# INTRODUCTION TO THE ARDUINO MICROCONTROLLER

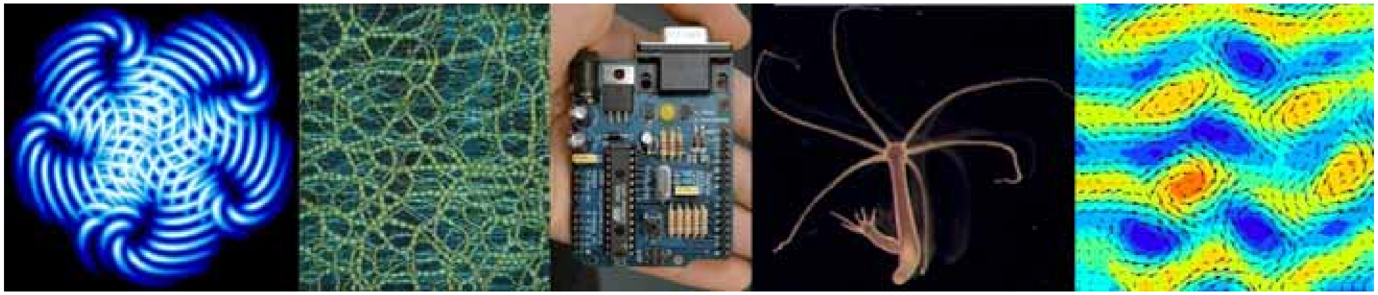
## Hands-on Research in Complex Systems

Shanghai Jiao Tong University

June 17 – 29, 2012

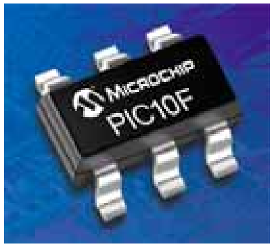
Instructor: Thomas E. Murphy (University of Maryland)

Assisted by: Hien Dao (UMD), Caitlin Williams (UMD) and 徐浩 (SJTU)



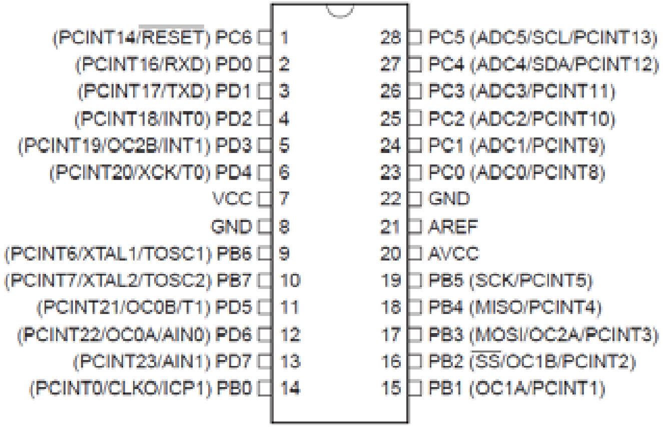
# What is a Microcontroller (µC, MCU)

- Computer on a single integrated chip
  - Processor (CPU)
  - Memory (RAM / ROM / Flash)
  - I/O ports (USB, I2C, SPI, ADC)
- Common microcontroller families:
  - Intel: 4004, 8008, etc.
  - Atmel: AT and AVR
  - Microchip: PIC
  - ARM: (multiple manufacturers)
- Used in:
  - Cellphones,
  - Toys
  - Household appliances
  - Cars
  - Cameras



# The ATmega328P Microcontroller
## *(used by the Arduino)*

- AVR 8-bit RISC architecture
- Available in DIP package
- Up to 20 MHz clock
- 32kB flash memory
- 1 kB SRAM
- 23 programmable I/O channels
- Six 10-bit ADC inputs
- Three timers/counters
- Six PWM outputs


(PCINT14/$\overline{\text{RESET}}$) PC6 □ 1
(PCINT16/RXD) PD0 □ 2
(PCINT17/TXD) PD1 □ 3
(PCINT18/INT0) PD2 □ 4
(PCINT19/OC2B/INT1) PD3 □ 5
(PCINT20/XCK/T0) PD4 □ 6
VCC □ 7
GND □ 8
(PCINT6/XTAL1/TOSC1) PB6 □ 9
(PCINT7/XTAL2/TOSC2) PB7 □ 10
(PCINT21/OC0B/T1) PD5 □ 11
(PCINT22/OC0A/AIN0) PD6 □ 12
(PCINT23/AIN1) PD7 □ 13
(PCINT0/CLKO/ICP1) PB0 □ 14
28 □ PC5 (ADC5/SCL/PCINT13)
27 □ PC4 (ADC4/SDA/PCINT12)
26 □ PC3 (ADC3/PCINT11)
25 □ PC2 (ADC2/PCINT10)
24 □ PC1 (ADC1/PCINT9)
23 □ PC0 (ADC0/PCINT8)
22 □ GND
21 □ AREF
20 □ AVCC
19 □ PB5 (SCK/PCINT5)
18 □ PB4 (MISO/PCINT4)
17 □ PB3 (MOSI/OC2A/PCINT3)
16 □ PB2 ($\overline{\text{SS}}$/OC1B/PCINT2)
15 □ PB1 (OC1A/PCINT1)

# What is Arduino Not?

- It is not a chip (IC)
- It is not a board (PCB)
- It is not a company or a manufacturer
- It is not a programming language
- It is not a computer architecture

*(although it involves all of these things...)*

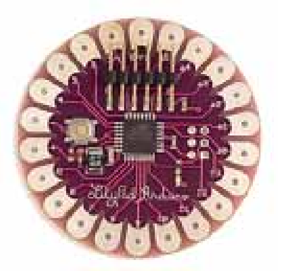
# So what *is* Arduino?

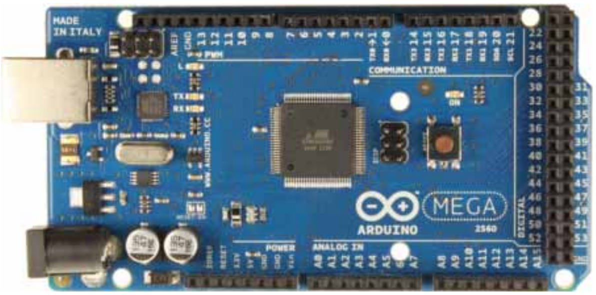
It's a ***movement***, not a microcontroller:

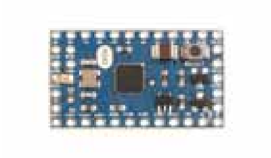
- Founded by Massimo Banzi and David Cuartielles in 2005
- Based on "Wiring Platform", which dates to 2003
- Open-source hardware platform
- Open source development environment
  - Easy-to learn language and libraries (based on Wiring language)
  - Integrated development environment (based on Processing programming environment)
  - Available for Windows / Mac / Linux

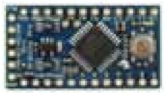
# The Many Flavors of Arduino

- Arduino Uno
- Arduino Leonardo
- Arduino LilyPad
- Arduino Mega
- Arduino Nano
- Arduino Mini
- Arduino Mini Pro
- Arduino BT



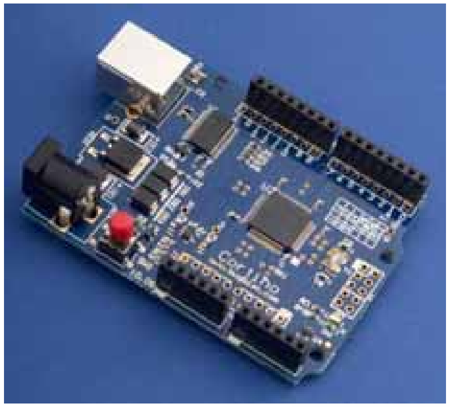






# Arduino-like Systems

- Cortino (ARM)
- Xduino (ARM)
- LeafLabs Maple (ARM)
- BeagleBoard (Linux)
- Wiring Board (Arduino predecessor)

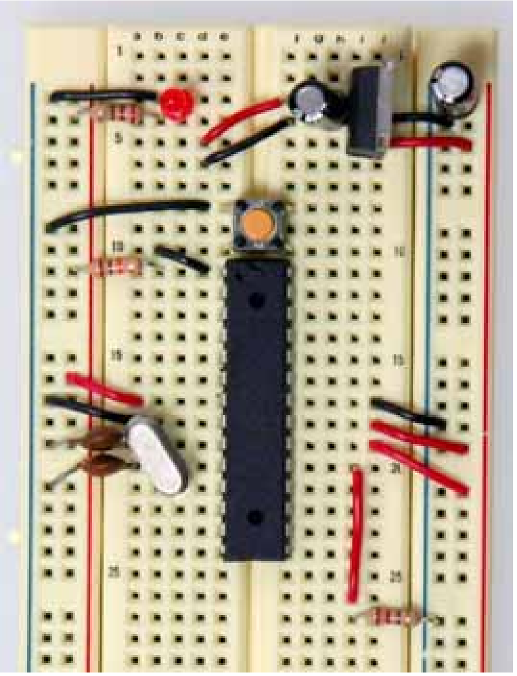


# Arduino Add-ons (Shields)

- TFT Touch Screen
- Data logger
- Motor/Servo shield
- Ethernet shield
- Audio wave shield
- Cellular/GSM shield
- WiFi shield
- Proto-shield
- ...many more

# Where to Get an Arduino Board

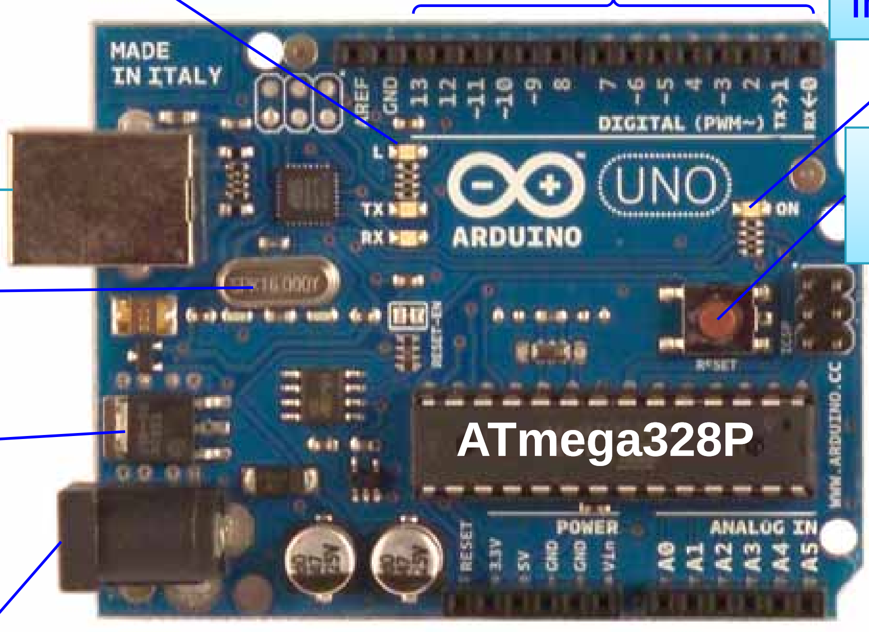
- Purchase from online vendor (available worldwide)
  - – Sparkfun
  - – Adafruit
  - – DFRobot
- ... or build your own
  - – PC board
  - – Solderless breadboard



http://itp.nyu.edu/physcomp/Tutorials/ArduinoBreadboard

# Getting to know the Arduino: Electrical Inputs and Outputs

- Input voltage: 7-12 V (USB, DC plug, or Vin)
- Max output current per pin: 40 mA



LED

14 digital inputs/outputs (6 PWM outputs)

Power indicator

Reset Button

USB connection

16 MHz clock

Voltage regulator

ATmega328P

AC/DC adapter jack

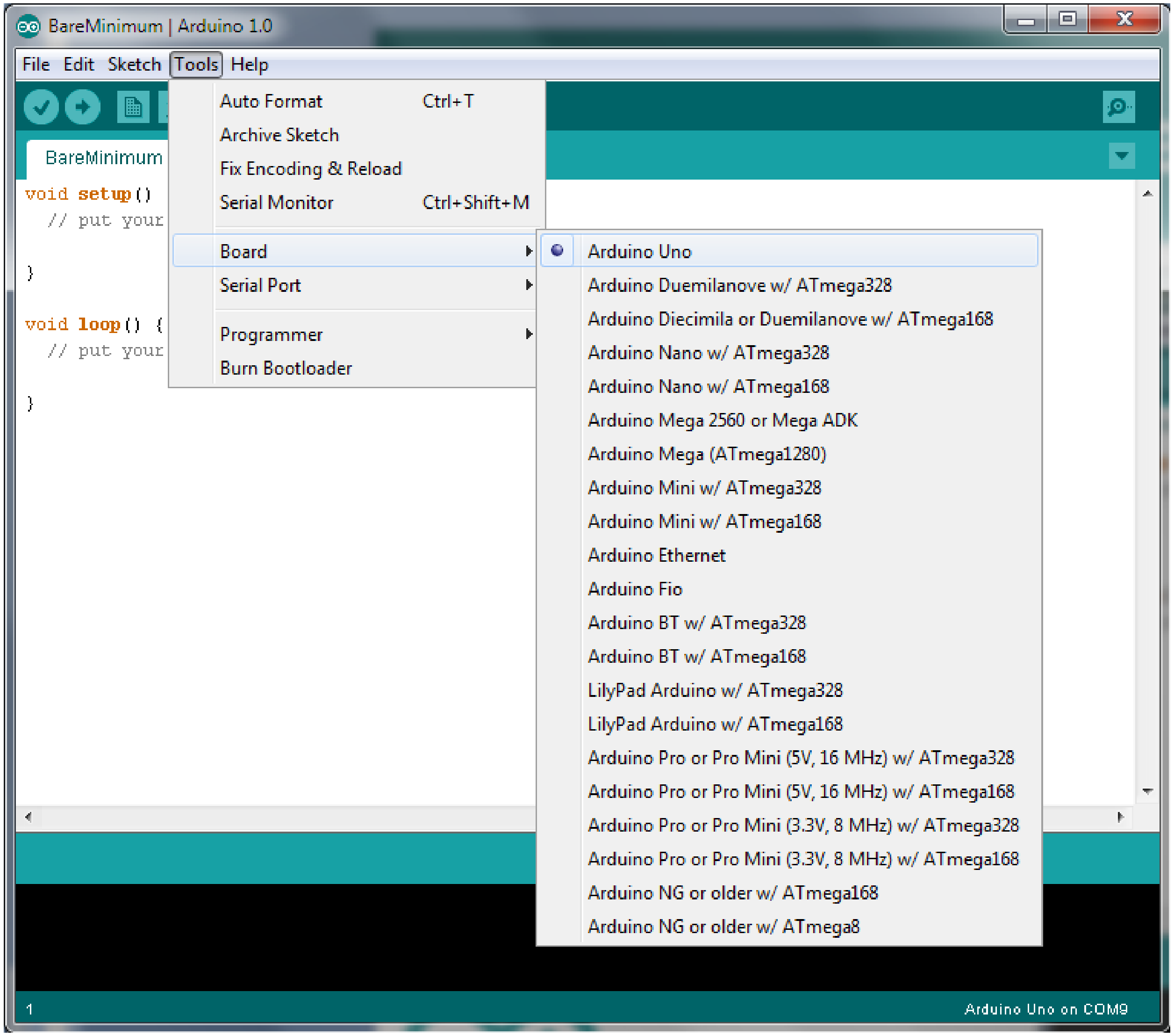
DC voltage supply (IN/OUT)

