]: The &quot;Hello World!&quot; of electronics, a simple blinking light
* [[http://www.ladyada.net/learn/arduino/lesson2.html | Lesson 2]]: Sketches, variables, procedures and hacking code
* [[http://www.ladyada.net/learn/arduino/lesson3.html | Lesson 3]]: Breadboards, resistors and LEDs, schematics, and basic RGB color-mixing
* [[http://www.ladyada.net/learn/arduino/lesson4.html | Lesson 4]]: The serial library and binary data - getting chatty with Arduino and crunching numbers
* [[http://www.ladyada.net/learn/arduino/lesson5.html | Lesson 5]]: Buttons &amp; switches, digital inputs, pull-up and pull-down resistors, if/if-else statements, debouncing and your first contract product design.

[[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]

[[http://todbot.com/blog/spookyarduino/ | Spooky Arduino]]: Longer presentation-format documents introducing Arduino from a Halloween hacking class taught by TodBot:
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]]
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]]
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class3.pdf | class 3 (communication, servos, and pwm)]]
* [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class4.pdf | class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)]]

[[http://todbot.com/blog/bionicarduino/ | Bionic Arduino]]: another Arduino class from TodBot, this one focusing on physical sensing and making motion.

[[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | Examples from Tom Igoe]]

[[http://www.grayfuse.com/blog/?p=15 | Examples from Jeff Gray]]

Restore

December 13, 2007, at 11:08 PM by David A. Mellis - adding debounce example.

Added line 16:

* [[Debounce]]: read a pushbutton, filtering noise.

Restore

August 28, 2007, at 11:15 PM by Tom Igoe -

Changed lines 71-72 from:

to:

* [[Tutorial/X10 | X10 output]] control devices over AC powerlines using X10

Restore

June 15, 2007, at 05:04 PM by David A. Mellis - adding link to Processing (for the communication examples)

Added lines 27-28:

''These examples include code that allows the Arduino to talk to Processing sketches running on the computer. For more information or to download Processing, see [[http://processing.org/ | processing.org]].''

Restore

June 12, 2007, at 08:57 AM by David A. Mellis - removing link to obsolete joystick example.

Deleted line 43:

* [[ Tutorial/JoyStick | Interfacing a Joystick]]

Restore

June 11, 2007, at 11:14 PM by David A. Mellis -

Changed lines 10-11 from:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the '''examples''' folder. (If you're looking for an older example, check the [[HomePage-0007 | Arduino 0007 tutorials page]].

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the '''examples''' folder. (If you're looking for an older example, check the [[HomePage-0007 | Arduino 0007 tutorials page]].)

Restore

June 11, 2007, at 11:13 PM by David A. Mellis -

Changed lines 10-11 from:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the '''examples''' folder.

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the '''examples''' folder. (If you're looking for an older example, check the [[HomePage-0007 | Arduino 0007 tutorials page]].

Restore

June 11, 2007, at 11:10 PM by David A. Mellis - updating to 0008 examples

Changed lines 10-11 from:

!!!! Digital Output

* [[ Tutorial/Blinking LED | Blinking LED]]

to:

Simple programs that demonstrate the use of the Arduino board. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the '''examples''' folder.

!!!! Digital I/O

* [[Blink]]: turn an LED on and off.

* [[Button]]: use a pushbutton to control an LED.

* [[Loop]]: controlling multiple LEDs with a loop and an array.

!!!! Analog I/O

* [[Analog Input]]: use a potentiometer to control the blinking of an LED.

* [[Fading]]: uses an analog output (PWM pin) to fade an LED.

* [[Knock]]: detect knocks with a piezo element.

* [[Smoothing]]: smooth multiple readings of an analog input.

!!!! Communication

* [[ASCII Table]]: demonstrates Arduino's advanced serial output functions.

