

INTRODUCTION TO THE ARDUINO MICROCONTROLLER

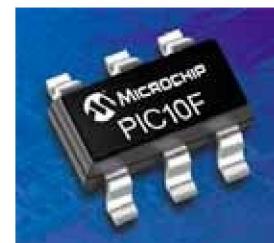
ELEMENTZ ENGINEERS GUILD PVT. LTD.

DEC, 2015



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

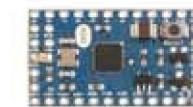
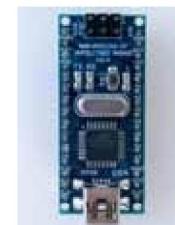
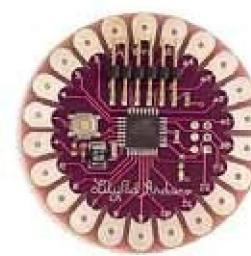
It's a ***movement***, not a microcontroller:

- 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



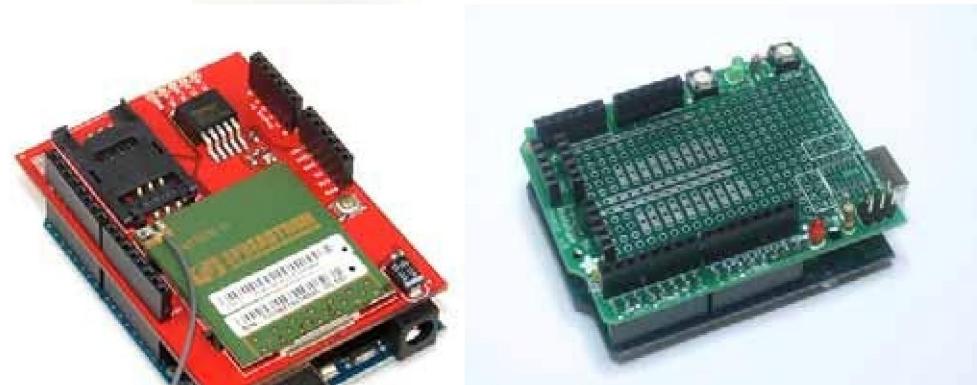
Arduino-like Systems

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



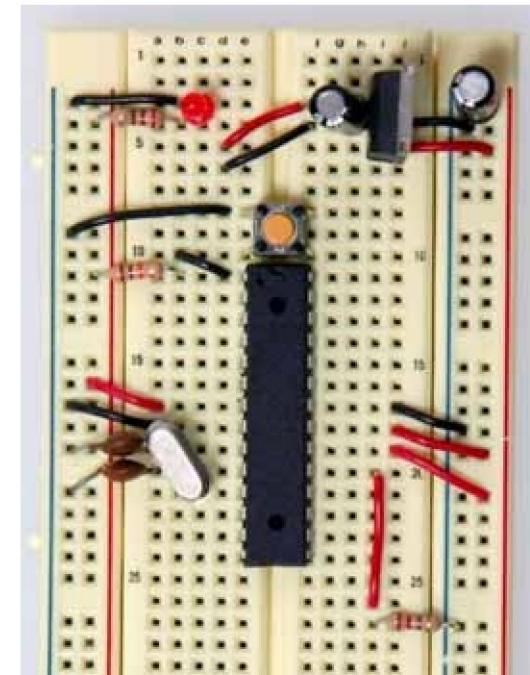
Arduino Add-ons (Shields)

- 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
 - Elementzonline
- ... or build your own
 - PC board
 - Solderless breadboard