6 analog inputs

# Download and Install

- Download Arduino compiler and development environment from: http://arduino.cc/en/Main/Software
- Current version:  1.0.1
- Available for:
  - Windows
  - MacOX
  - Linux
- No installer needed... just unzip to a convenient location
- ***Before running Arduino***, plug in your board using USB cable (external power is not necessary)
- When USB device is not recognized, navigate to and select the appopriate driver from the installation directory
- Run Arduino

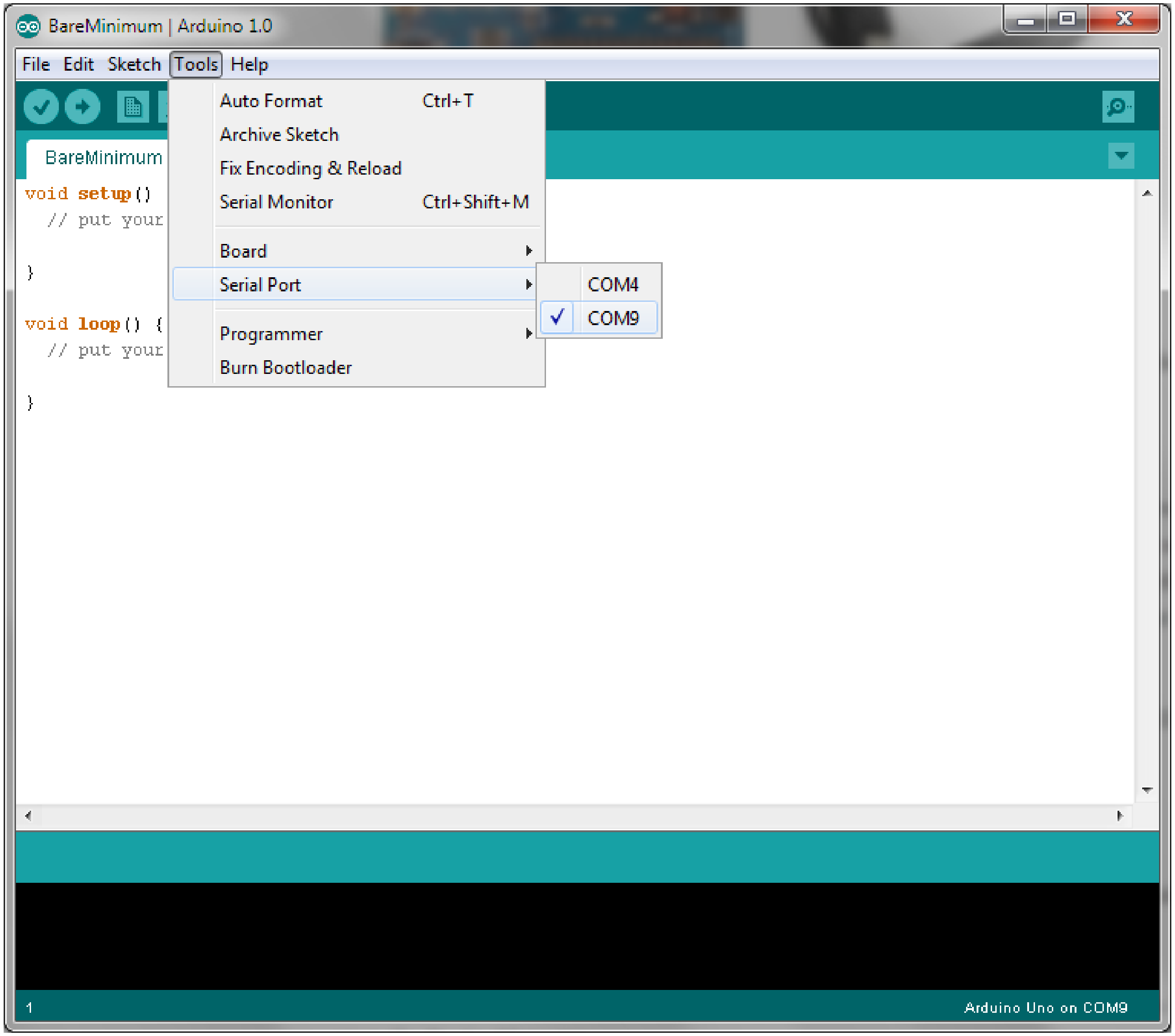
# Select your Board


BareMinimum | Arduino 1.0
File
Edit
Sketch
Tools
Help
BareMinimum
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Programmer
Burn Bootloader
● Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8

```
void setup()
  // put your

}

void loop() {
  // put your

}
```

```
1                                   Arduino Uno on COM9
```

# Select Serial Port



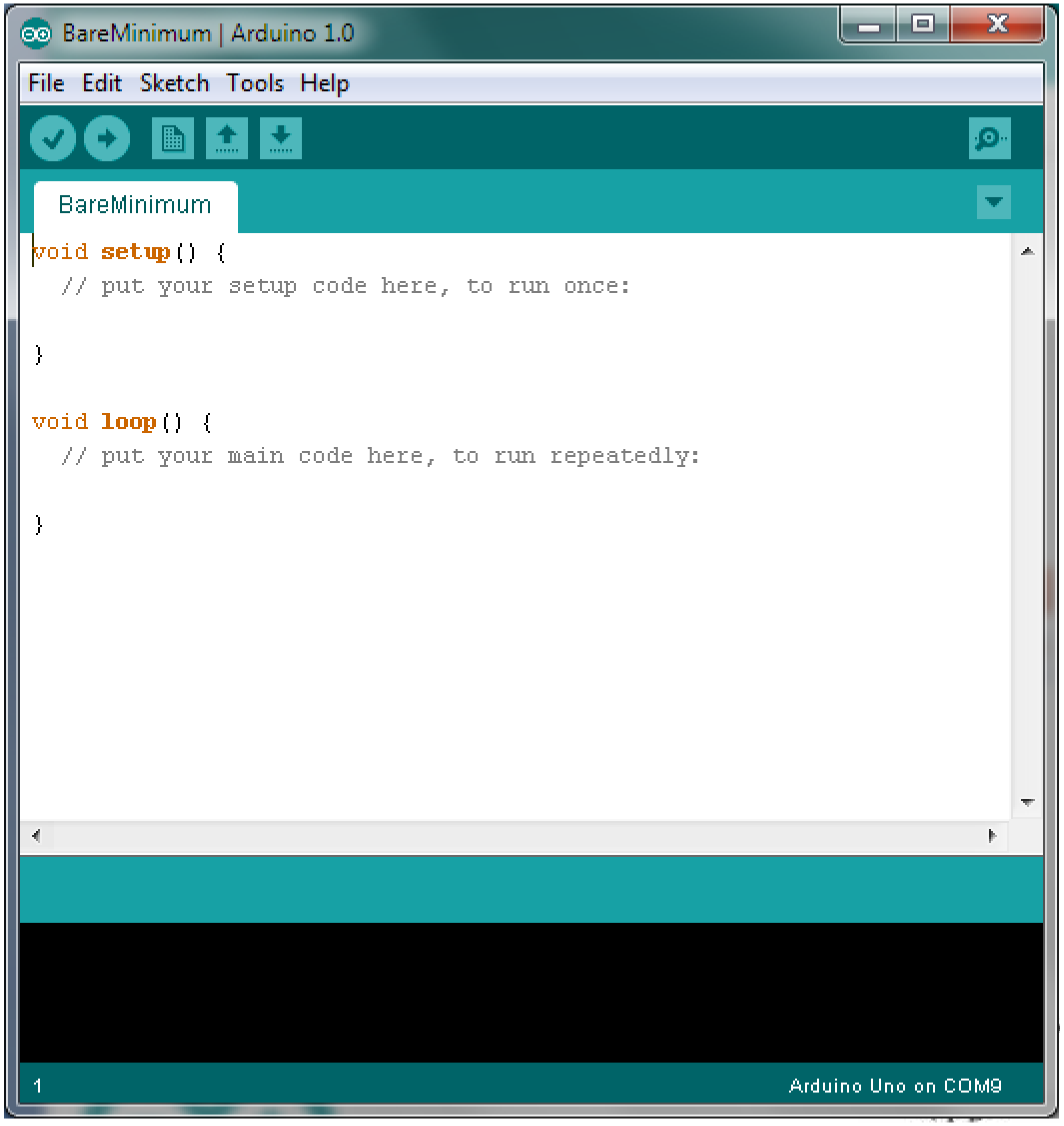
✓ COM9

```
void setup()
  // put your
}

void loop() {
  // put your
}
```

1 Arduino Uno on COM9

# Elements of the Arduino IDE

- Text editor
  - syntax and keyword coloring
  - automatic indentation
  - programming shortcuts
- Compiler
- Hardware Interface
  - Uploading programs
  - Communicating with Arduino via USB



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

# Using the Arduino IDE

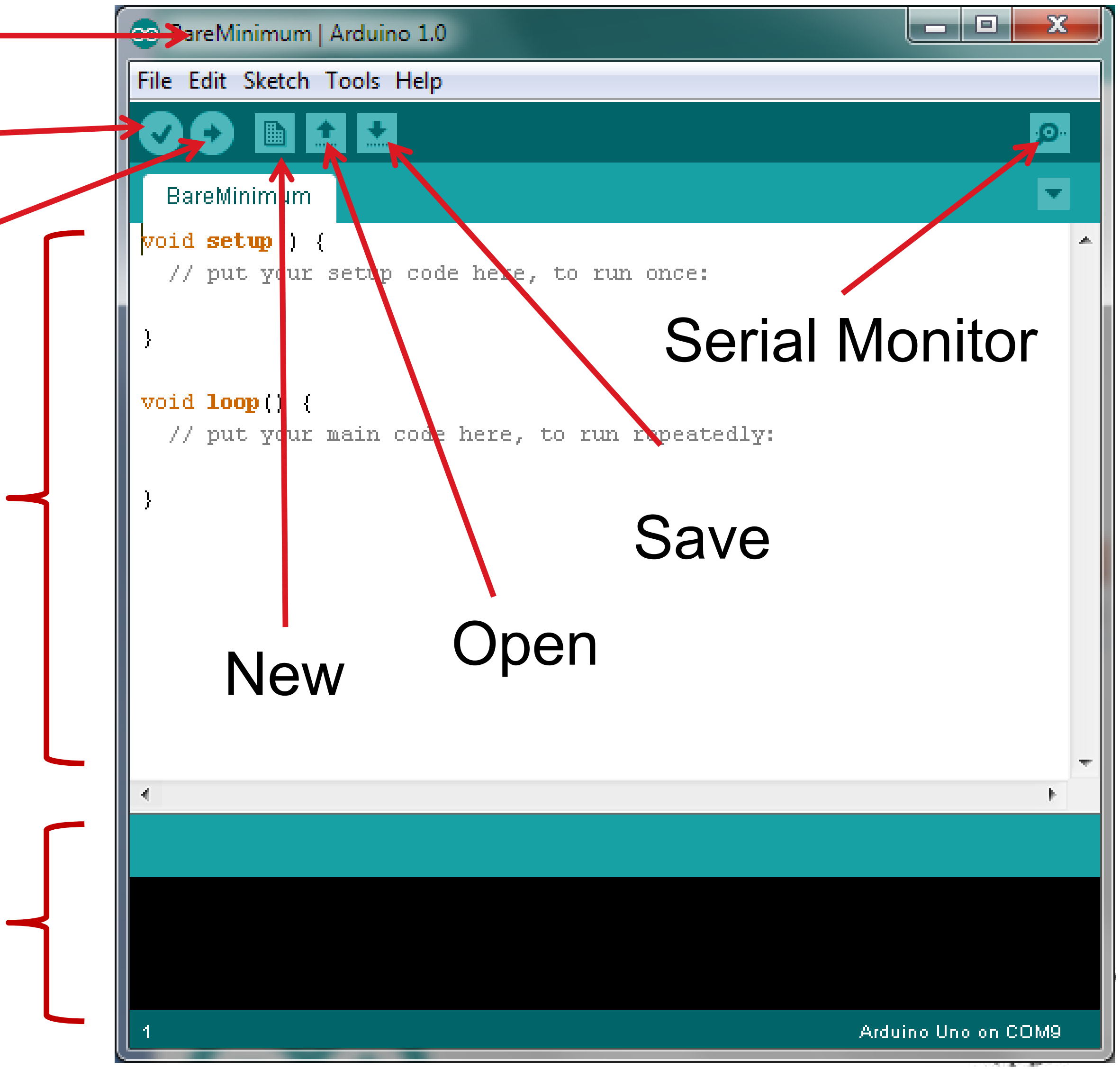
Name of sketch

Compile sketch

Upload to board

Program area

Messages / Errors



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

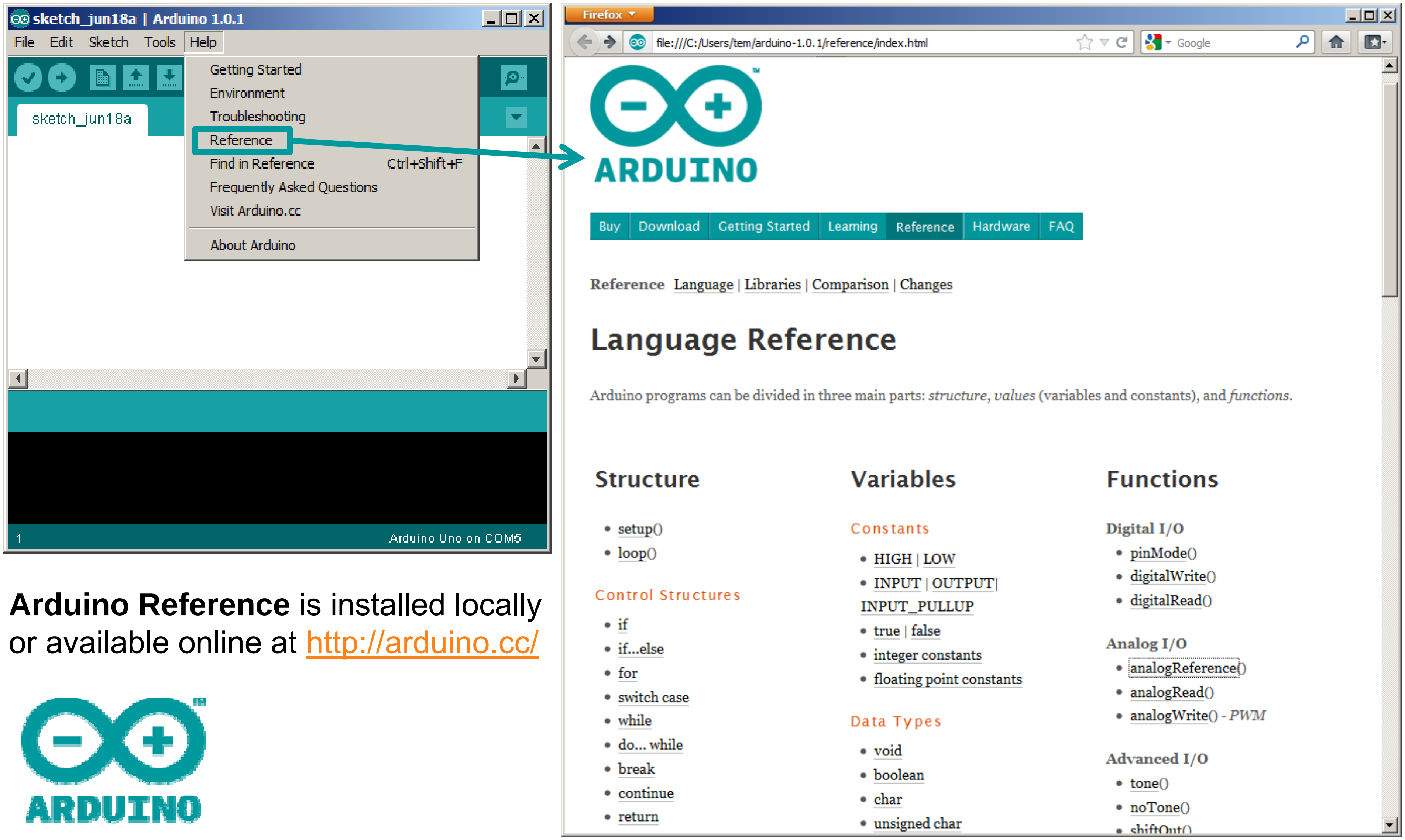
Serial Monitor

Save

Open

New

# Arduino Reference



**Reference** Language | Libraries | Comparison | Changes

## Language Reference

Arduino programs can be divided in three main parts: *structure*, *values* (variables and constants), and *functions*.

### Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return

### Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT| INPUT_PULLUP
- true | false
- integer constants
- floating point constants

Data Types

- void
- boolean
- char
- unsigned char

### Functions

**Digital I/O**

- pinMode()
- digitalWrite()
- digitalRead()

**Analog I/O**

- analogReference()
- analogRead()
- analogWrite() - *PWM*

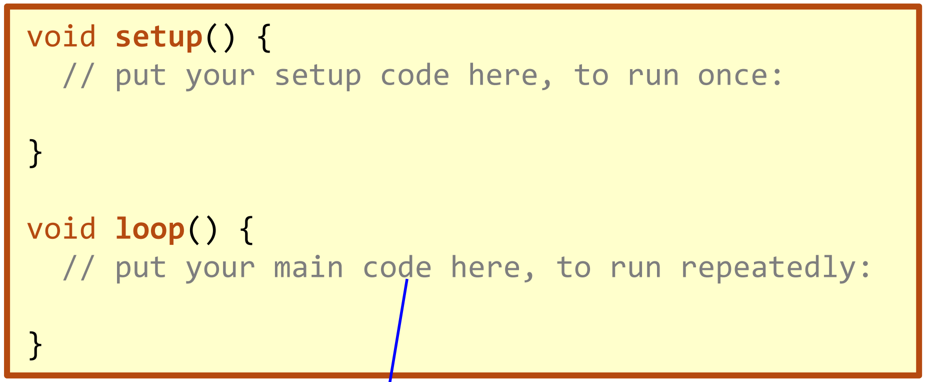
**Advanced I/O**

- tone()
- noTone()
- shiftOut()

**Arduino Reference** is installed locally or available online at http://arduino.cc/

# Arduino Sketch Structure

- `void setup()`
  - Will be executed only when the program begins (or reset button is pressed)
- `void loop()`
  - Will be executed repeatedly



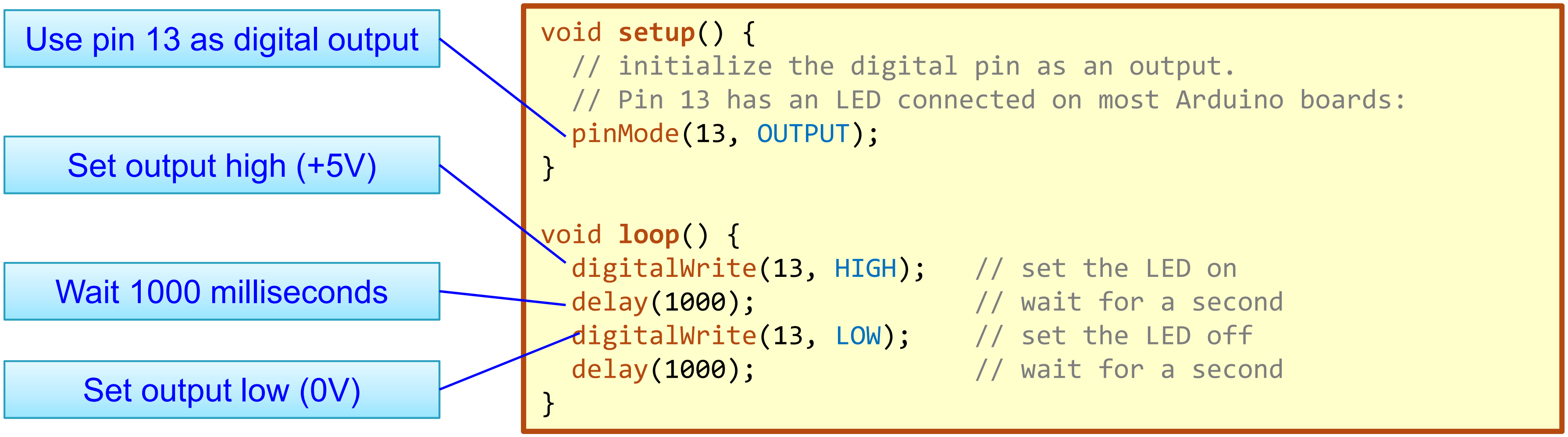
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Text that follows // is a comment (ignored by compiler)

Useful IDE Shortcut: Press Ctrl-/ to comment (or uncomment) a selected portion of your program.

# Activity 1: LED Blink

- Load the “Blink” example (File→Examples→Basics→Blink)



Use pin 13 as digital output

Set output high (+5V)

Wait 1000 milliseconds

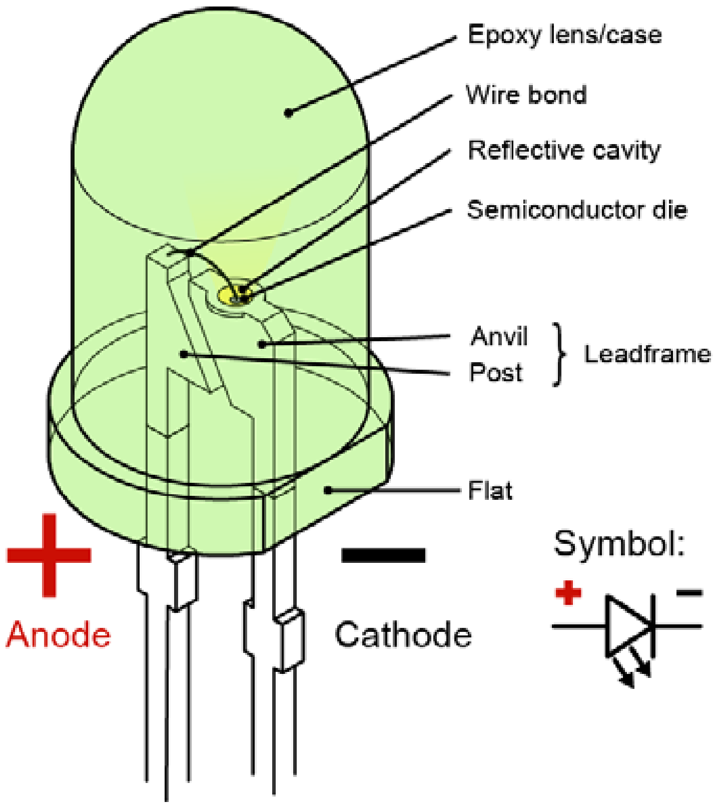
Set output low (0V)

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

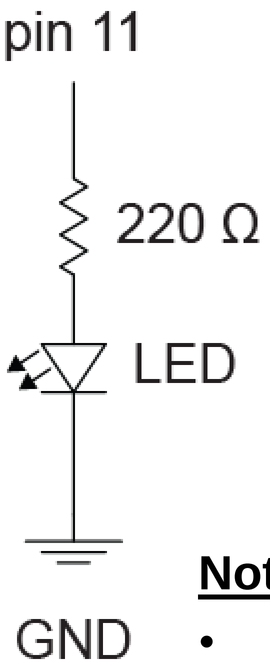
- Compile, then upload the program
- Congratulations!  you are now blinkers!

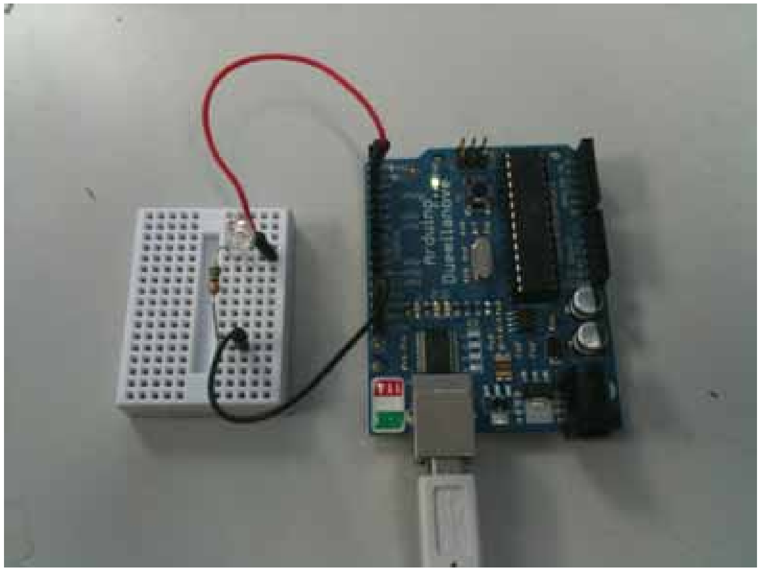
# Now connect your own LED

Anatomy of an LED:



[http://www.wikipedia.org/](http://www.wikipedia.org/)

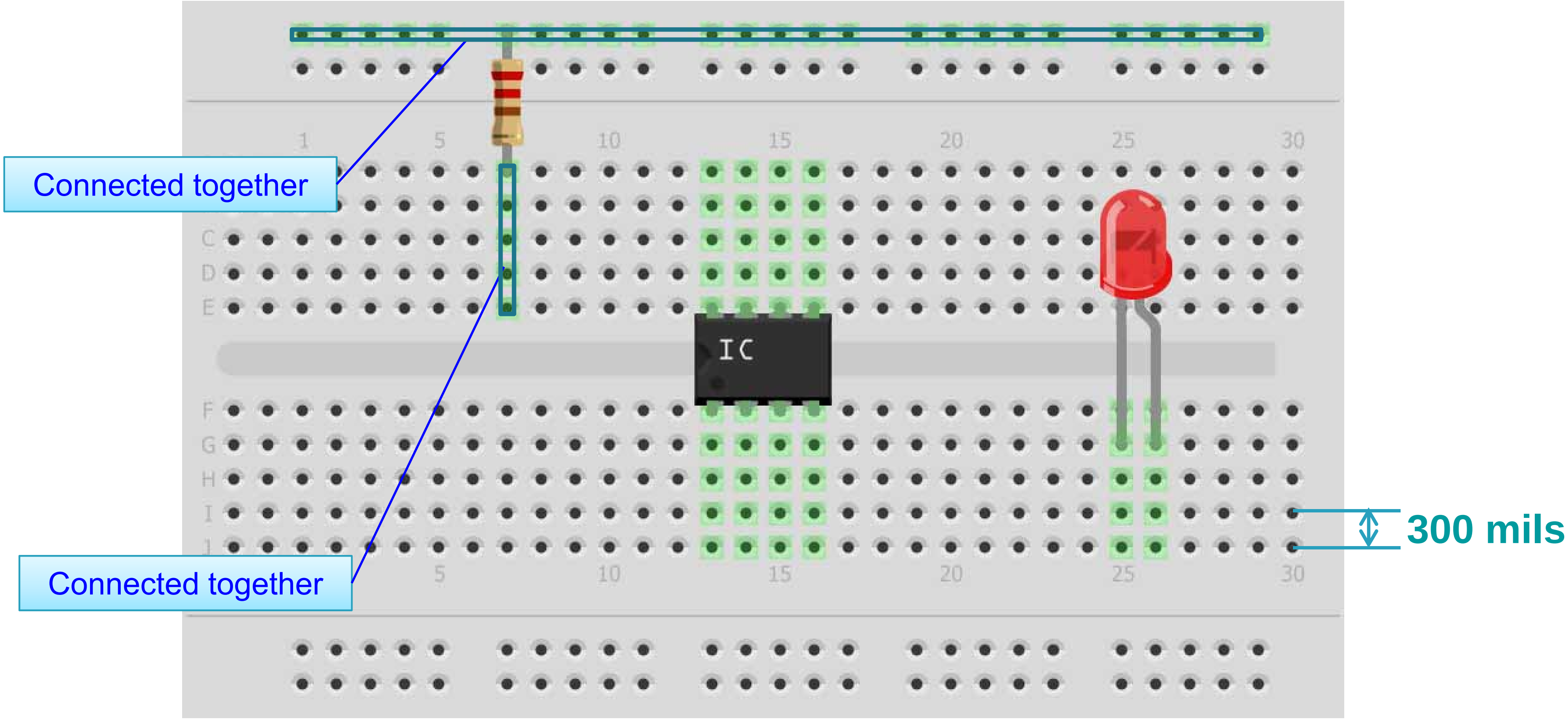




**Notes**:

- Resistor is needed to limit current
- Resistor and LED may be interchanged
  (but polarity of LED is important)
- Pin 13 is special: has built-in resistor and LED
- Change program and upload