* [[Dimmer]]: move the mouse to change the brightness of an LED.

```
* [[Graph]]: sending data to the computer and graphing it in Processing.
* [[Physical Pixel]]: turning on and off an LED by sending data from Processing.
* [[Virtual Color Mixer]]: sending multiple variables from Arduino to the computer and reading them in Processing.



(:cell width=50%:)
!!!Tutorials

These are more complex tutorials for using particular electronic components or accomplishing specific tasks. The code is included in the tutorial.

!!!!Miscellaneous
Deleted lines 42-51:
* [[ Tutorial/Dimming LEDs | Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM) ]]
* [[ Tutorial/Color Crossfader | More complex dimming/color crossfader ]]
* [[ Tutorial/Knight Rider|Knight Rider example]]
* [[ Tutorial/ShootingStar | Shooting star]]
* [[ http://www.arduino.cc/playground/Main/PWMallPins | PWM all of the digital pins in a sinewave pattern]]

!!!! Digital Input
* [[http://itp.nyu.edu/physcomp/Labs/DigitalInOut | Digital Input and Output]] (from [[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]])
* [[ Tutorial/Pushbutton | Read a Pushbutton]]
* [[ Tutorial/Switch | Using a pushbutton as a switch]]
Deleted lines 43-45:

!!!! Analog Input
* [[ Tutorial/Potentiometer | Read a Potentiometer]]
Deleted lines 45-46:
* [[ Tutorial/Knock Sensor | Read a Piezo Sensor]]
* [[ Tutorial/LED cross-fades with potentiometer | 3 LED cross-fades with a potentiometer ]]
Changed lines 52-53 from:
*[[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 | Use two Arduino pins as a capacitive sensor]]
to:
Deleted line 54:
* [[http://www.arduino.cc/playground/Main/Freqout|More sound ideas]]
Added line 64:
* [[ Tutorial/DMX Master | Build your own DMX Master device]]
Changed lines 70-72 from:
*[[Tutorial/ShiftIn | Multiple digital inputs with a CD4021 Shift Register]]

!!!Other Arduino Examples
to:



!!!!Other Arduino Tutorials
* [[http://www.arduino.cc/playground/Learning/Tutorials | Tutorials from the Arduino playground]]
Added line 75:
* [[http://todbot.com/blog/category/arduino/ | Spooky Arduino and more from Todbot]]
Deleted lines 78-105:
(:cell width=50%:)
!!!Interfacing with Other Software
* [[http://itp.nyu.edu/physcomp/Labs/Serial | Introduction to Serial Communication]] (from
[[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]])
* [[http://www.arduino.cc/playground/Interfacing/Flash | Arduino + Flash]]
* [[http://www.arduino.cc/playground/Interfacing/Processing | Arduino + Processing]]
* [[http://www.arduino.cc/playground/Interfacing/PD | Arduino + PD]]
* [[http://www.arduino.cc/playground/Interfacing/MaxMSP | Arduino + MaxMSP]]
* [[http://www.arduino.cc/playground/Interfacing/VVVV | Arduino + VVVV]]
* [[http://www.arduino.cc/playground/Interfacing/Director | Arduino + Director]]
* [[http://www.arduino.cc/playground/Interfacing/Ruby | Arduino + Ruby]]
* [[http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino/ | Arduino + C]]
```

!!!Tech Notes (from the [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl | forums]] or [[http://www.arduino.cc/playground/ | playground]])

* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1147888882 | Software serial]] (serial on pins besides 0 and 1)
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1138310274 | L297 motor driver]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1135701338 | Hex inverter]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1138666403 | Analog multiplexer]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1138892708 | Power supplies]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1139161553 | The components on the Arduino board]]
* [[http://www.arduino.cc/playground/Learning/BuildProcess | Arduino build process]]
* [[http://www.arduino.cc/playground/Code/OSXISPMKII | AVRISP mkII on the Mac]]
* [[http://www.arduino.cc/playground/Code/EEPROM-Flash | Non-volatile memory (EEPROM)]]
* [[http://www.arduino.cc/playground/Learning/Tutorial01 | Bluetooth]]
* [[http://mrtof.danslchamp.org/AXIC | Zigbee]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1146679536 | LED as light sensor]] (en Francais)
* [[http://www.arduino.cc/playground/Learning/Asuro | Arduino and the Asuro robot]]
* [[http://www.arduino.cc/playground/Learning/CommandLine | Using Arduino from the command line]]