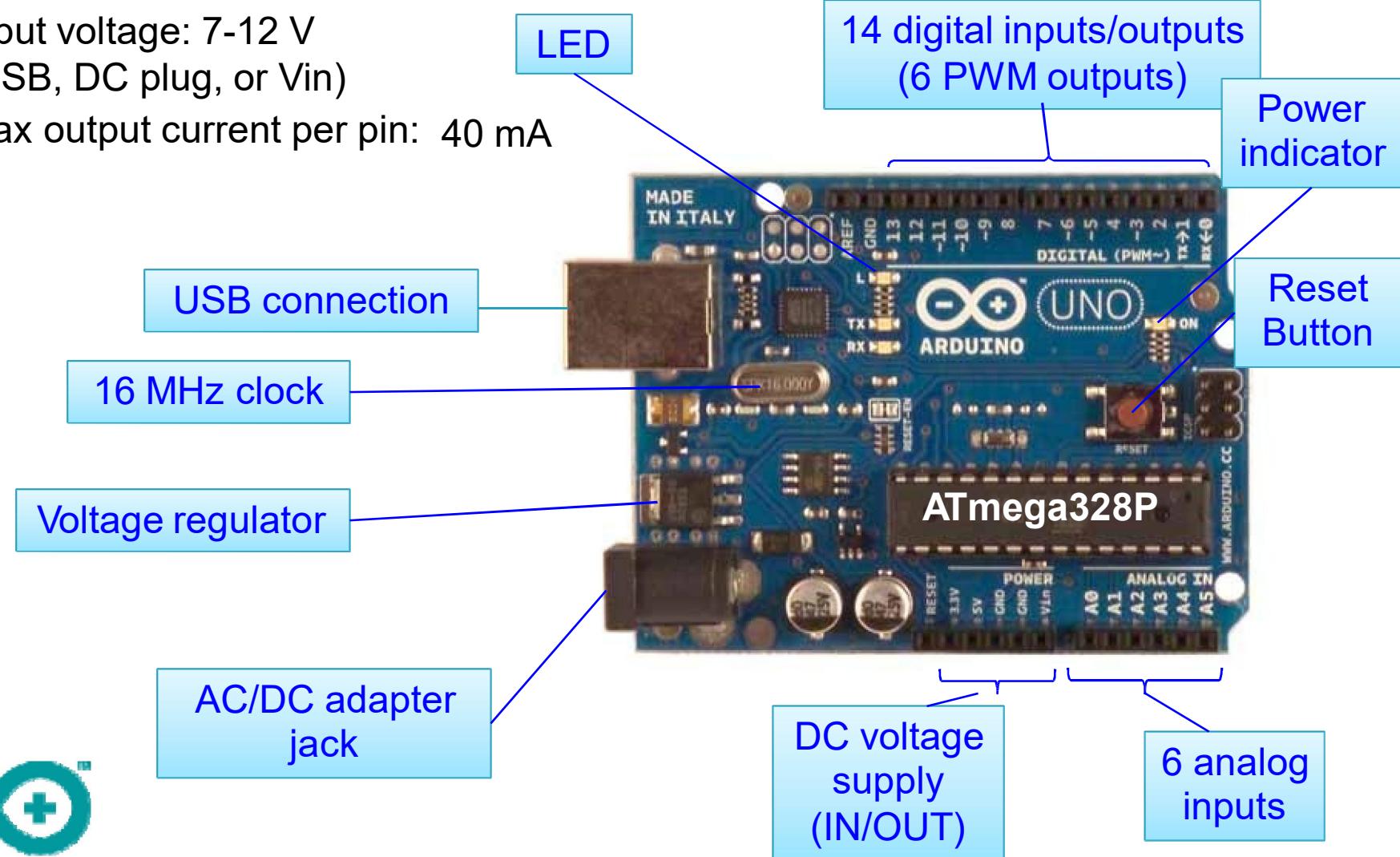


<http://itp.nyu.edu/physcomp/Tutorials/ArduinoBreadboard>



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

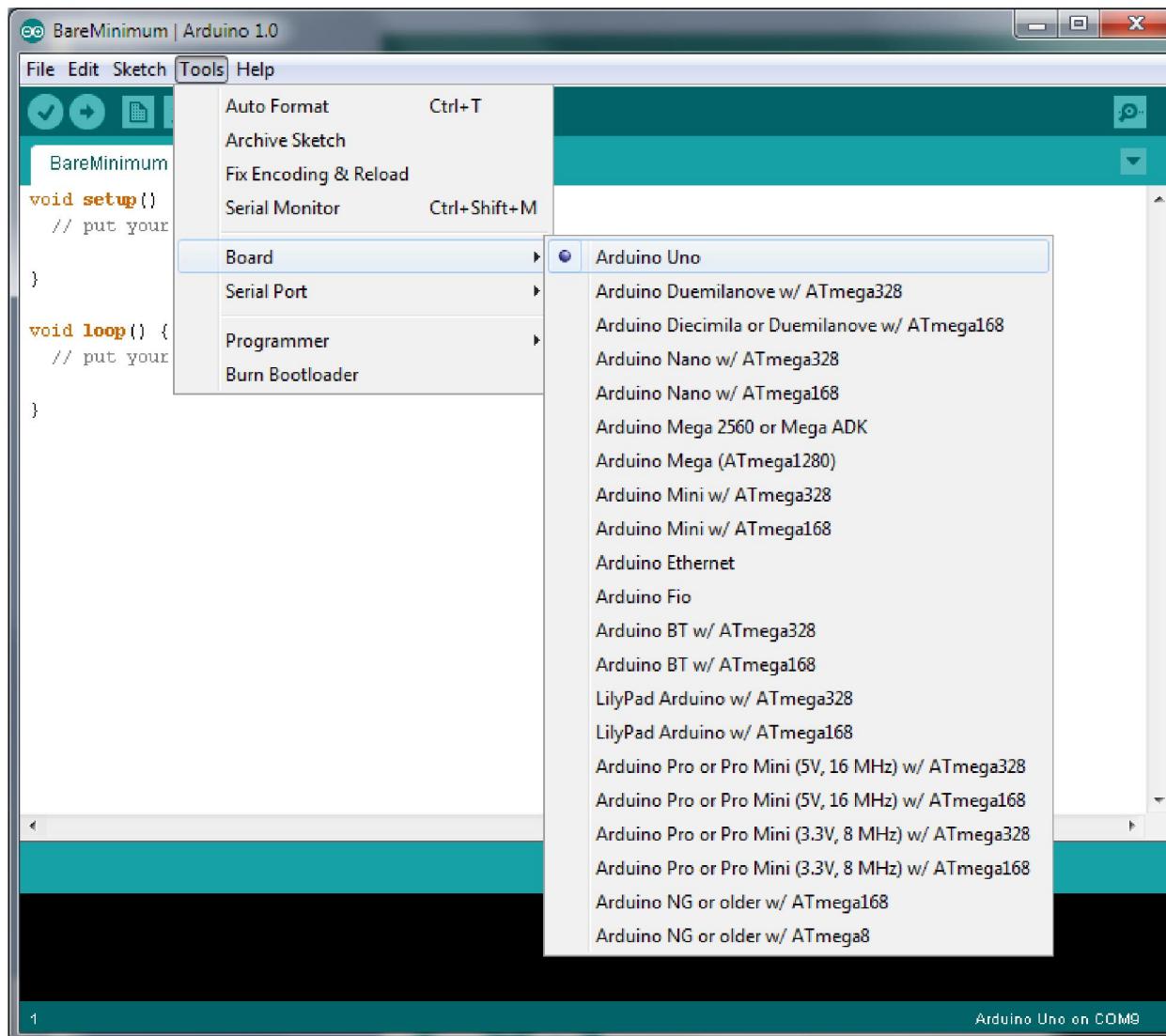


Download and Install

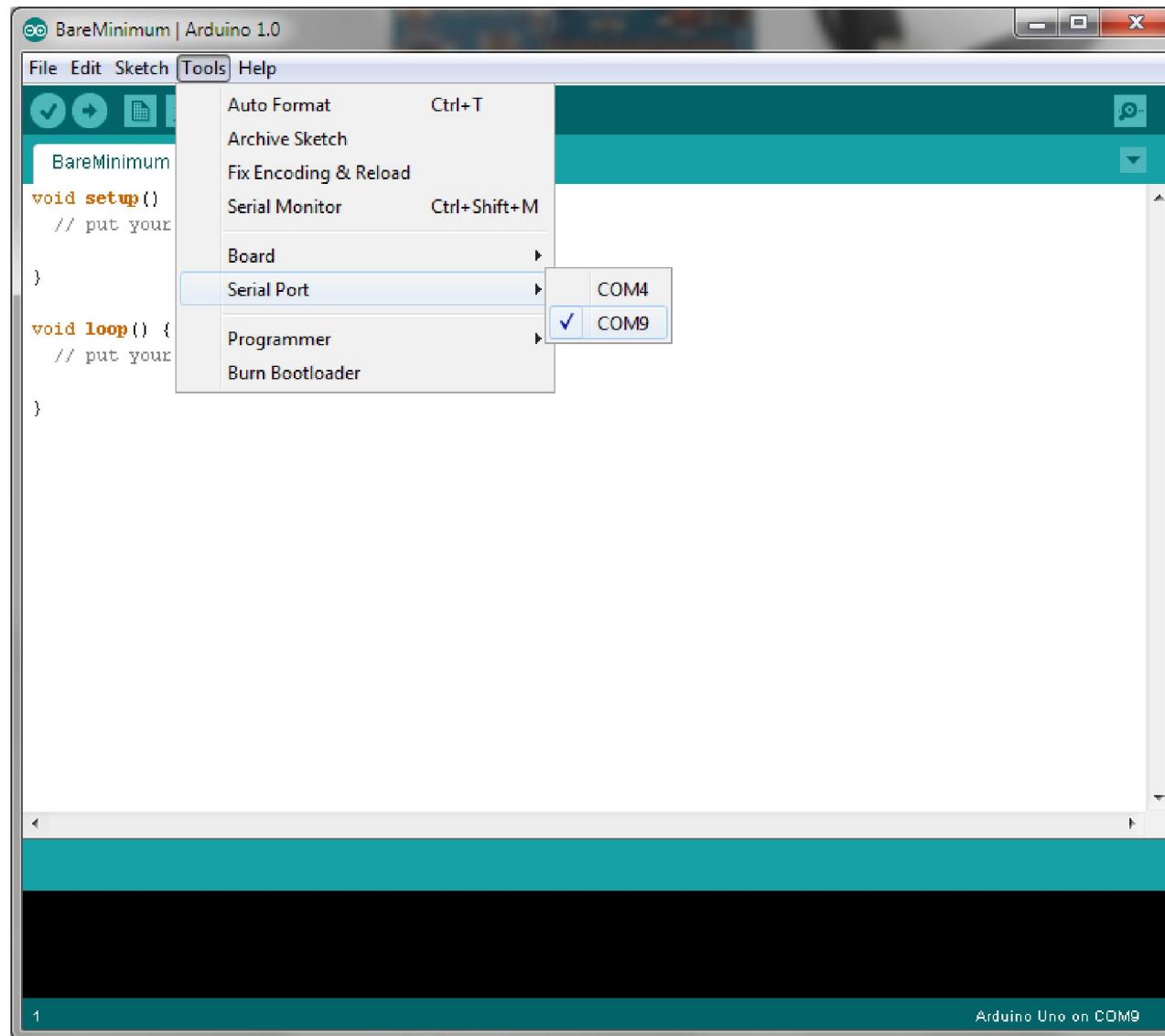
- Download Arduino compiler and development environment from:
<http://www.arduino.org/software>
- Current version: 1.7.8
- Available for:
 - Windows
 - MacOX
 - Linux
- No installer needed... just unzip to a convenient location
- ***Before running Arduino***, plug in your board using USB cable (external power is not necessary)
- When USB device is not recognized, navigate to and select the appropriate driver from the installation directory
- Run Arduino