# Aside: Using a Solderless Breadboard

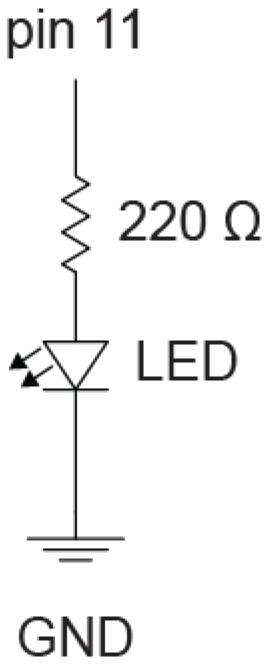


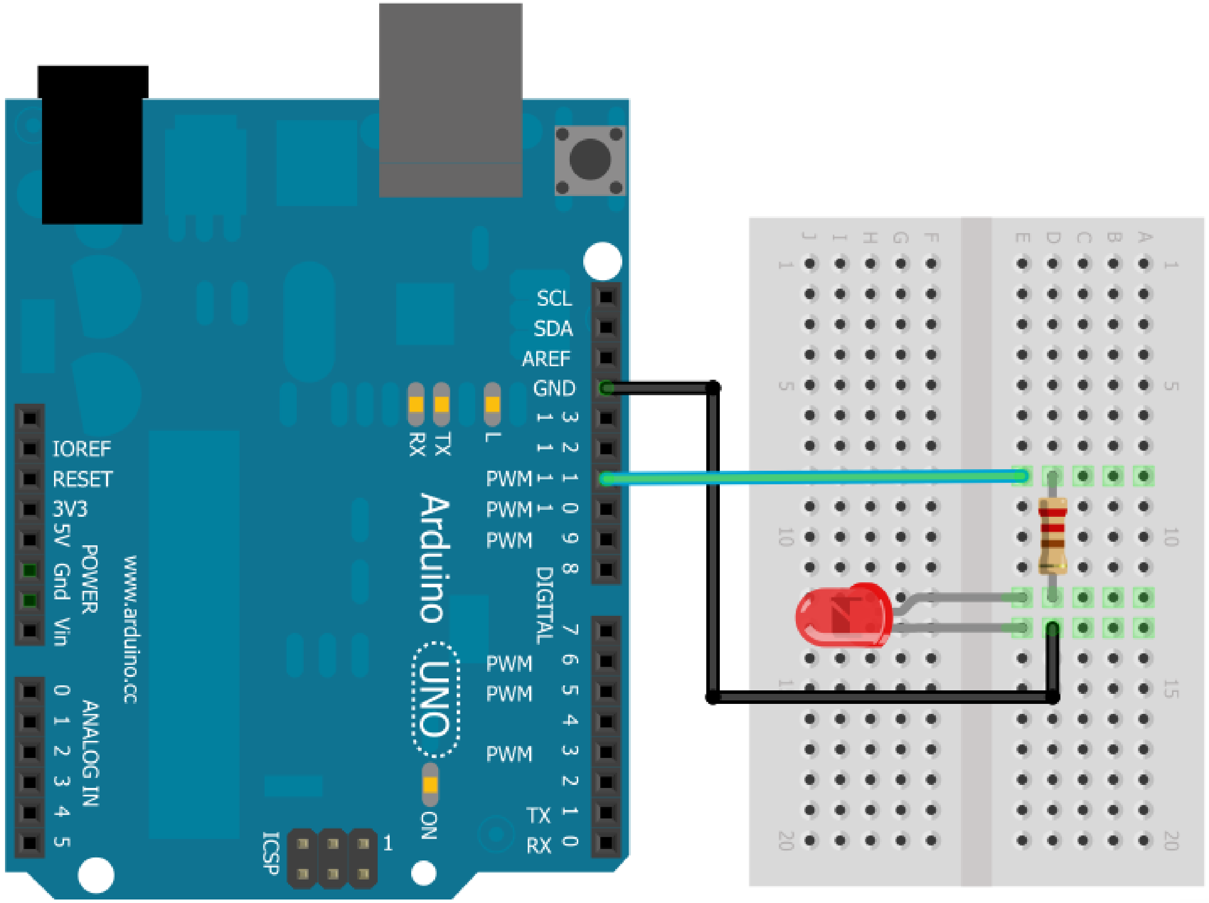
Connected together

Connected together

300 mils

# Example: Using a Solderless Breadboard

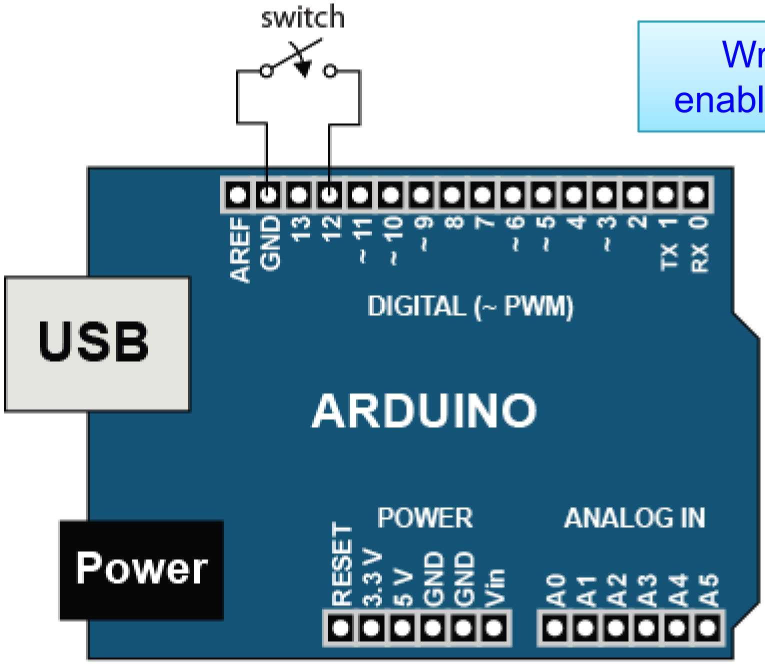




# Experimenting

- Change the blink rate
  - how fast can the LED blink (before you can no longer perceive the blinking?)
- How would you make the LED dimmer?
  - (...without changing the resistor?)

# Digital Input: Reading Switches and Buttons



Writing HIGH to an input pin: enables an internal pull-up resistor

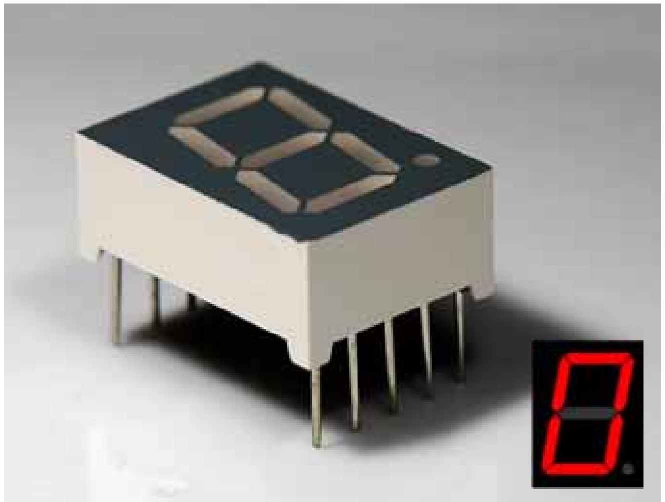
```
void setup() {
  pinMode(11, OUTPUT);      // Use pin 11 for digital out
  pinMode(12, INPUT);       // Use pin 12 for digital input
  digitalWrite(12, HIGH);   // Enable pull-up resistor
}

void loop() {
  boolean state;
  state = digitalRead(12);   // read state of pin 12
  digitalWrite(11, state);   // set state of pin 11 (LED)
  delay(100);                // wait for a 1/10 second
}
```

- Turn on/off LED based on switch
- Pin 12 reads `LOW` when switch is closed
- Pin 12 reads `HIGH` when switch is open (pull-up)

Without the internal pull-up resistor, unconnected digital inputs could read either high or low

# Activity 2: Seven-Segment Display

- Write a that program that counts from 0 to 9 and displays the result on a seven-segment LED display



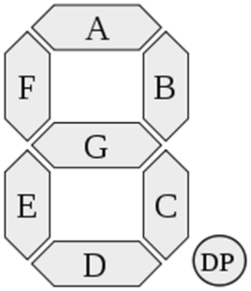
- Consider writing a function:

```
void writeDigit(int n)
```

that writes a single digit

# Seven-Segment Display Table

| Digit | ABCDEFG | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|
| **0** | 0×7E | on | on | on | on | on | on | off |
| **1** | 0×30 | off | on | on | off | off | off | off |
| **2** | 0×6D | on | on | off | on | on | off | on |
| **3** | 0×79 | on | on | on | on | off | off | on |
| **4** | 0×33 | off | on | on | off | off | on | on |
| **5** | 0×5B | on | off | on | on | off | on | on |
| **6** | 0×5F | on | off | on | on | on | on | on |
| **7** | 0×70 | on | on | on | off | off | off | off |
| **8** | 0×7F | on | on | on | on | on | on | on |
| **9** | 0×7B | on | on | on | on | off | on | on |



**Useful:**

- `bitRead(x,n)`
  Get the value of the n$^{th}$ bit of an integer x
  *Example:*
  - `bitRead(0x7E,7);   // returns 1 (see table above)`

# Serial Communication - Writing

**IMPORTANT**: USB serial communication is shared with Arduino pins 0 and 1 (RX/TX)

`Serial.begin(baud)`

Initialize serial port for communication (and sets baud rate)

*Example:*

```
– Serial.begin(9600); // 9600 baud
```

**Note**: `Serial.end()` command is usually unnecessary, unless you need to use pins 0 & 1

Format can be: BIN, HEX, OCT, or an integer specifying the number of digits to display

`Serial.print(val), Serial.print(val,fmt)`

Prints data to the serial port

*Examples:*

```
– Serial.print(“Hi”);       // print a string
– Serial.print(78);         // works with numbers, too
– Serial.print(variable);   // works with variables
– Serial.print(78,BIN);     // will print 1001110
```

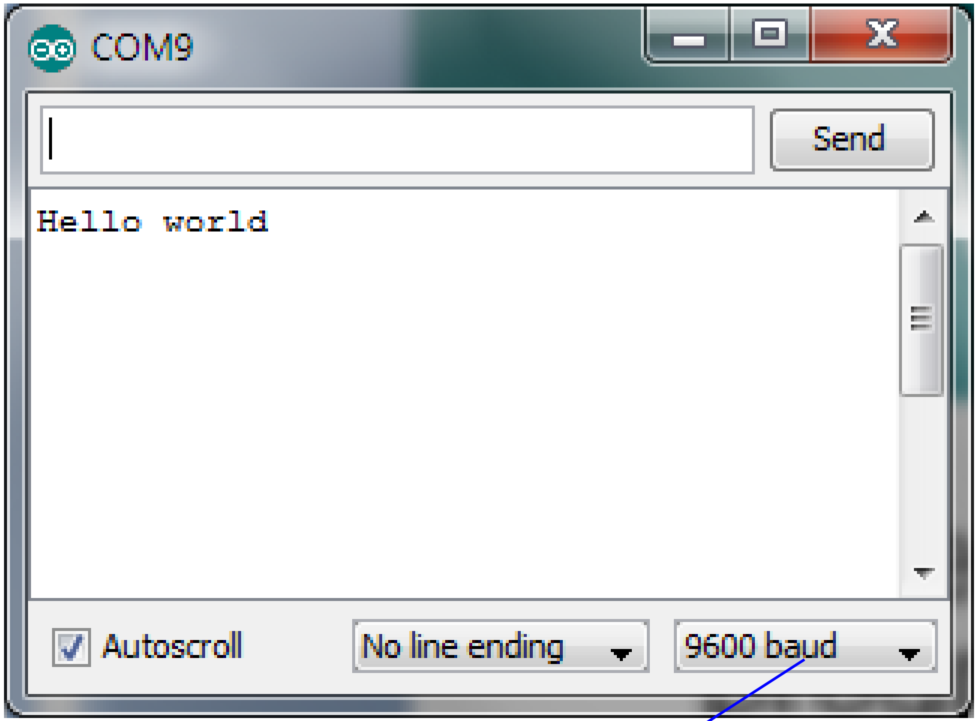
- `Serial.println(val)`

  Same as `Serial.print()`, but with line-feed

# Activity 3:  Hello World!

- Write an Arduino program that prints the message “Hello world” to the serial port
- ...whenever you press a switch/button
- Use the Serial Monitor to see the output (Ctrl-Shift-M)
- Try increasing baud rate

Serial Monitor:



```
Hello world
```

Make sure this agrees with your program,  i.e., Serial.begin(9600);

# Serial Communication - Reading

- `Serial.available()`

  Returns the number of bytes available to be read, if any

  *Example:*

  ```
  if (Serial.available() > 0) {
     data = Serial.read();
  }
  ```

To read data from serial port:

- `letter = Serial.read()`
- `letters = Serial.readBytesUntil(character, buffer, length)`
- `number = Serial.parseInt()`
- `number = Serial.parseFloat()`

# Activity 4 – User Controlled Blinker

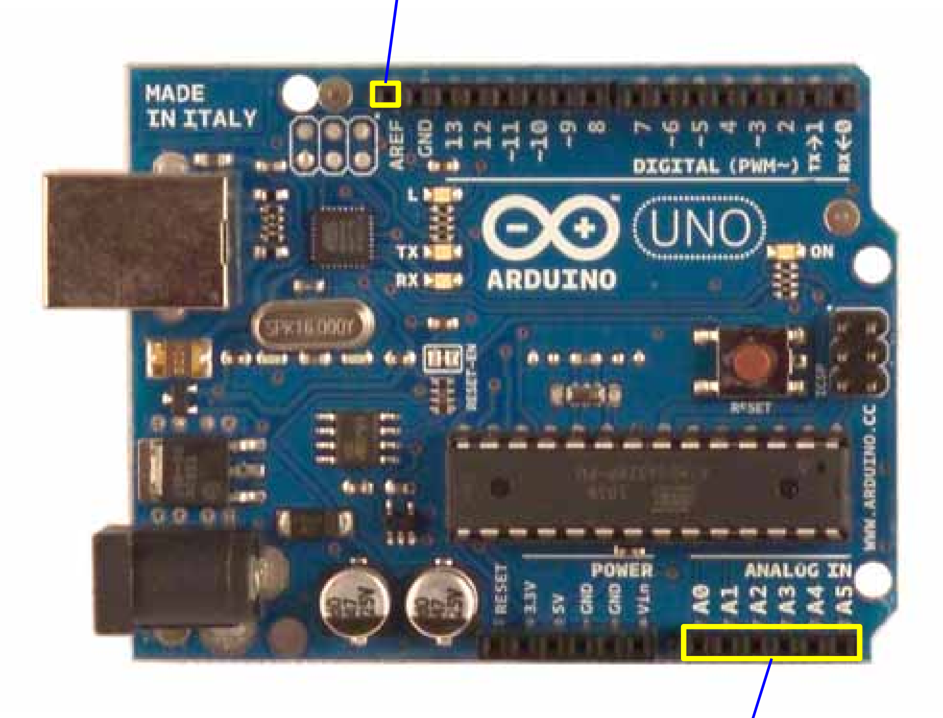
- When available (`Serial.available`), read an integer from the serial port (`Serial.parseInt`), and use the result to change the blink rate of the LED (pin 13)

**Useful:**

- `constrain(x,a,b)`
  Constrains the variable x to be from a to b
  *Examples:*
  - `constrain(5,1,10);   // returns 5`
  - `constrain(50,1,10);  // returns 10`
  - `constrain(0,1,10);   // returns 1`

# Analog Input and Sensors

Reference Voltage (optional)