Restore

May 11, 2007, at 06:06 AM by Paul Badger -

Changed lines 17-18 from:

to:

* [[ http://www.arduino.cc/playground/Main/PWMallPins | PWM all of the digital pins in a sinewave pattern]]

Restore

May 10, 2007, at 07:07 PM by Paul Badger -

Changed lines 36-37 from:

* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259]] |Use a couple of Arduino pins as a capacitive sensor]]

to:

*[[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259 | Use two Arduino pins as a capacitive sensor]]

Restore

May 10, 2007, at 07:05 PM by Paul Badger -

Changed lines 36-37 from:

* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259]] Use a couple of Arduino pins as a capacitive sensor

to:

* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259]] |Use a couple of Arduino pins as a capacitive sensor]]

Restore

May 10, 2007, at 07:04 PM by Paul Badger -

Changed lines 36-37 from:

to:

* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1171076259]] Use a couple of Arduino pins as a capacitive sensor

Restore

May 10, 2007, at 06:59 PM by Paul Badger -

Added line 39:

* [[http://www.arduino.cc/playground/Main/Freqout|More sound ideas]]

Restore

April 24, 2007, at 03:40 PM by Clay Shirky -

Changed lines 13-14 from:

* [[ Tutorial/Dimming LEDs | Dimming 3 LEDs with Pulse-Width Modulation (PWM) ]]

to:

* [[ Tutorial/Dimming LEDs | Simple Dimming 3 LEDs with Pulse-Width Modulation (PWM) ]]
* [[ Tutorial/Color Crossfader | More complex dimming/color crossfader ]]

Restore

February 08, 2007, at 12:02 PM by Carlyn Maw -

Changed lines 52-53 from:

to:

*[[Tutorial/ShiftIn | Multiple digital inputs with a CD4021 Shift Register]]

Restore

February 06, 2007, at 02:52 PM by Carlyn Maw -

Changed lines 52-54 from:

*[[Tutorial/ShiftIn | Multiple digital ins with a CD4021 Shift Register]]

to:

Restore

February 06, 2007, at 02:51 PM by Carlyn Maw -

Changed lines 52-53 from:

to:
*[[Tutorial/ShiftIn | Multiple digital ins with a CD4021 Shift Register]]

Restore
January 30, 2007, at 03:37 PM by David A. Mellis -
Deleted line 46:
* [[ Tutorial/DMX Master | Build your own DMX Master device]]
Restore
December 25, 2006, at 11:57 PM by David A. Mellis -
Added line 20:
* [[ Tutorial/Switch | Using a pushbutton as a switch]]
Restore
December 07, 2006, at 06:04 AM by David A. Mellis - adding link to todbot's C serial port code.
Changed lines 69-70 from:
to:
* [[http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino/ | Arduino + C]]
Restore
December 02, 2006, at 10:43 AM by David A. Mellis -
Added line 1:
(:title Tutorials:)
Restore
November 21, 2006, at 10:13 AM by David A. Mellis -
Added line 64:
* [[http://www.arduino.cc/playground/Interfacing/MaxMSP | Arduino + MaxMSP]]
Changed lines 67-68 from:
to:
* [[http://www.arduino.cc/playground/Interfacing/Ruby | Arduino + Ruby]]
Restore
November 18, 2006, at 02:42 AM by David A. Mellis -
Changed lines 20-21 from:
* [[ Tutorial/ControleLEDcircleWithJoystick | Controlling an LED circle with a joystick]]
to:
Added line 24:
* [[ Tutorial/ControleLEDcircleWithJoystick | Controlling an LED circle with a joystick]]
Restore
November 09, 2006, at 03:10 PM by Carlyn Maw -
Changed lines 50-51 from:

to:
*[[Tutorial/ShiftOut | Multiple digital outs with a 595 Shift Register]]