Select your Board

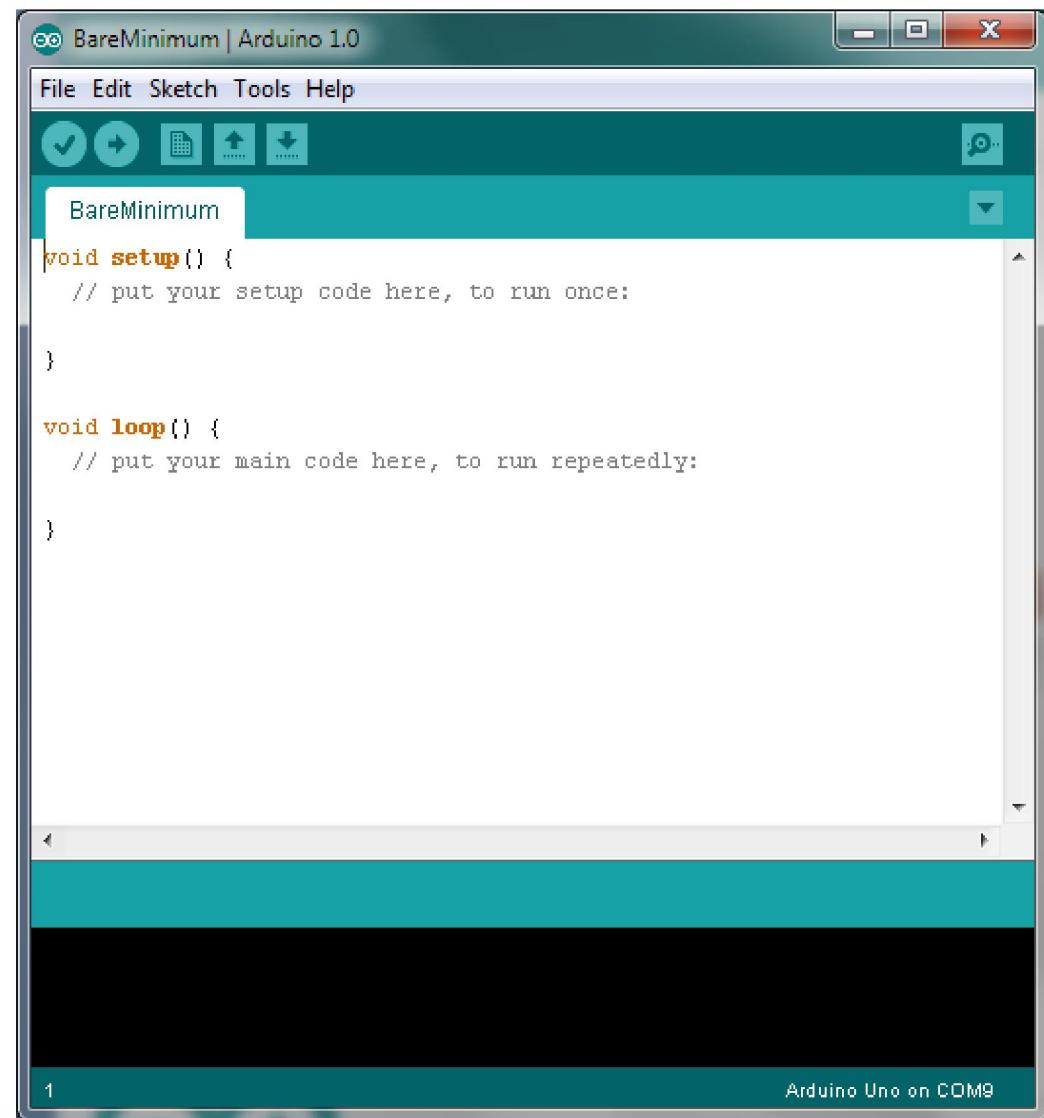


Select Serial Port



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



Using the Arduino IDE

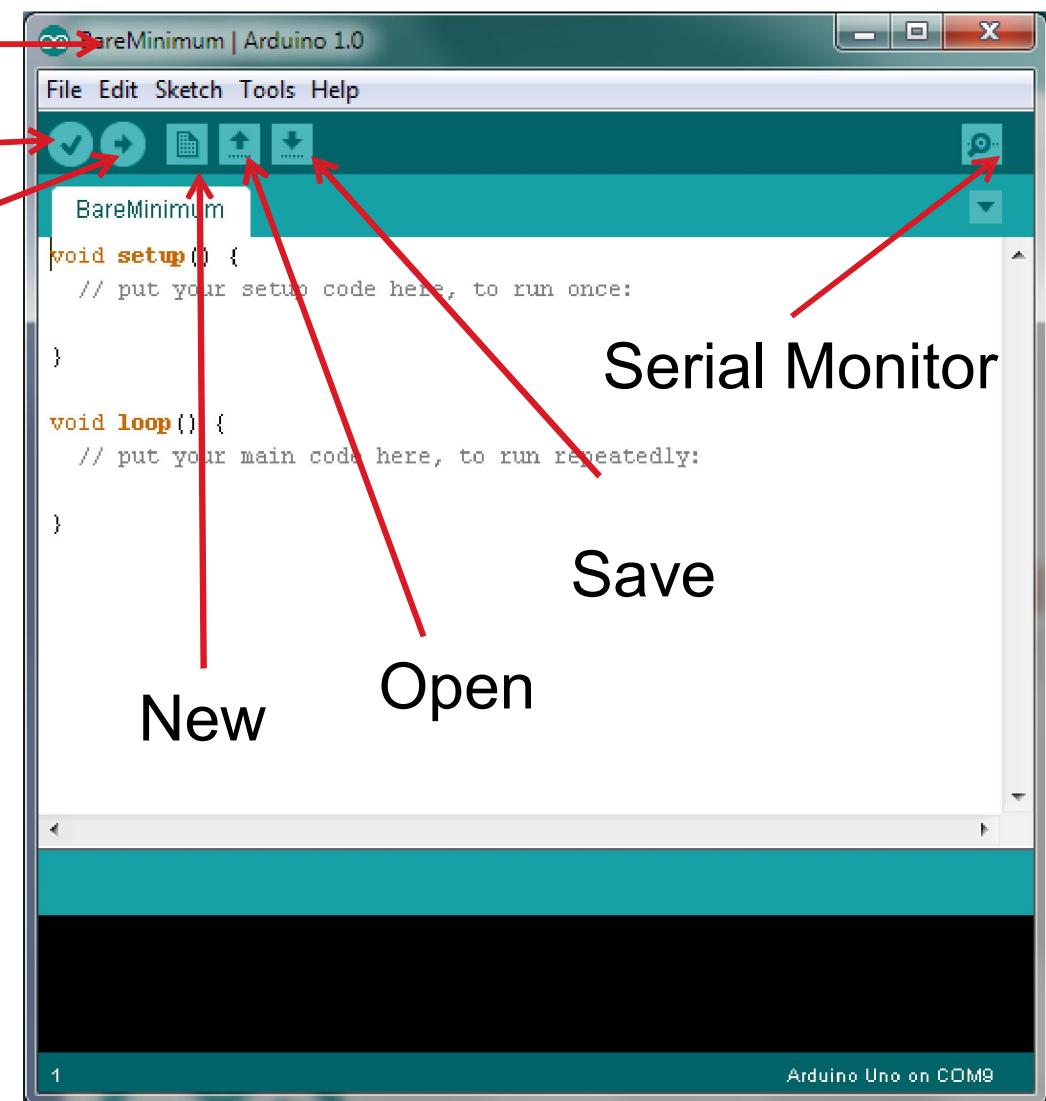
Name of sketch

Compile sketch

Upload to board

Program area

Messages / Errors



Arduino Reference

The image shows two windows side-by-side. On the left is the Arduino IDE interface, specifically version 1.0.1, with a sketch named "sketch_jun18a" open. A blue arrow points from the "Reference" option in the IDE's top menu bar to the "Reference" section of the Arduino Reference website on the right. The website is displayed in a Firefox browser window, featuring the Arduino logo and navigation links for Buy, Download, Getting Started, Learning, Reference, Hardware, and FAQ.