Analog Inputs

- Six analog inputs:
  `A0, A1, A2, A3, A4, A5`
- `AREF` = Reference voltage (default = +5 V)
- 10 bit resolution:
  - returns an integer from 0 to 1023
  - result is proportional to the pin voltage
- All voltages are measured relative to `GND`

**Note**: If you need additional digital I/O, the analog pins can be re-assigned for digital use:
`pinMode(A0, OUTPUT);`

# Reading Analog Values

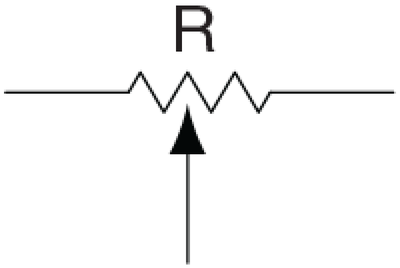
- `value = analogRead(pin)`
  `Reads the analog measurement on pin`
  `Returns integer between 0 and 1023`

- `analogReference(type)`
  type can be:
  - `DEFAULT` - the default analog reference of 5 volts (on 5V Arduino boards)
  - `INTERNAL` – Built-in reference voltage (1.1 V)
  - `EXTERNAL` – AREF input pin

**Note**: Do **NOT** use `pinMode(A0, INPUT)` unless you want to use A0 for **DIGITAL** input.

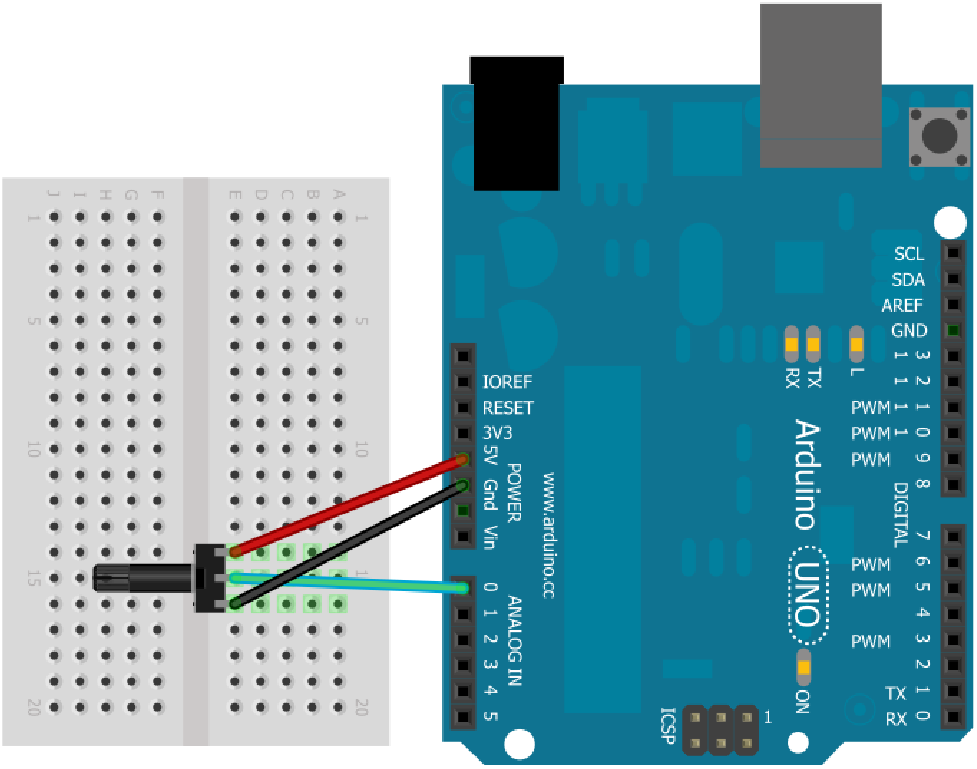
# Aside: Potentiometers

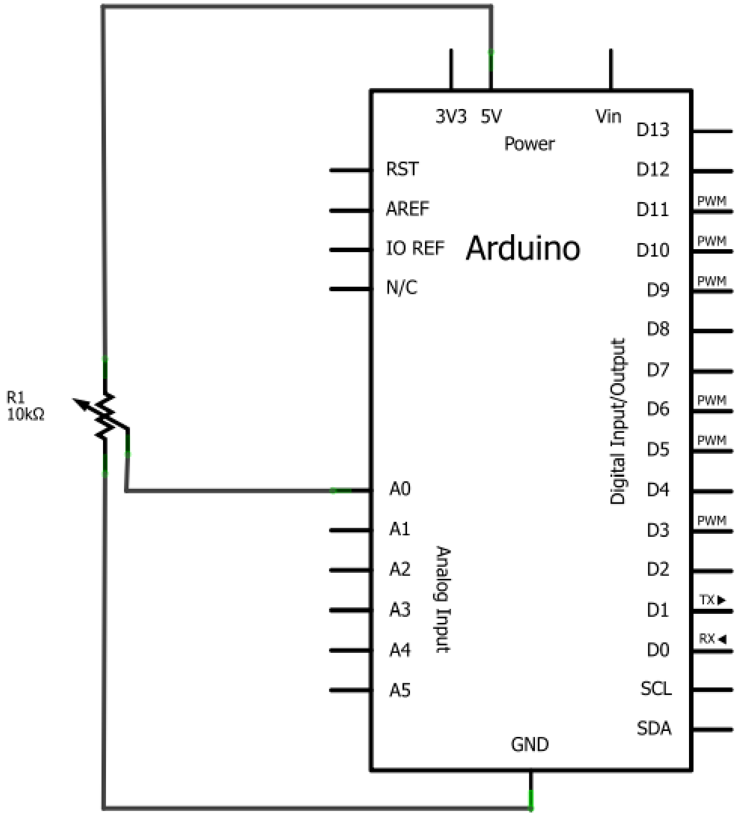
## (variable resistors, rheostats)



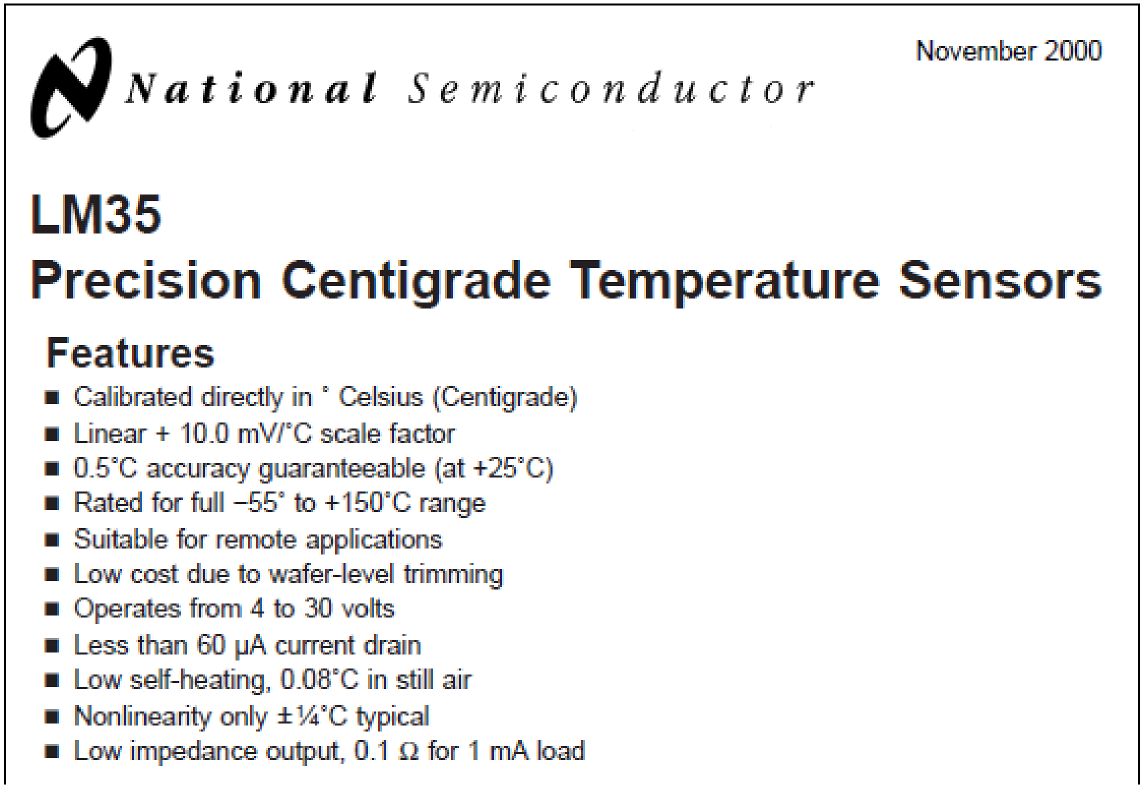
# Activity 5 – Volume Knob

- Connect the potentiometer from 5V to GND
- Use `analogRead(A0)` to measure the voltage on the center pin
- Set the LED blink rate depending on the reading





# Activity 6 – Arduino Thermometer



National Semiconductor

November 2000

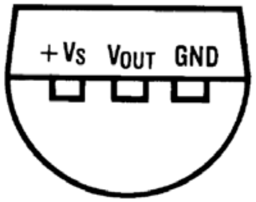
## LM35

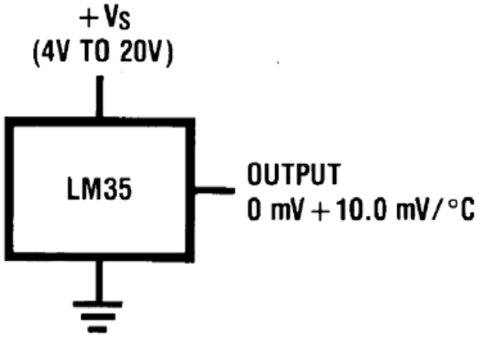
## Precision Centigrade Temperature Sensors

### Features

- Calibrated directly in ˚ Celsius (Centigrade)
- Linear + 10.0 mV/˚C scale factor
- 0.5˚C accuracy guaranteeable (at +25˚C)
- Rated for full −55˚ to +150˚C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08˚C in still air
- Nonlinearity only ±¼˚C typical
- Low impedance output, 0.1 Ω for 1 mA load

TO-92
Plastic Package





- Build a circuit and write a sketch to read and report the temperature at 1 second intervals

# Data Logging Ideas

- `millis()`
  Returns the number of milliseconds elapsed since program started (or reset)

Time functions

**Note**: this uses the Time library:
`#include <Time.h>`

- `setTime(hr,min,sec,day,month,yr)`
- `hour(), minute(), day(), month(), year()`

Real-time Clock (RTC):

- Use an external, battery-powered chip (e.g., DS1307) to provide clock

# Activity 7 – Arduino Nightlight

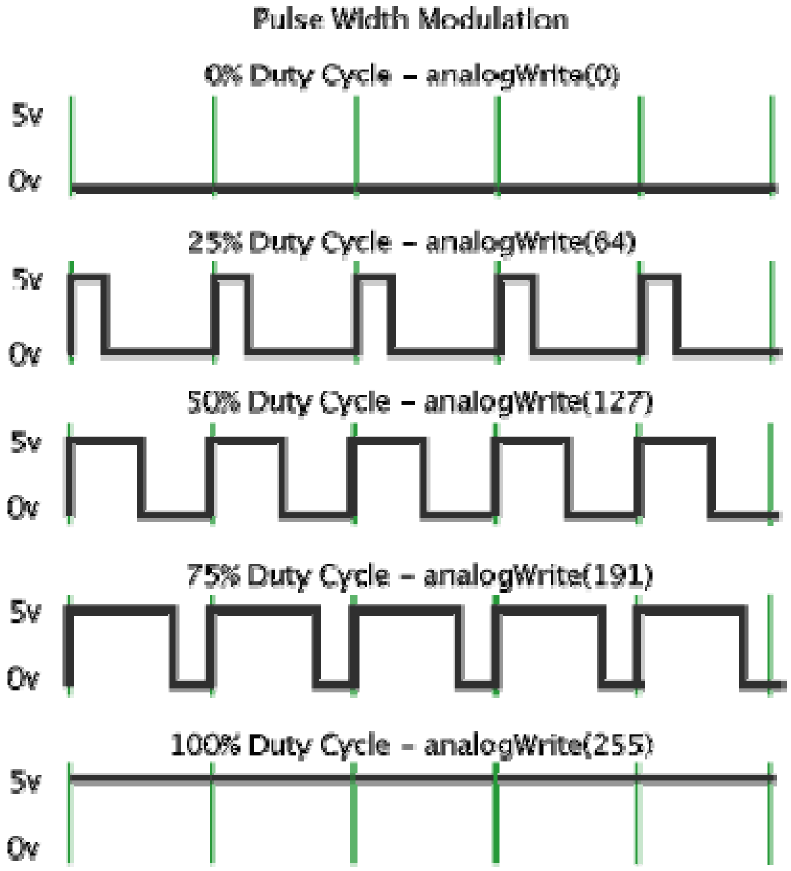
- CdS Photoresistor:
  resistance depends on ambient light level

- Build a circuit and write a sketch that turns on an LED whenever it gets dark

  *Hint:* connect the photoresistor in a voltage divider

# Analog Output?

- Most microcontrollers have only digital outputs
- **Pulse-width Modulation**: Analog variables can be represented by the duty-cycle (or pulse-width) of a digital signal

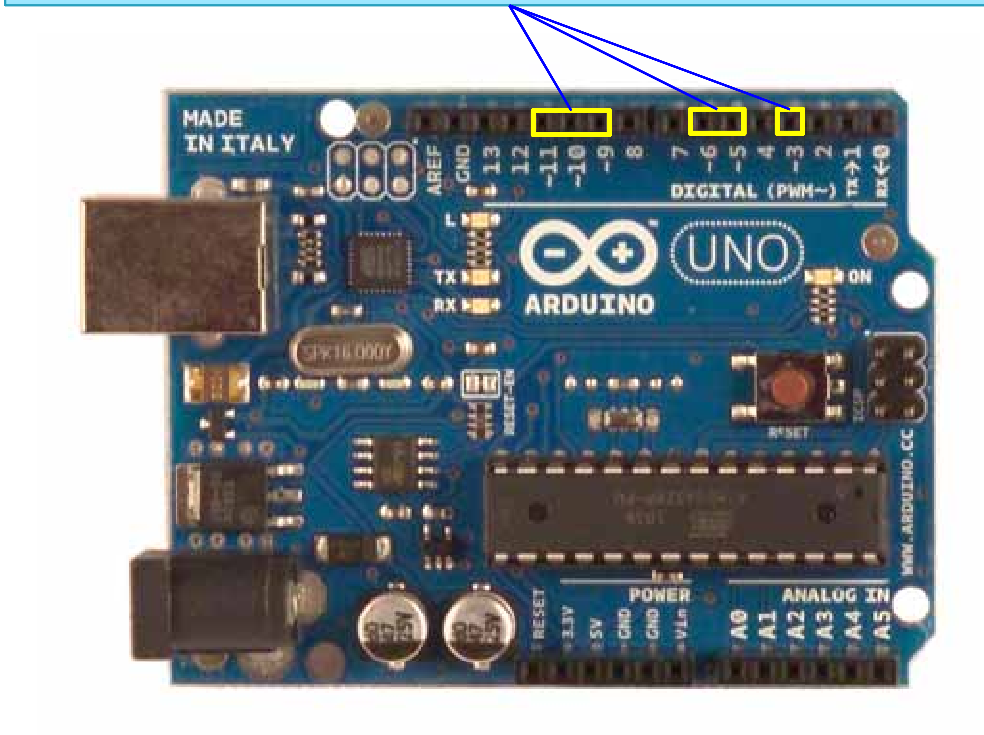


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
25% Duty Cycle – analogWrite(64)
50% Duty Cycle – analogWrite(127)
75% Duty Cycle – analogWrite(191)
100% Duty Cycle – analogWrite(255)
5v
0v