Restore
November 06, 2006, at 10:49 AM by David A. Mellis -
Changed lines 37-38 from:
* [[http://itp.nyu.edu/physcomp/Labs/MIDIOutput | MIDI Output]] (from [[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]])
to:
* [[http://itp.nyu.edu/physcomp/Labs/MIDIOutput | MIDI Output]] (from [[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]]) and [[http://todbot.com/blog/2006/10/29/spooky-arduino-projects-4-and-musical-arduino/ | from Spooky Arduino]]
Restore
November 04, 2006, at 12:25 PM by David A. Mellis -
Deleted line 53:
Deleted line 54:
Restore
November 04, 2006, at 12:24 PM by David A. Mellis -
Added lines 51-58:

!!!Other Arduino Examples
* [[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]

* [[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | Examples from Tom Igoe]]

* [[http://www.grayfuse.com/blog/?p=15 | Examples from Jeff Gray]]
Deleted lines 83-89:
!!!Other Arduino Examples
* [[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]

* [[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | Examples from Tom Igoe]].

* [[http://www.grayfuse.com/blog/?p=15 | Examples from Jeff Gray]].
Restore
November 04, 2006, at 12:24 PM by David A. Mellis -
Changed lines 50-51 from:
!!!! [[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]
to:
Changed lines 77-78 from:
Also, see the [[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | examples from Tom Igoe]] and [[http://www.grayfuse.com/blog/?p=15 | those from Jeff Gray]].
to:
* [[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]

* [[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | Examples from Tom Igoe]].

* [[http://www.grayfuse.com/blog/?p=15 | Examples from Jeff Gray]].
Restore
November 04, 2006, at 12:23 PM by David A. Mellis -
Changed line 77 from:
!!!Other Arduino Sites
to:
!!!Other Arduino Examples
Deleted lines 79-81:
!!!Do you need extra help?
Is there a sensor you would like to see characterized for Arduino, or is there something you would like to see published in this site? Refer to the [[ http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl | forum]] for further help.
Restore
November 04, 2006, at 10:38 AM by David A. Mellis -
Changed lines 3-4 from:
Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino [[Guide/Homepage | guide]].
to:
Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino [[Guide/HomePage | guide]].
Restore
November 04, 2006, at 10:37 AM by David A. Mellis - lots of content moved to the new guide.
Deleted lines 52-67:
!!!The Arduino board

This [[ Tutorial/ArduinoBoard | guide to the Arduino board]] explains the functions of the various parts of the board.

!!!The Arduino environment

This [[Main/Environment | guide to the Arduino IDE]] (integrated development environment) explains the functions of the various buttons and menus.

The [[Main/libraries]] page explains how to use libraries in your sketches and how to make your own.

!!!Video Lectures by Tom Igoe
[[http://www.sbk.flr4.org/arduino/index.html | Watch Tom]] introduce Arduino. Thanks to Pollie Barden for the great videos.

!!!Course Guides
[[http://todbot.com/blog | todbot]] has some very detailed, illustrated tutorials from his [[http://todbot.com/blog/spookyarduino/ | Spooky Projects]] course: [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class3.pdf | class 3 (communication, servos, and pwm)]],

[[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class4.pdf | class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)]]
Deleted lines 82-87:
!!!External Resources
[[http://www.instantsoup.org/ | Instant Soup]] is an introduction to electronics through a series of beautifully-documented fun projects.

[[http://www.makezine.com/ | Make magazine]] has some great links in its
[[http://www.makezine.com/blog/archive/electronics/ | electronics archive]].