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

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 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

```
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
```

Set output high (+5V)

Wait 1000 milliseconds

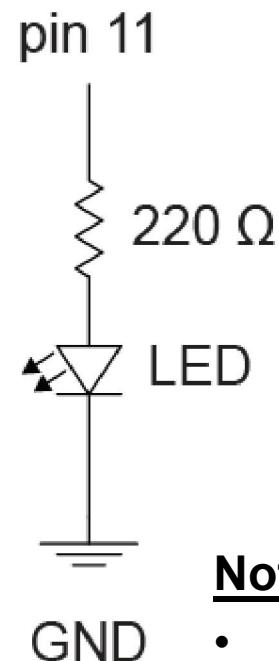
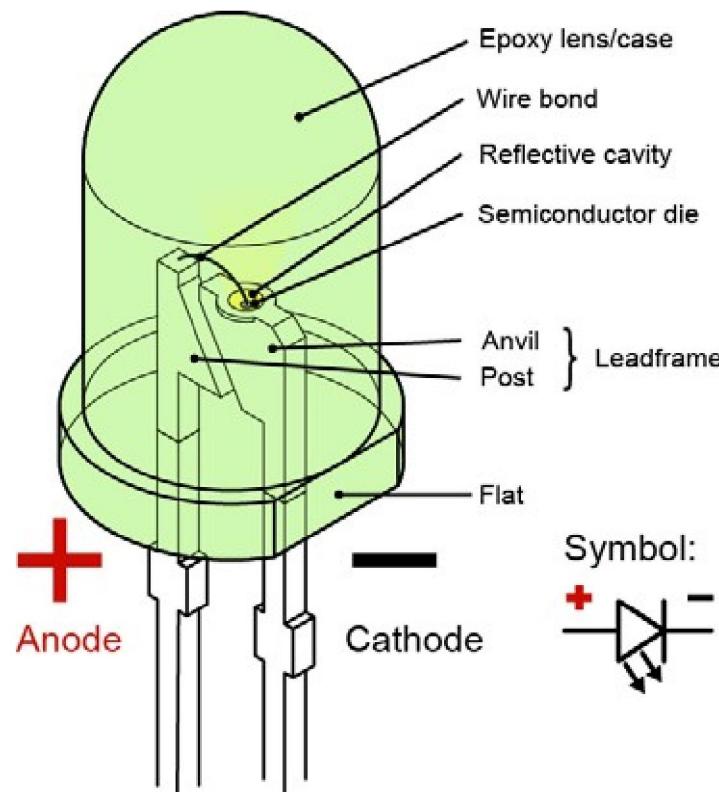
Set output low (0V)

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



Now connect your own LED

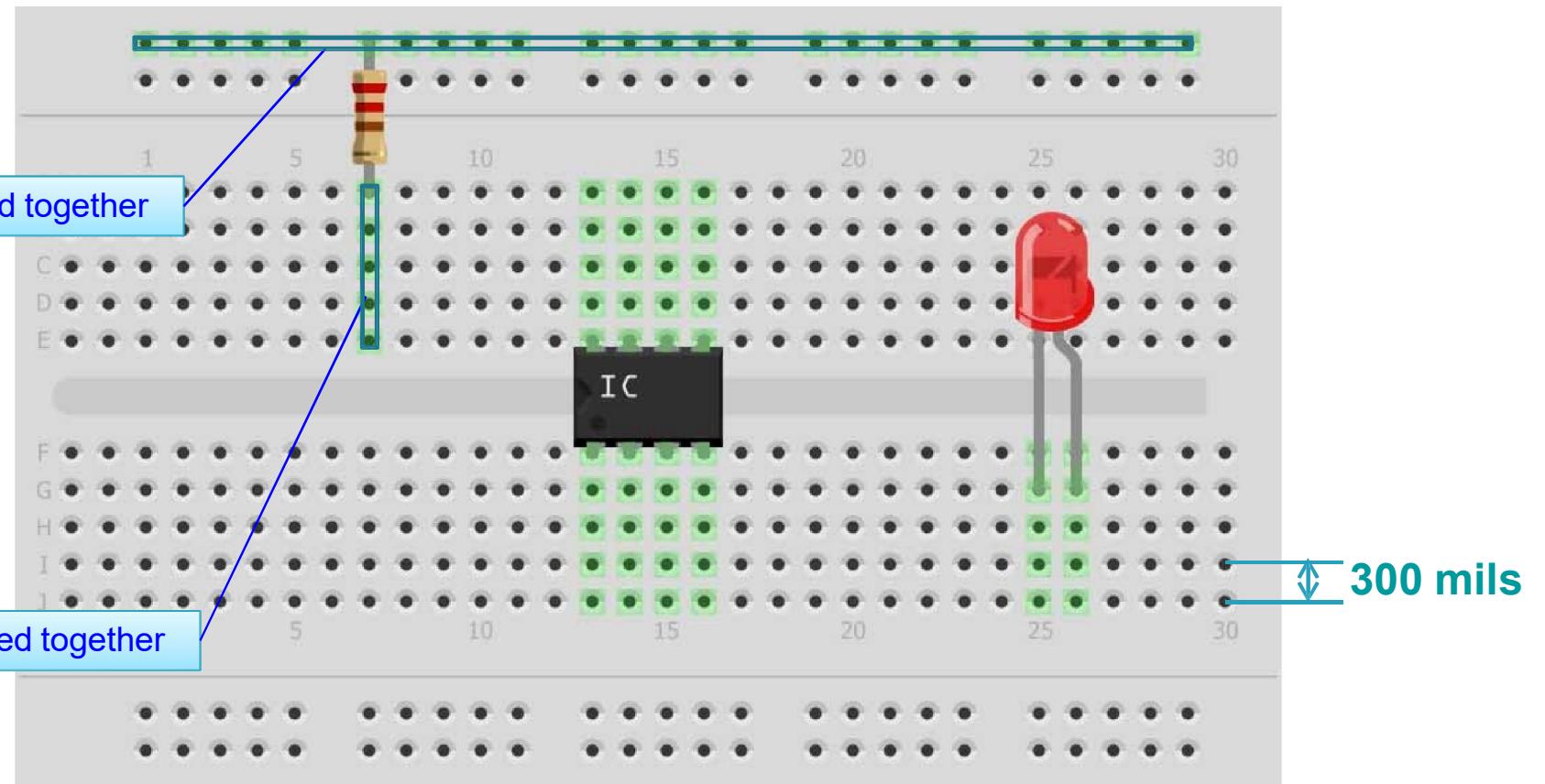
Anatomy of an LED:



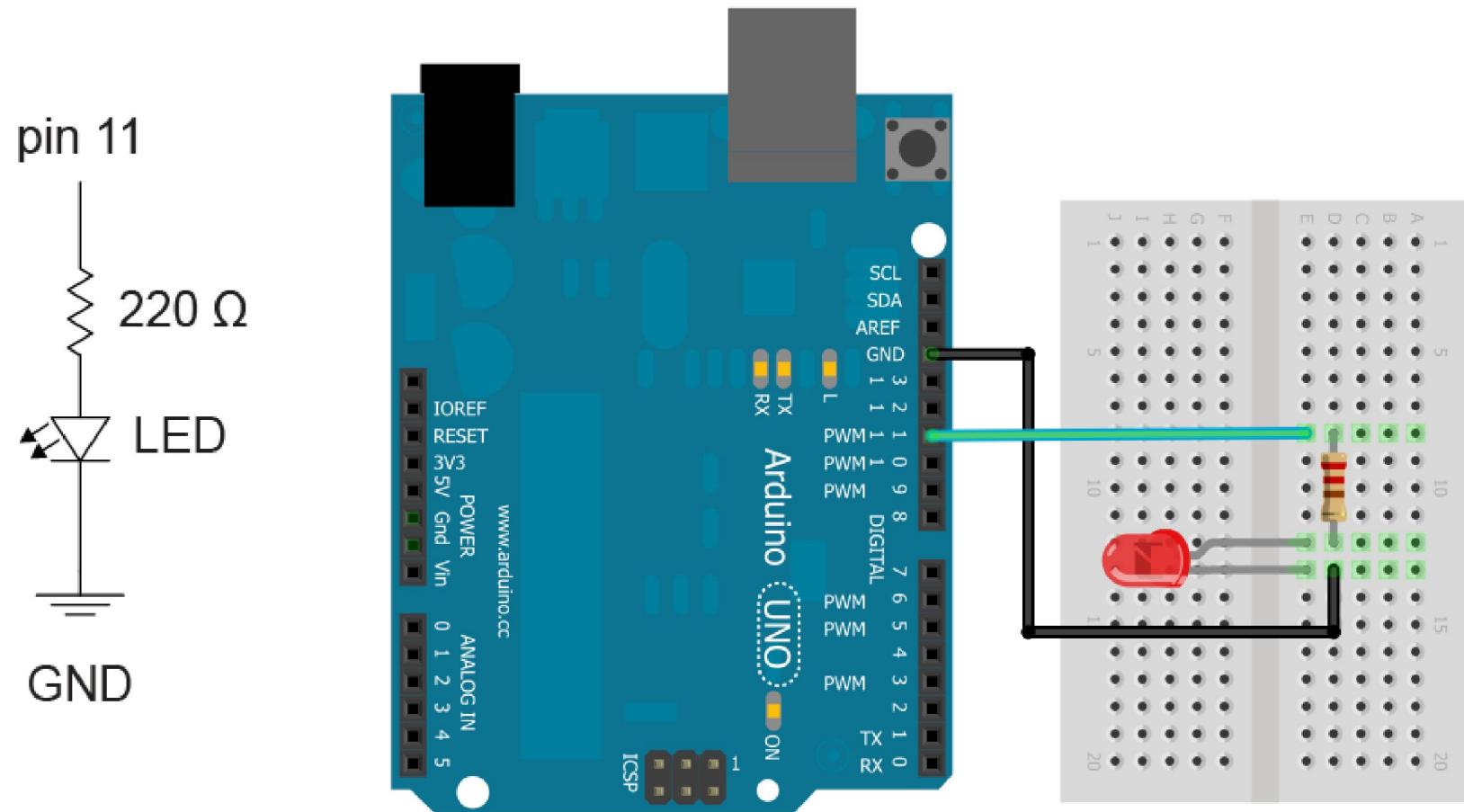
Notes:

- 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



Example: Using a Solderless Breadboard

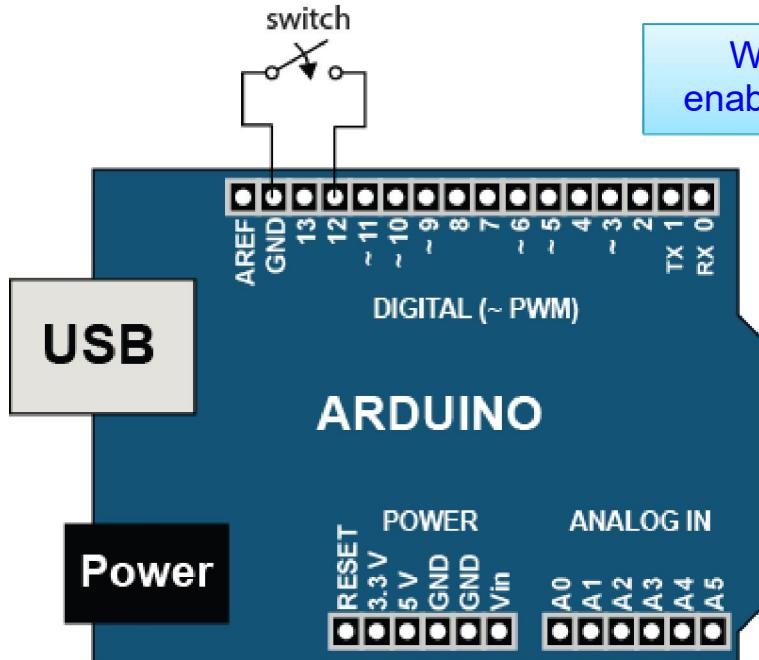


Experimenting

- 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  
}
```

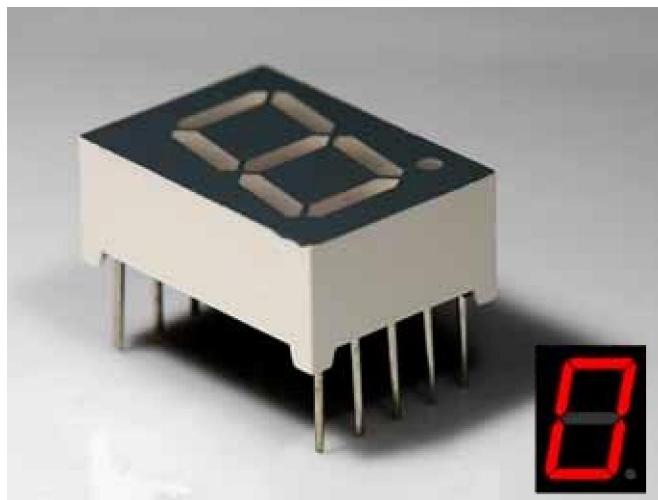
- Turn on/off LED based on switch
- Pin 12 reads **LOW** when switch is closed
- Pin 12 reads **HIGH** when switch is open (pull-up)



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

Activity 2: Seven-Segment Display

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

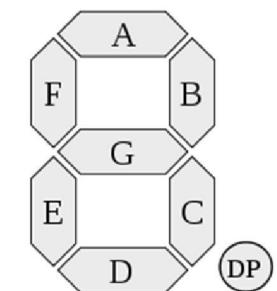


- Consider writing a function:
`void writeDigit(int n)`
that writes a single digit



Seven-Segment Display Table

Digit	ABCDEFG	A	B	C	D	E	F	G
0	0x7E	on	on	on	on	on	on	off