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

# PulseWidth Modulation (PWM)

PWM available on pins 3, 5, 6, 9, 10, 11



- `analogWrite(pin,val)` set the PWM fraction:
  - `val = 0`: always off
  - `val = 255`: always on
- Remember to designate pin for digital output: `pinMode(pin,OUTPUT);` *(usually in setup)*
- Default PWM frequency:
  - **16 MHz / $2^{15}$ = 488.28125 Hz**

**Note**: the PWM frequency and resolution can be changed by re-configuring the timers

# Activity 8 – PWM LED Dimmer

- Use PWM to control the brightness of an LED
  - connect LED to pin 3, 5, 6, 9, 10 or 11
  - remember to use 220 Ω current-limiting resistor
- Set the brightness from the serial port, or potentiometer
- Watch the output on an oscilloscope

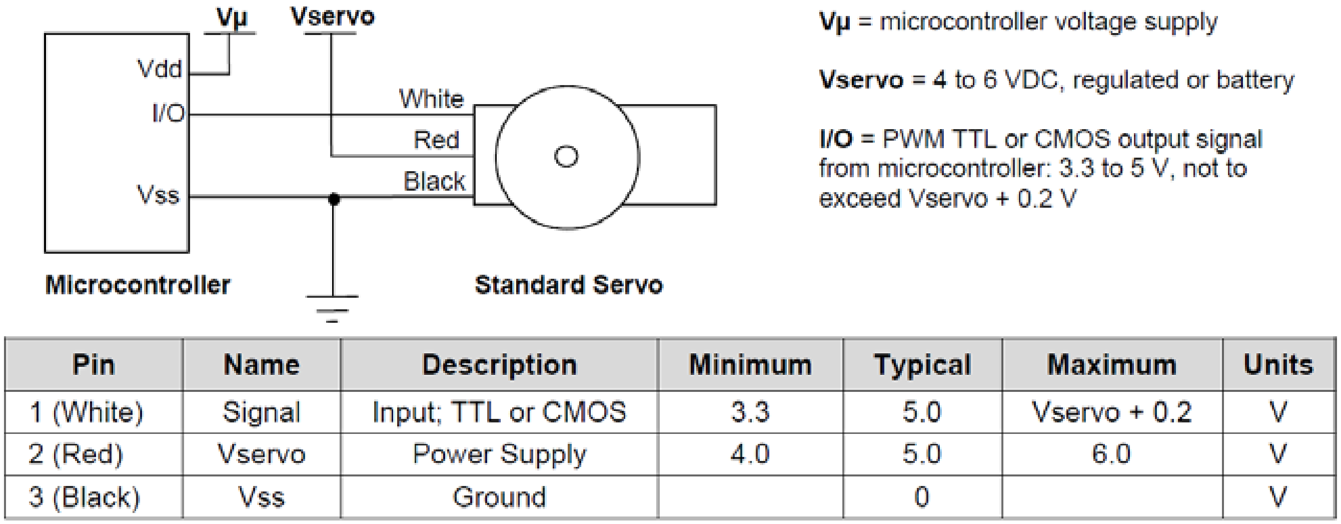
Useful:

- `newValue = map(oldValue, a, b, c, d)`
  Converts/maps a number in the range (`a:b`) to a new number in the range (`c:d`)
  *Example:*
  - `newValue = map(oldValue,0,1023,0,255);`

# Activity 8 – PWM LED Dimmer (cont’d)

- Change your program to sinusoidally modulate the intensity of the LED, at a 1 Hz rate
  - – *Hint*: use the `millis()`, `sin()` , and `analogWrite()` functions

# Servomotors



**Vμ** = microcontroller voltage supply

**Vservo** = 4 to 6 VDC, regulated or battery

**I/O** = PWM TTL or CMOS output signal from microcontroller: 3.3 to 5 V, not to exceed Vservo + 0.2 V

| Pin | Name | Description | Minimum | Typical | Maximum | Units |
|---|---|---|---|---|---|---|
| 1 (White) | Signal | Input; TTL or CMOS | 3.3 | 5.0 | Vservo + 0.2 | V |
| 2 (Red) | Vservo | Power Supply | 4.0 | 5.0 | 6.0 | V |
| 3 (Black) | Vss | Ground | | 0 | | V |

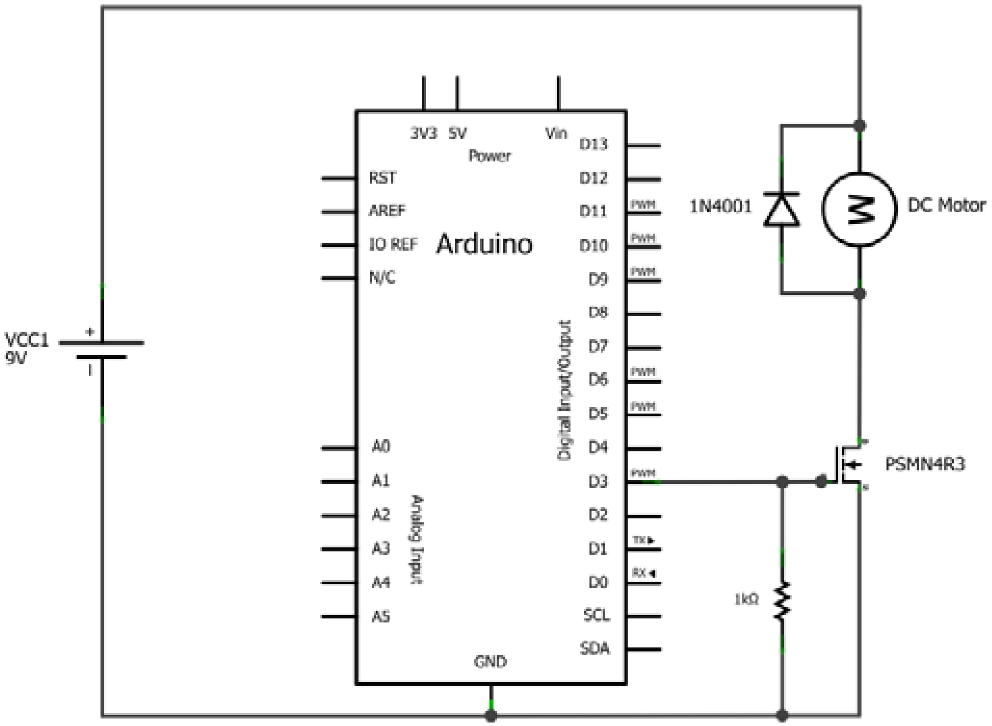
http://www.parallax.com/

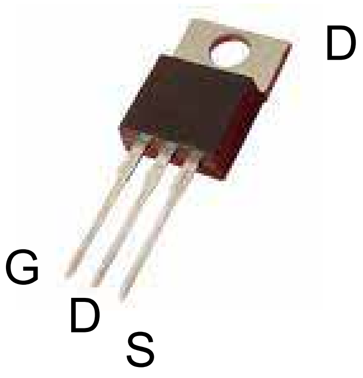
- Standard servo:
  - PWM duty cycle controls direction:
  - 0% duty cycle → 0 degrees
  - 100% duty cycle → 180 degrees
- Continuous-rotation servo:
  - duty cycle sets speed and/or direction

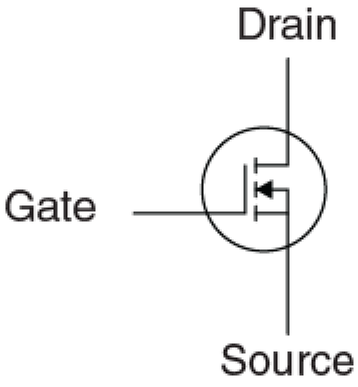
# Activity 9 – Servomotor Control

- Build a program that turns a servomotor from 0 to 180 degrees, based on potentiometer reading
- Report setting to the serial monitor

# Solid State Switching - MOSFETs





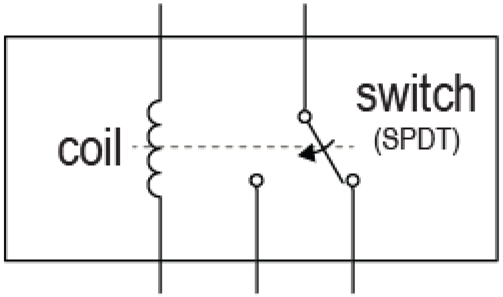


- Logic-level MOSFET (requires only 5 V)
- Acts like a voltage-controlled switch
- Works with PWM!

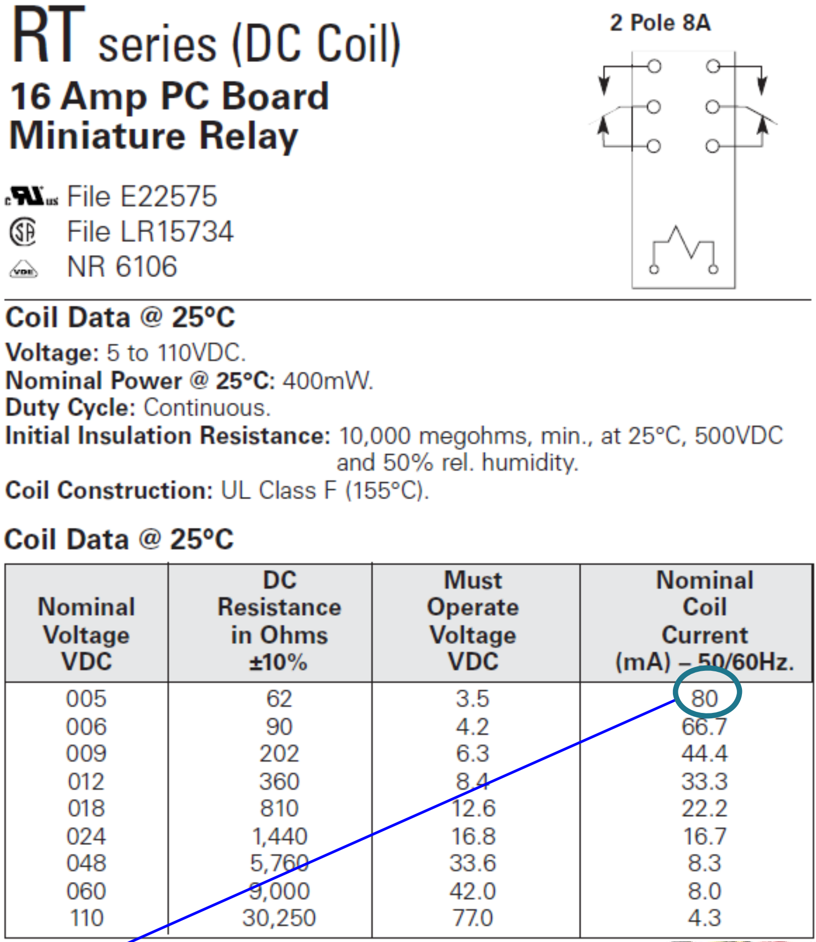
# Activity 10 – PWM Speed Control

- Build a circuit to control the speed of a motor using a PWM-controlled MOSFET
- Enter the speed (PWM setting) from the serial port (`Serial.parseInt`)

# Controlling Relays and Solenoids



- Electromechanically-actuated switch
- Provides electrical isolation
- Typically few ms response time



RT series (DC Coil)

**16 Amp PC Board Miniature Relay**

2 Pole 8A

File E22575
File LR15734
NR 6106

**Coil Data @ 25°C**

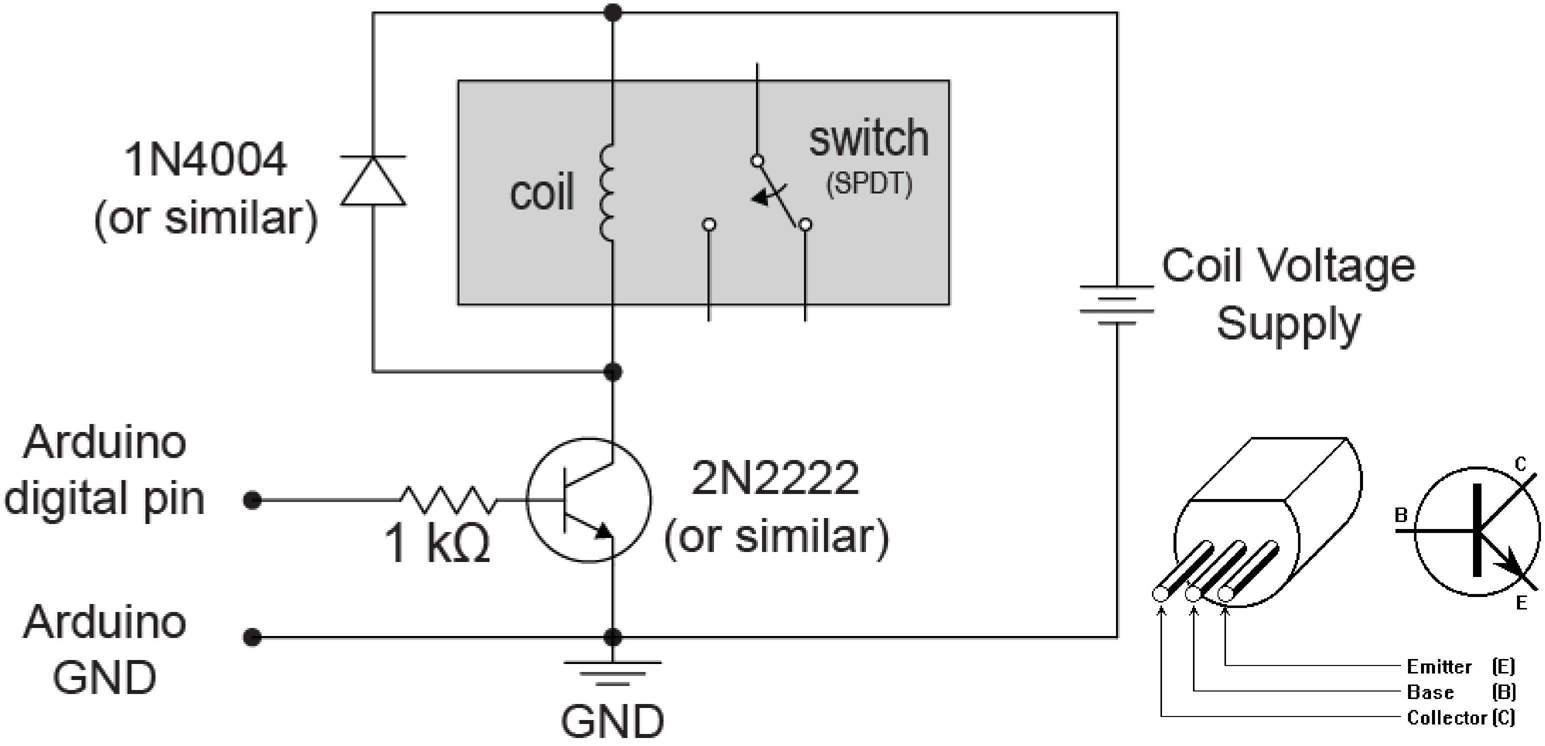
**Voltage:** 5 to 110VDC.
**Nominal Power @ 25°C:** 400mW.
**Duty Cycle:** Continuous.
**Initial Insulation Resistance:** 10,000 megohms, min., at 25°C, 500VDC and 50% rel. humidity.
**Coil Construction:** UL Class F (155°C).