[[http://www.hackaday.com/ | hack a day]] has links to interesting hacks and how-to articles on various topics.
Restore
November 04, 2006, at 10:17 AM by David A. Mellis -
Changed lines 3-4 from:
Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the [[Main/Howto]].
to:
Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the Arduino [[Guide/Homepage | guide]].
Restore
November 01, 2006, at 06:54 PM by Carlyn Maw -
Deleted line 49:
*[[Tutorial/ShiftOut | Extend your digital outs with 74HC595 shift registers]]
Restore
November 01, 2006, at 06:06 PM by Carlyn Maw -
Added line 50:
*[[Tutorial/ShiftOut | Extend your digital outs with 74HC595 shift registers]]
Restore
October 31, 2006, at 10:47 AM by Tod E. Kurt -
Changed lines 67-68 from:
[[http://todbot.com/blog | todbot]] has some very detailed, illustrated tutorials from his
[[http://todbot.com/blog/spookyarduino/ | Spooky Projects]] course: [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class3.pdf | class 3 (communication, servos, and pwm)]].
to:
[[http://todbot.com/blog | todbot]] has some very detailed, illustrated tutorials from his
[[http://todbot.com/blog/spookyarduino/ | Spooky Projects]] course: [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class3.pdf | class 3 (communication, servos, and pwm)]],
[[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class4.pdf | class 4 (piezo sound & sensors, arduino+processing, stand-alone operation)]]
Restore
October 22, 2006, at 12:52 PM by David A. Mellis -
Changed lines 1-4 from:
!!Learning to use Arduino

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the [[Main/Howto]].
to:
!!Arduino Tutorials

Here you will find a growing number of examples and tutorials for accomplishing specific tasks or interfacing to other hardware and software with Arduino. For instructions on getting the board and environment up and running, see the [[Main/Howto]].
Restore
October 22, 2006, at 12:51 PM by David A. Mellis -
Changed lines 67-68 from:
[[http://todbot.com/blog | todbot]] has some very detailed, illustrated tutorials from his
[[http://todbot.com/blog/spookyarduino/ | Spooky Projects]] course: [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]], [[http://todbot.com/blog/wp-

content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]].
to:
[[http://todbot.com/blog | todbot]] has some very detailed, illustrated tutorials from his [[http://todbot.com/blog/spookyarduino/ | Spooky Projects]] course: [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class3.pdf | class 3 (communication, servos, and pwm)]].
Restore
October 21, 2006, at 04:25 PM by David A. Mellis - adding links to todbot's class notes.
Added lines 66-68:
!!!Course Guides
[[http://todbot.com/blog | todbot]] has some very detailed, illustrated tutorials from his [[http://todbot.com/blog/spookyarduino/ | Spooky Projects]] course: [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class1.pdf | class 1 (getting started)]], [[http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf | class 2 (input and sensors)]].
Restore
October 08, 2006, at 05:46 PM by David A. Mellis -
Changed lines 59-62 from:
This [[Environment | guide to the Arduino IDE]] (integrated development environment) explains the functions of the various buttons and menus.

The [[libraries]] page explains how to use libraries in your sketches and how to make your own.
to:
This [[Main/Environment | guide to the Arduino IDE]] (integrated development environment) explains the functions of the various buttons and menus.

The [[Main/libraries]] page explains how to use libraries in your sketches and how to make your own.
Restore
October 08, 2006, at 05:45 PM by David A. Mellis -
Changed lines 3-4 from:
Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the [[Howto]].
to:
Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the [[Main/Howto]].
Restore
October 08, 2006, at 05:38 PM by David A. Mellis -
Added lines 1-102:
!!Learning to use Arduino

Here you will find a growing number of step by step guides on how to learn the basics of arduino and the things you can do with it. For instructions on getting the board and IDE up and running, see the [[Howto]].

(:table width=90% border=0 cellpadding=5 cellspacing=0:)
(:cell width=50%:)
!!!Examples

!!!! Digital Output
* [[ Tutorial/Blinking LED | Blinking LED]]
* [[ Tutorial/BlinkWithoutDelay | Blinking an LED without using the delay() function]]
* [[ Tutorial/Dimming LEDs | Dimming 3 LEDs with Pulse-Width Modulation (PWM) ]]
* [[ Tutorial/Knight Rider|Knight Rider example]]
* [[ Tutorial/ShootingStar | Shooting star]]

!!!! Digital Input
* [[http://itp.nyu.edu/physcomp/Labs/DigitalInOut | Digital Input and Output]] (from [[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]])
* [[ Tutorial/Pushbutton | Read a Pushbutton]]
* [[ Tutorial/Tilt Sensor | Read a Tilt Sensor]]
* [[ Tutorial/ControleLEDcircleWithJoystick | Controlling an LED circle with a joystick]]

!!!! Analog Input
* [[ Tutorial/Potentiometer | Read a Potentiometer]]
* [[ Tutorial/JoyStick | Interfacing a Joystick]]
* [[ Tutorial/Knock Sensor | Read a Piezo Sensor]]

```
* [[ Tutorial/LED cross-fades with potentiometer | 3 LED cross-fades with a potentiometer ]]
* [[ Tutorial/LED color mixer with 3 potentiometers | 3 LED color mixer with 3 potentiometers]]

!!!! Complex Sensors
* [[ Tutorial/Accelerometer Memsic 2125 | Read an Accelerometer]]
* [[ Tutorial/Ultrasound Sensor | Read an Ultrasonic Range Finder (ultrasound sensor)]]
* [[ Tutorial/qt401 | Reading the qprox qt401 linear touch sensor]]

!!!! Sound
* [[Tutorial/Play Melody | Play Melodies with a Piezo Speaker]]
* [[Tutotial/Keyboard Serial | Play Tones from the Serial Connection]]
* [[http://itp.nyu.edu/physcomp/Labs/MIDIOutput | MIDI Output]] (from [[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]])

!!!! Interfacing w/ Hardware
* [[ Tutorial/LED Driver | Multiply the Amount of Outputs with an LED Driver ]]
* [[ Tutorial/LCD 8 bits | Interfacing an LCD display with 8 bits]]
**[[Tutorial/LCD library | LCD interface library]]
* [[http://itp.nyu.edu/physcomp/Labs/DCMotorControl | Driving a DC Motor with an L293]] (from [[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]]).
* [[ Tutorial/Stepper Unipolar | Driving a Unipolar Stepper Motor]]
* [[ Tutorial/DMX Master | Build your own DMX Master device]]
* [[ Tutorial/Software Serial | Implement a software serial connection]]
** [[ http://www.arduino.cc/en/Tutorial/ArduinoSoftwareRS232 | RS-232 computer interface]]
*[[Tutorial/SPI_EEPROM | Interface with a serial EEPROM using SPI]]
*[[Tutorial/SPI_Digital_Pot | Control a digital potentiometer using SPI]]
!!!! [[ http://itp.nyu.edu/physcomp/Labs/Labs | Example labs from ITP]]

(:cell width=50%:)
!!!The Arduino board

This [[ Tutorial/ArduinoBoard | guide to the Arduino board]] explains the functions of the various parts of the board.

!!!The Arduino environment

This [[Environment | guide to the Arduino IDE]] (integrated development environment) explains the functions of the various buttons and menus.

The [[libraries]] page explains how to use libraries in your sketches and how to make your own.

!!!Video Lectures by Tom Igoe
[[http://www.sbk.flr4.org/arduino/index.html | Watch Tom]] introduce Arduino. Thanks to Pollie Barden for the great videos.

!!!Interfacing with Other Software
* [[http://itp.nyu.edu/physcomp/Labs/Serial | Introduction to Serial Communication]] (from [[http://itp.nyu.edu/physcomp/Labs/Labs | ITP physcomp labs]])
* [[http://www.arduino.cc/playground/Interfacing/Flash | Arduino + Flash]]
* [[http://www.arduino.cc/playground/Interfacing/Processing | Arduino + Processing]]
* [[http://www.arduino.cc/playground/Interfacing/PD | Arduino + PD]]
* [[http://www.arduino.cc/playground/Interfacing/VVVV | Arduino + VVVV]]
* [[http://www.arduino.cc/playground/Interfacing/Director | Arduino + Director]]

!!!Tech Notes (from the [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl | forums]] or [[http://www.arduino.cc/playground/ | playground]])
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1147888882 | Software serial]] (serial on pins besides 0 and 1)
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1138310274 | L297 motor driver]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1135701338 | Hex inverter]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1138666403 | Analog multiplexer]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1138892708 | Power supplies]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1139161553 | The components on the Arduino board]]
* [[http://www.arduino.cc/playground/Learning/BuildProcess | Arduino build process]]
* [[http://www.arduino.cc/playground/Code/OSXISPMKII | AVRISP mkII on the Mac]]
* [[http://www.arduino.cc/playground/Code/EEPROM-Flash | Non-volatile memory (EEPROM)]]
* [[http://www.arduino.cc/playground/Learning/Tutorial01 | Bluetooth]]
```