**Coil Data @ 25°C**

| Nominal Voltage VDC | DC Resistance in Ohms ±10% | Must Operate Voltage VDC | Nominal Coil Current (mA) – 50/60Hz. |
|---|---|---|---|
| 005 | 62 | 3.5 | 80 |
| 006 | 90 | 4.2 | 66.7 |
| 009 | 202 | 6.3 | 44.4 |
| 012 | 360 | 8.4 | 33.3 |
| 018 | 810 | 12.6 | 22.2 |
| 024 | 1,440 | 16.8 | 16.7 |
| 048 | 5,760 | 33.6 | 8.3 |
| 060 | 9,000 | 42.0 | 8.0 |
| 110 | 30,250 | 77.0 | 4.3 |

Note: Arduino cannot supply enough current to drive relay coil

# Relay Driver Circuit

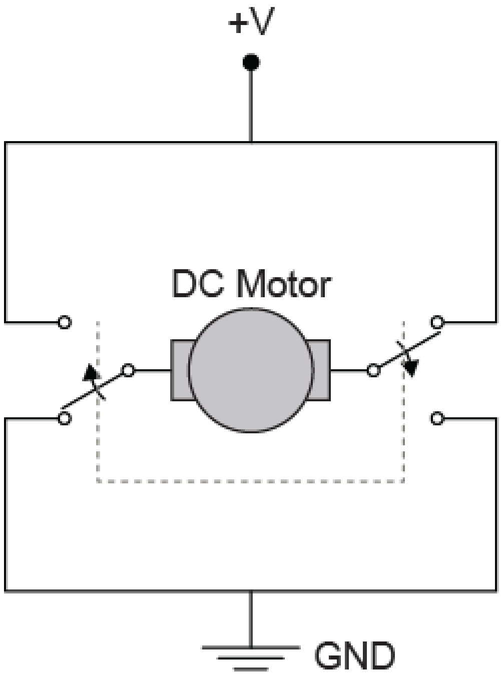


- NPN transistor: acts like a current-controlled switch
- MOSFET will also work
- Diode prevents back-EMF (associated with inductive loads)
- Coil voltage supply and Arduino share common GND

# Activity 11: Bidirectional Motor Driver

- Build a circuit (and write an Arduino sketch) that will use a DPDT relay to change the direction of a DC motor:

**Note**: this is called an H-bridge circuit. It can also be made with transistors



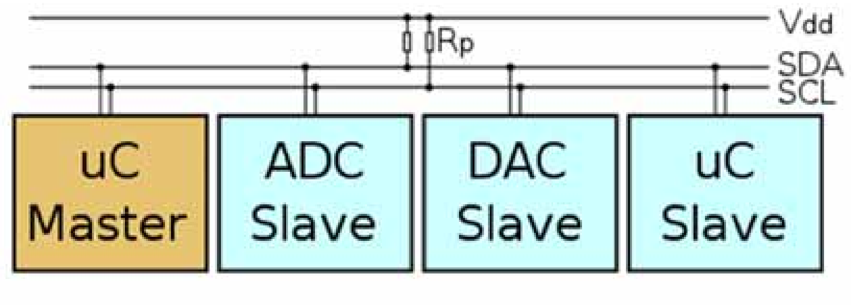
(**Note**: control coil not shown)

# Communication: $I^2C$, SPI

- $I^2C$ (Inter-Integrated Circuit)
  - Developed by Phillips
  - Speed = 100 kHz, 400 kHz, and 3.4 MHz *(not supported by Arduino)*
  - Two bi-directional lines: **SDA, SCL**
  - Multiple slaves can share same bus
- SPI (Serial Peripheral Interface Bus)
  - Speed = 1-100 MHz (clock/device limited)
  - Four-wire bus: **SCLK, MOSI, MISO, SS**
  - Multiple slaves can share same bus (but each needs a dedicated **SS**, slave select)

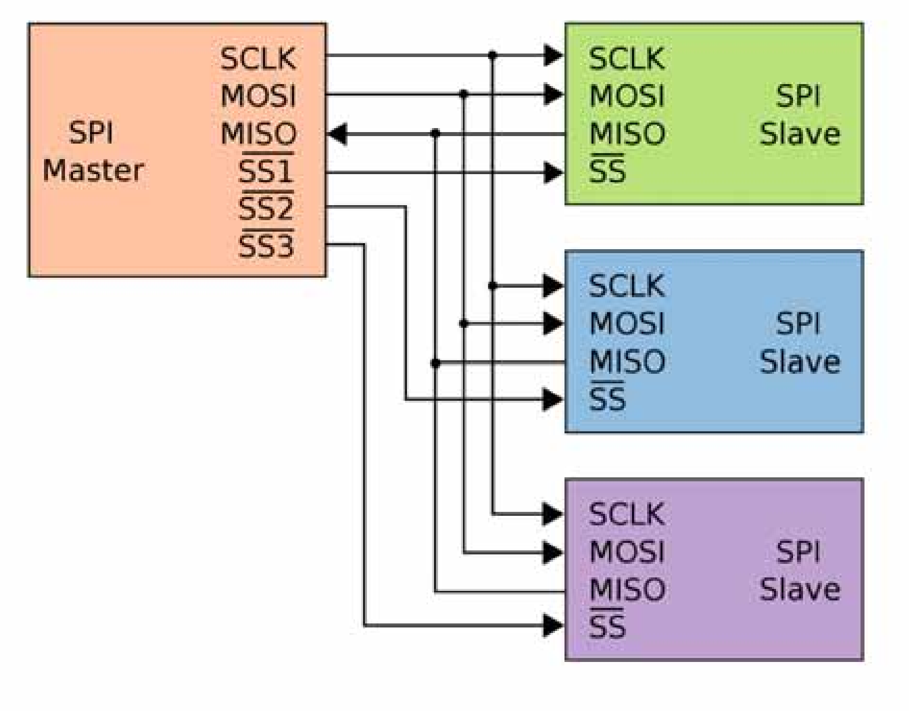
# Connecting Multiple Devices ($I^2C$ and SPI)

Master (µC) with three $I^2C$ slaves:

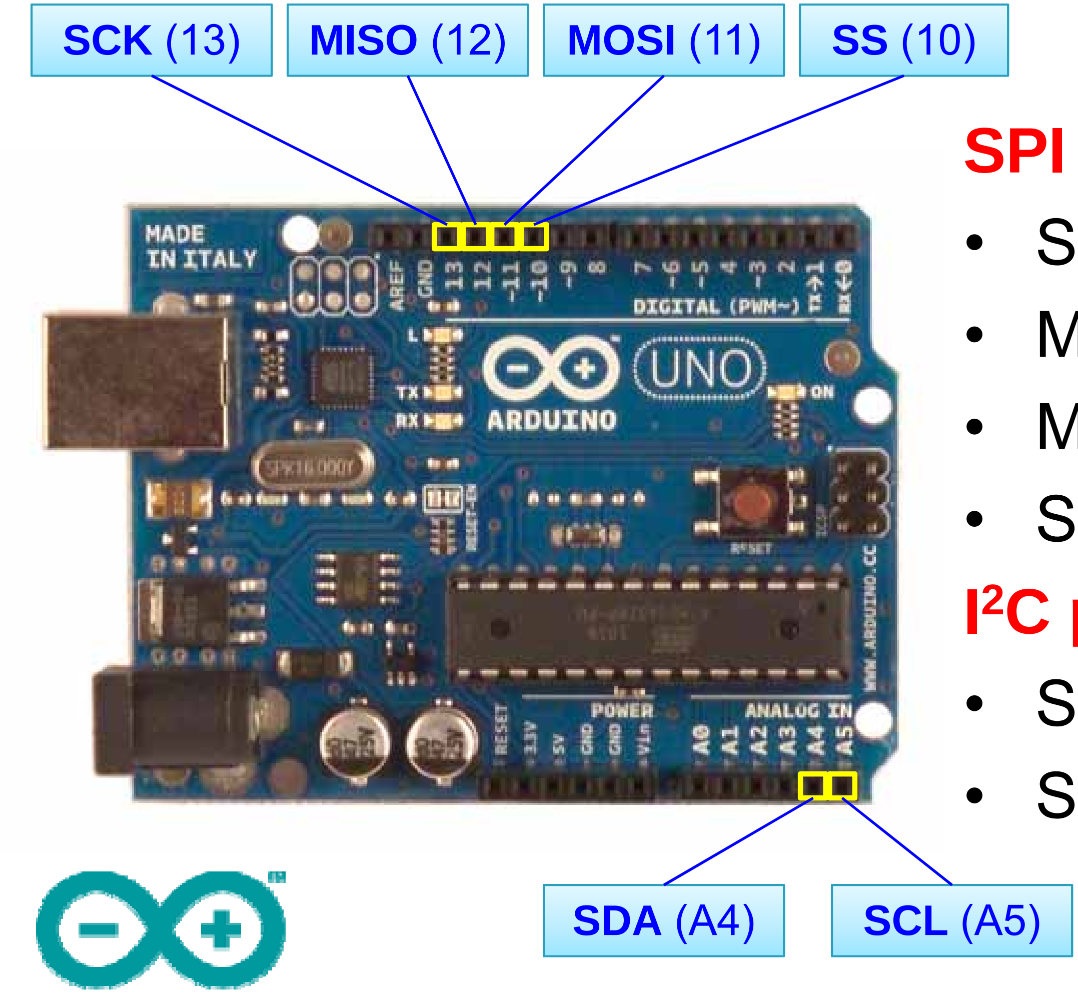


http://en.wikipedia.org/

Master with three SPI slaves:



# SPI and I$^2$C on the Arduino



**SCK** (13)

**MISO** (12)

**MOSI** (11)

**SS** (10)

**SDA** (A4)

**SCL** (A5)

**SPI pins**:

- SCK = serial clock
- MISO = master in, slave out
- MOSI = master out slave in
- SS = slave select

**I$^2$C pins**:

- SDA = data line
- SCL = clock line

# Basic Arduino I²C Commands

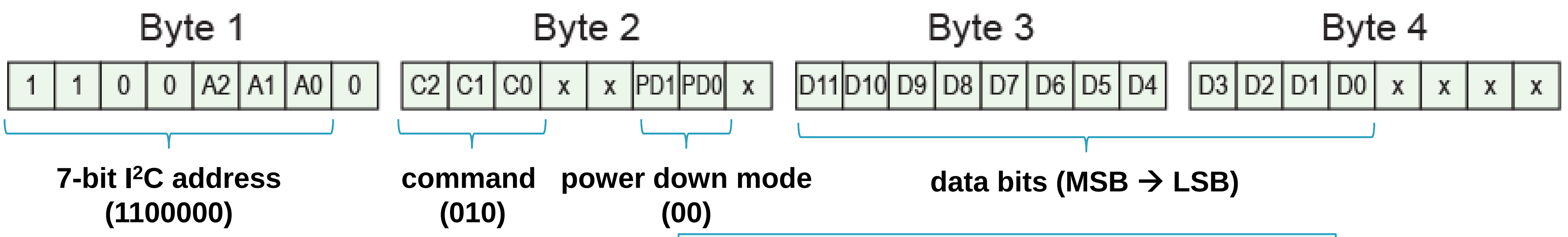
| COMMAND | EXPLANATION |
|---|---|
| `Wire.begin()` | Join the I²C bus as master (usually invoked in `setup`) |
| `Wire.beginTransmission(address)` | Begin communicating to a slave device |
| `Wire.write(byte)` | Write one byte to I²C bus (after request) |
| `Wire.endTransmission(address)` | End transmission to slave device |

**Note**: you must include the Wire library:
`#include <Wire.h>`

**Note**: `pinMode()` not needed for I²C on pins A4 and A5

# Example: MCP4725 12-bit DAC

MCP4725 write command (taken from data sheet)



| Byte 1 | | | | | | | | Byte 2 | | | | | | | | Byte 3 | | | | | | | | Byte 4 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | A2 | A1 | A0 | 0 | C2 | C1 | C0 | x | x | PD1 | PD0 | x | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | x | x | x | x |

**7-bit I$^2$C address (1100000)**

**command (010)**

**power down mode (00)**

**data bits (MSB → LSB)**

**Note**: binary numbers are preceded by B:
B1100000 = 96

Arduino program segment:

`data >> 4`: shift bits left by four positions

```
Wire.beginTransmission(B1100000);     // Byte 1 (Initiate communication)
Wire.write(B01000000);                // Byte 2 (command and power down mode)
Wire.write(data >> 4);                // Byte 3 (send bits D11..D4)
Wire.write((data & B00001111) << 4); // Byte 4 (send bits D3..D0)
Wire.endTransmission();
```

**Remember**: you must include the Wire library at the top:
`#include <Wire.h>`
and you must also use `Wire.begin()` in setup

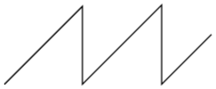
# Additional I$^2$C Commands

| COMMAND | EXPLANATION |
|---|---|
| `Wire.begin()` | Join the I$^2$C bus as master (usually invoked in `setup`) |
| `Wire.begin(address)` | Join the I$^2$C bus as slave, with address specified (usually invoked in `setup`) |
| `Wire.beginTransmission(address)` | Begin communicating to a slave device |
| `Wire.write(byte)` | Write one byte to I$^2$C bus (after request) |
| `Wire.write(bytes,length)` | Write `length` bytes to I$^2$C bus |
| `Wire.endTransmission(address)` | End transmission to slave device |
| `Wire.requestFrom(address, quantity)`<br>`Wire.requestFrom(address, quantity, stop)` | Request bytes (quantity) from slave |
| `Wire.available()` | The number of bytes available for reading |
| `Wire.read()` | Reads a byte that was transmitted from a slave. (Preceded by `Wire.requestFrom`) |