* [[http://mrtof.danslchamp.org/AXIC | Zigbee]]
* [[http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1146679536 | LED as light sensor]] (en Francais)
* [[http://www.arduino.cc/playground/Learning/Asuro | Arduino and the Asuro robot]]
* [[http://www.arduino.cc/playground/Learning/CommandLine | Using Arduino from the command line]]

!!!Other Arduino Sites
Also, see the [[http://www.tigoe.net/pcomp/code/archives/avr/arduino/index.shtml | examples from Tom Igoe]] and [[http://www.grayfuse.com/blog/?p=15 | those from Jeff Gray]].

!!!Do you need extra help?
Is there a sensor you would like to see characterized for Arduino, or is there something you would like to see published in this site? Refer to the [[ http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl | forum]] for further help.

!!!External Resources
[[http://www.instantsoup.org/ | Instant Soup]] is an introduction to electronics through a series of beautifully-documented fun projects.

[[http://www.makezine.com/ | Make magazine]] has some great links in its
[[http://www.makezine.com/blog/archive/electronics/ | electronics archive]].

[[http://www.hackaday.com/ | hack a day]] has links to interesting hacks and how-to articles on various topics.
(:tableend:)
Restore

## Time Since Start

A sketch to illustrate keeping track of real world time since the board was started or reset. The sketch is based on the millis() function, which returns the number of milliseconds since the Freeduino was reset.

One problem with millis() is that the unsigned long variable it returns will overflow every 9.3 hours. If this overflow is not managed, math in a sketch will break.

Re: Millis Rollover Workaround Reply #1 - 06.09.2007 at 01:06:47 Quote You can access it by putting:

extern unsigned long timer0_overflow_count;

at the top of your sketch. However, there's probably a better way to deal with the overflow. For example, if you just need to be able to get the duration between multiple calls to millis(), you should be able to do a millis() - previous_millis (which should work past the rollover).

Or, you could do something like:

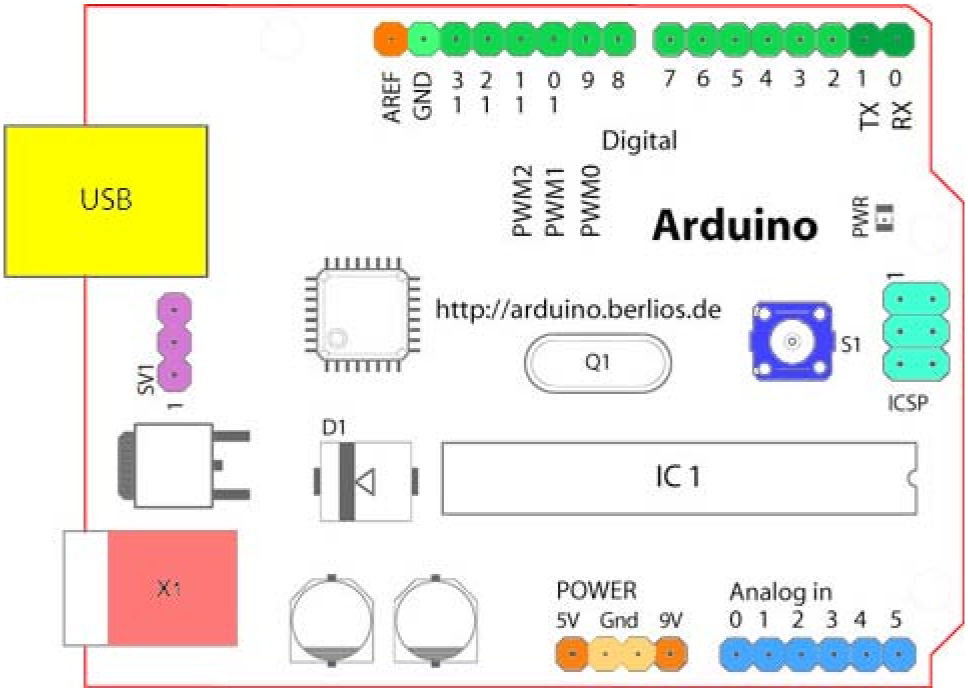
current_millis_value = millis(); m += current_millis_value - previous_millis_value; // should work even when millis rolls over seconds += m / 1000; m = m % 1000; minutes += seconds / 60; seconds = seconds % 60; hours += minutes / 60; minutes = minutes % 60; previous_millis_value = current_millis_value;

*(redirected from Tutorial.ArduinoBoard)*

**Guide** Contents | Introduction | How To: Windows, Mac OS X, Linux; Arduino Nano, Arduino Mini, Arduino BT, LilyPad Arduino; Xbee shield | Troubleshooting | Environment (:redirect Reference/Board:)

## Introduction to the Arduino Board

Looking at the board from the top down, this is an outline of what you will see (parts of the board you might interact with in the course of normal use are highlighted):



Starting clockwise from the top center:

- Analog Reference pin (orange)
- Digital Ground (light green)
- Digital Pins 2-13 (green)
- Digital Pins 0-1/Serial In/Out - TX/RX (dark green) - *These pins cannot be used for digital i/o (**digitalRead** and **digitalWrite**) if you are also using serial communication (e.g. **Serial.begin**).*
- Reset Button - S1 (dark blue)
- In-circuit Serial Programmer (blue-green)
- Analog In Pins 0-5 (light blue)
- Power and Ground Pins (power: orange, grounds: light orange)
- External Power Supply In (9-12VDC) - X1 (pink)
- Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1 (purple)
- USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow)

### Digital Pins

In addition to the specific functions listed below, the digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands. Each pin has an internal pull-up resistor which can be turned on and off using digitalWrite() (w/ a value of HIGH or LOW, respectively) when the pin is configured as an input. The maximum current per pin is 40 mA.

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. On the Arduino Diecimila, these pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip. On the Arduino BT, they are connected to the corresponding pins of the WT11 Bluetooth module. On the Arduino Mini and LilyPad Arduino, they are intended for use with an external TTL serial module (e.g. the Mini-USB Adapter).
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function. On boards with an ATmega8, PWM output is available only on pins 9, 10, and 11.
- **BT Reset: 7.** (Arduino BT-only) Connected to the reset line of the bluetooth module.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
- **LED: 13.** On the Diecimila and LilyPad, there is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

### Analog Pins

In addition to the specific functions listed below, the analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19. Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins.

- **I$^2$C: 4 (SDA) and 5 (SCL).** Support I$^2$C (TWI) communication using the Wire library (documentation on the Wiring website).

### Power Pins

- **VIN** (sometimes labelled "9V"). The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin. Note that different boards accept different input voltages ranges, please see the documentation for your board. Also note that the LilyPad has no VIN pin and accepts only a regulated input.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** (Diecimila-only) A 3.3 volt supply generated by the on-board FTDI chip.
- **GND.** Ground pins.

### Other Pins

- **AREF.** Reference voltage for the analog inputs. Not currently supported by the Arduino software.
- **Reset.** (Diecimila-only) Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.