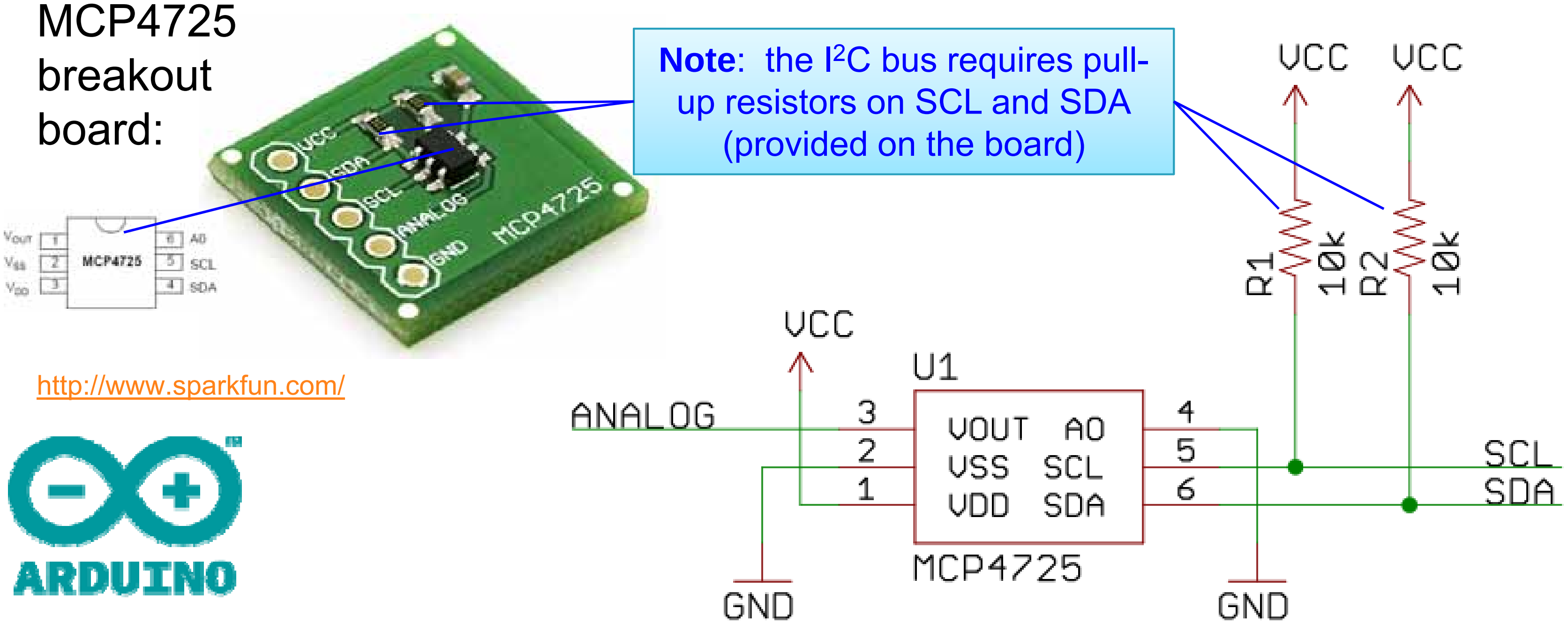
**Note**: you must include the Wire library:
`#include <Wire.h>`

**Note**: `pinMode()` not needed for I$^2$C on pins A4 and A5

# Activity 12:  Sawtooth Wave

- Program the MCP4725 DAC to produce a sawtooth (ramp) wave: 
  - What is the frequency of the sawtooth wave?
  - Can you make f = 100 Hz?

MCP4725 breakout board:



**Note**:  the I$^2$C bus requires pull-up resistors on SCL and SDA (provided on the board)

http://www.sparkfun.com/

# Basic Arduino SPI Commands

| COMMAND | EXPLANATION |
|---|---|
| `SPI.begin()` | Initializes the SPI bus, setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low and SS high. |
| `byteIn = SPI.transfer(byteOut)` | Transfer one byte (both send and receive) returns the received byte |

**Note**: you must include the SPI library:

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

**Note**: `pinMode()` not needed. It is automatically configured in `SPI.begin()`

# Additional Arduino SPI Commands

| COMMAND | EXPLANATION |
|---|---|
| `SPI.begin()` | Initializes the SPI bus, setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low and SS high. |
| `SPI.end()` | Disables the SPI bus (leaving pin modes unchanged) – in case you need to use pins 10-13 again |
| `SPI.setBitOrder(order)` | Set bit order for SPI<br>`order` = {`LSBFIRST`, `MSBFIRST`} |
| `SPI.setClockDivider(divider)` | Set the SPI clock divider<br>`divider` = {2, 4, 8, 16, 32, 64, 128}<br>SPI clock speed = 16 MHz/divider |
| `SPI.setDataMode(mode)` | Set the SPI data mode<br>`mode` = {`SPI_MODE0`, `SPI_MODE1`, `SPI_MODE2`, `SPI_MODE3`} |
| `SPI.transfer(byte)` | Transfer one byte (both send and receive)<br>returns the received byte |

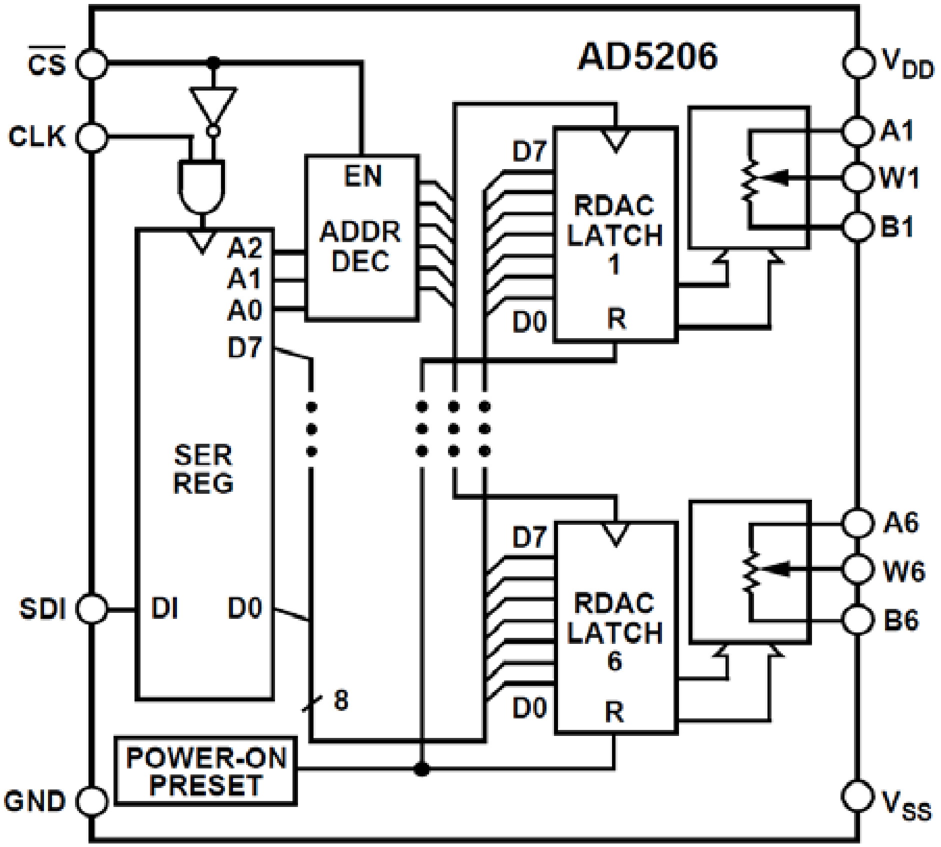
**Note**: you must include the SPI library:

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

**Note**: `pinMode()` not needed

# Example: AD5206 Digital Potentiometer

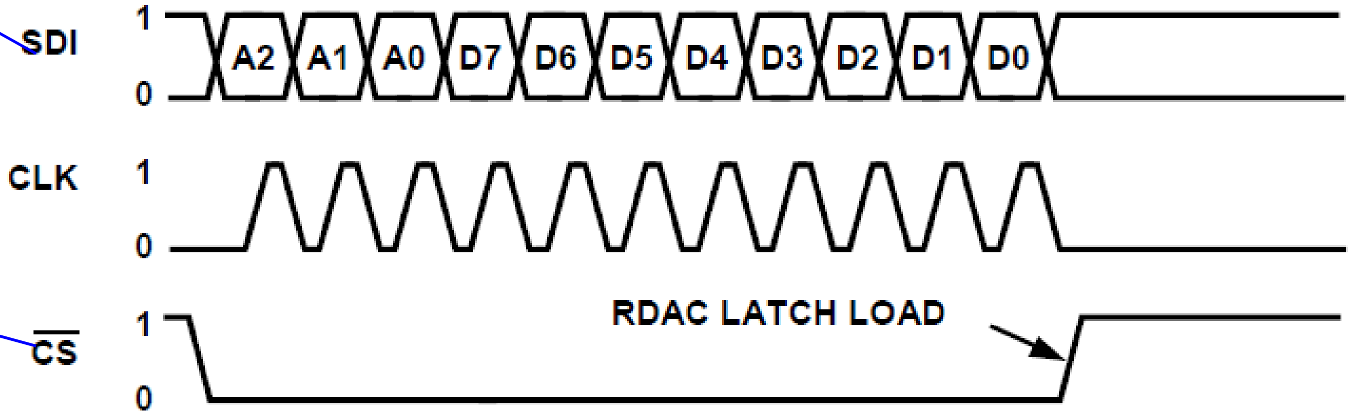
Functional block diagram:



## Features:

- six independent, 3-wiper potentiometers
- 8-bit precision (256 possible levels)
- Available in 10kΩ, 50kΩ and 100kΩ
- Programmed through SPI interface

# AD5206 Write Sequence



**Note**: same as MOSI (master out slave in)

**Note**: same as SS (slave select)

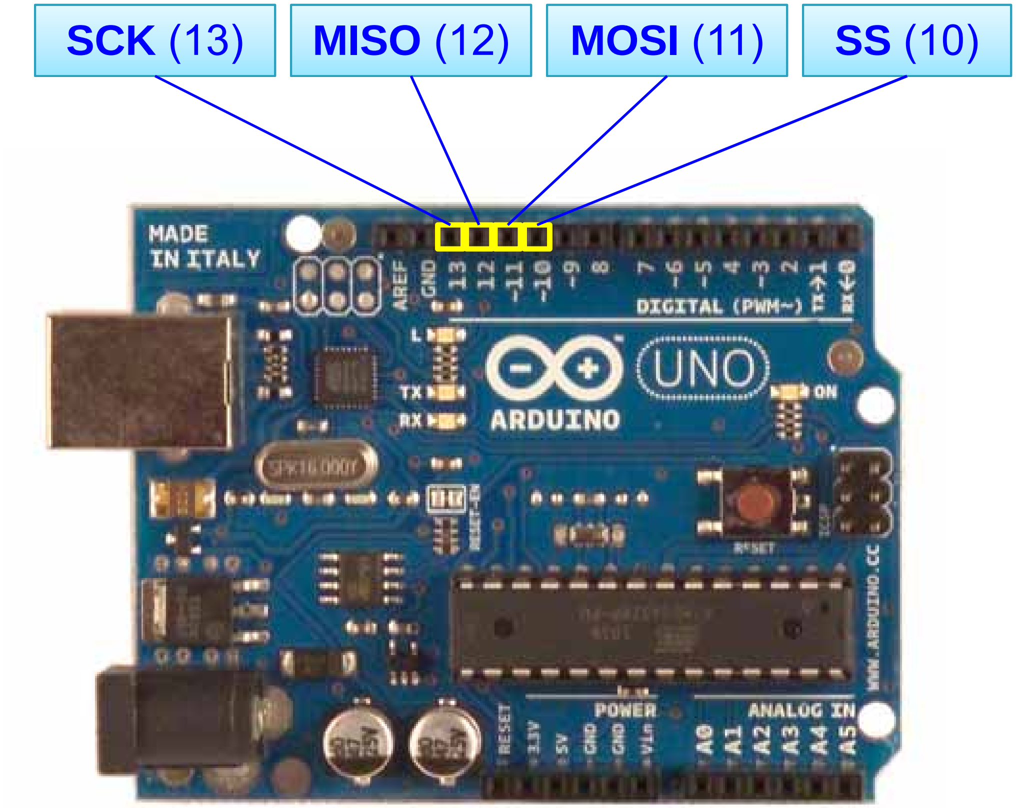
Arduino program segment:

```
SPI.begin();                 // initialize SPI (in setup)
...
digitalWrite(SS,LOW);        // hold SS pin low to select chip
SPI.transfer(potnumber);     // determine which pot (0..5)
SPI.transfer(wipervalue);    // transfer 8-bit wiper setting
digitalWrite(SS,HIGH);       // de-select the chip
```

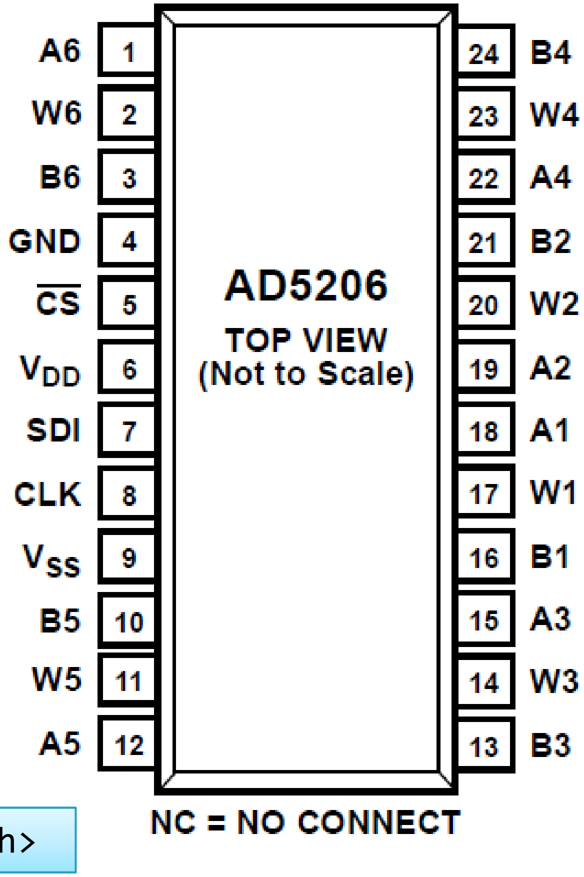
# Activity 13: Programmable Voltage Divider

- Use the AD5206 to build a programmable voltage divider
- Allow the user to set the resistance from the serial port
- Measure resistance with an Ohm meter, or using `analogRead()`

# AD5206: Summary of Pins and Commands



**SCK** (13) **MISO** (12) **MOSI** (11) **SS** (10)



**Remember**: `SPI.begin()` needed in `setup()` and `#include <SPI.h>`

```
digitalWrite(SS,LOW);      // hold SS pin low to select chip
SPI.transfer(potnumber);   // determine which pot (0..5)
SPI.transfer(wipervalue);  // transfer 8-bit wiper setting
digitalWrite(SS,HIGH);     // de-select the chip
```