1	0x30	off	on	on	off	off	off	off
2	0x6D	on	on	off	on	on	off	on
3	0x79	on	on	on	on	off	off	on
4	0x33	off	on	on	off	off	on	on
5	0x5B	on	off	on	on	off	on	on
6	0x5F	on	off	on	on	on	on	on
7	0x70	on	on	on	off	off	off	off
8	0x7F	on						
9	0x7B	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**

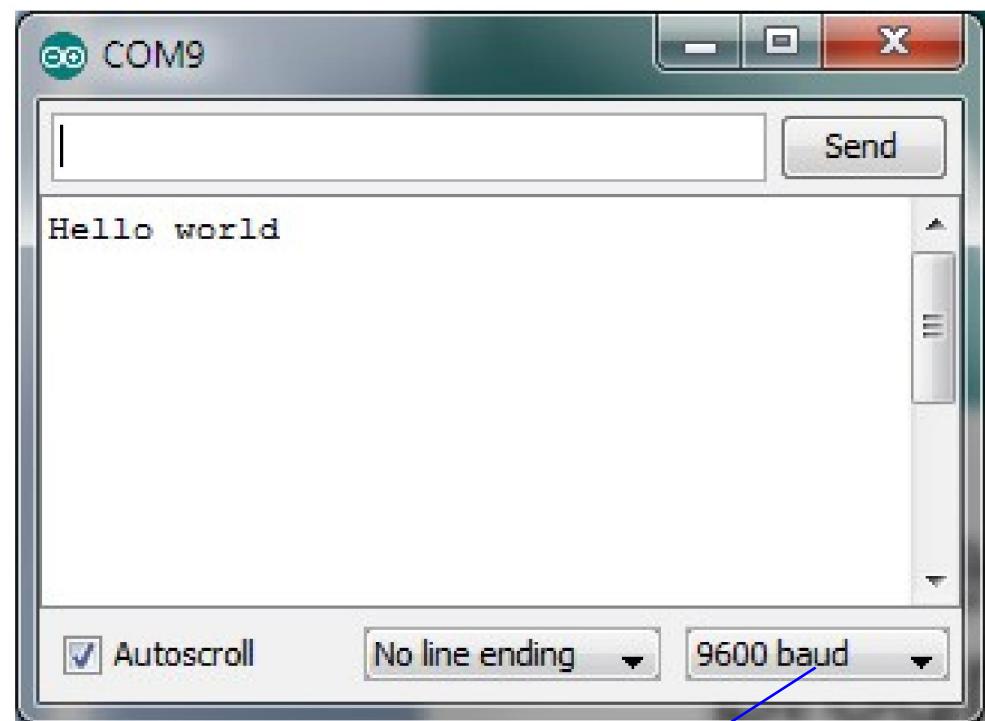
- **Serial.println(val)**
Same as **Serial.print()**, but with line-feed



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:



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

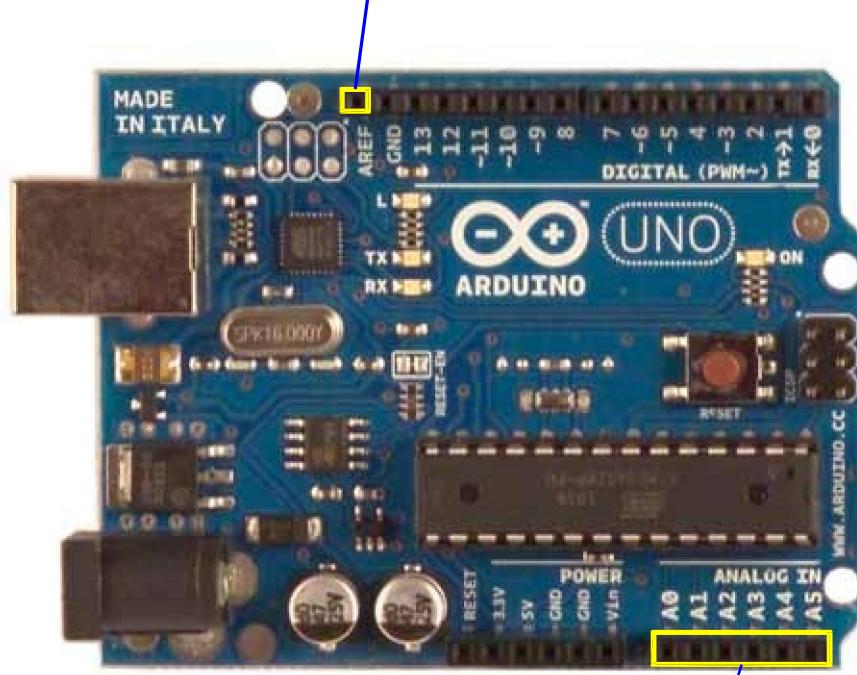
Examples:

- `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);`

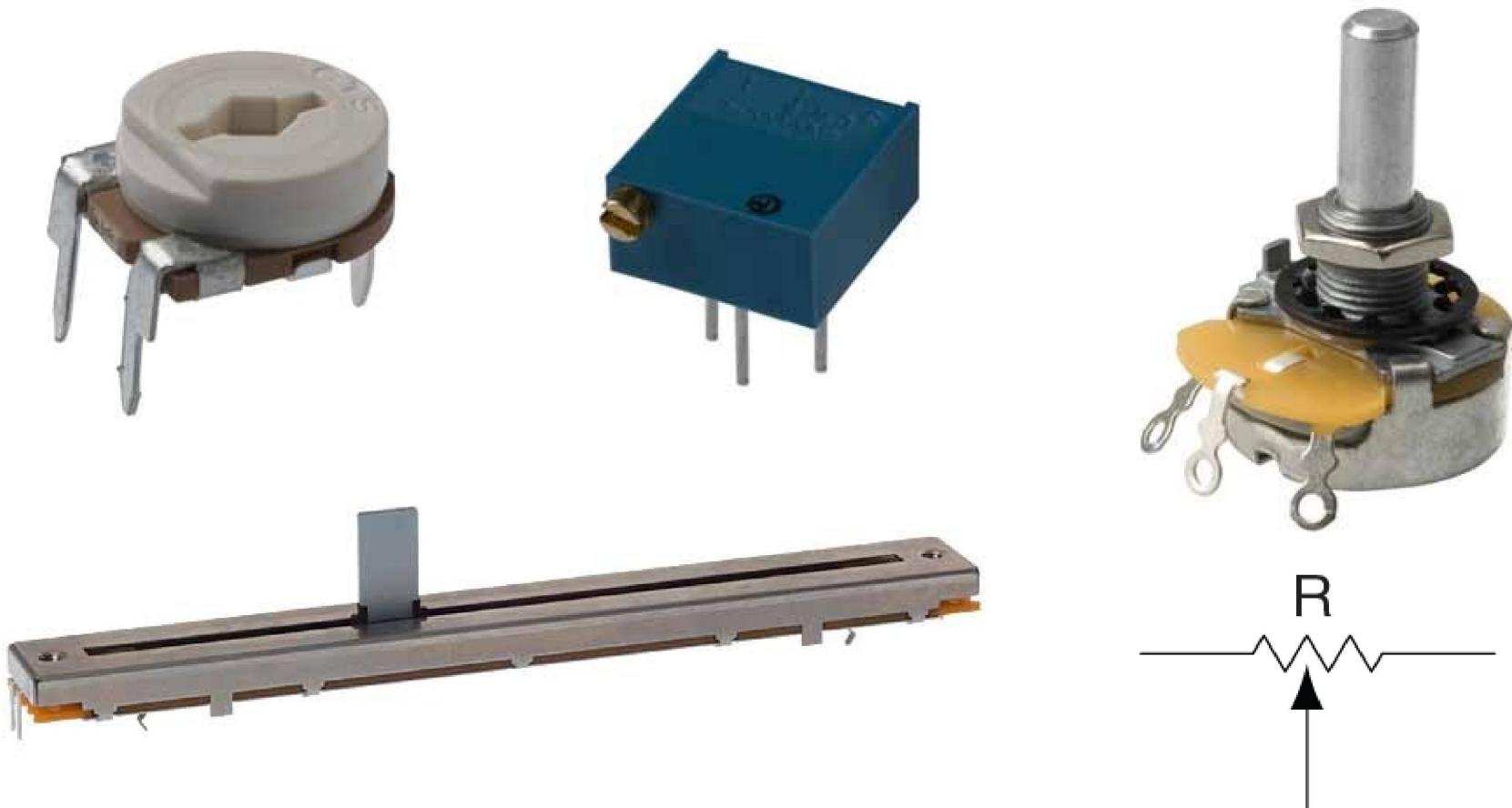
Reading Analog Values

- `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



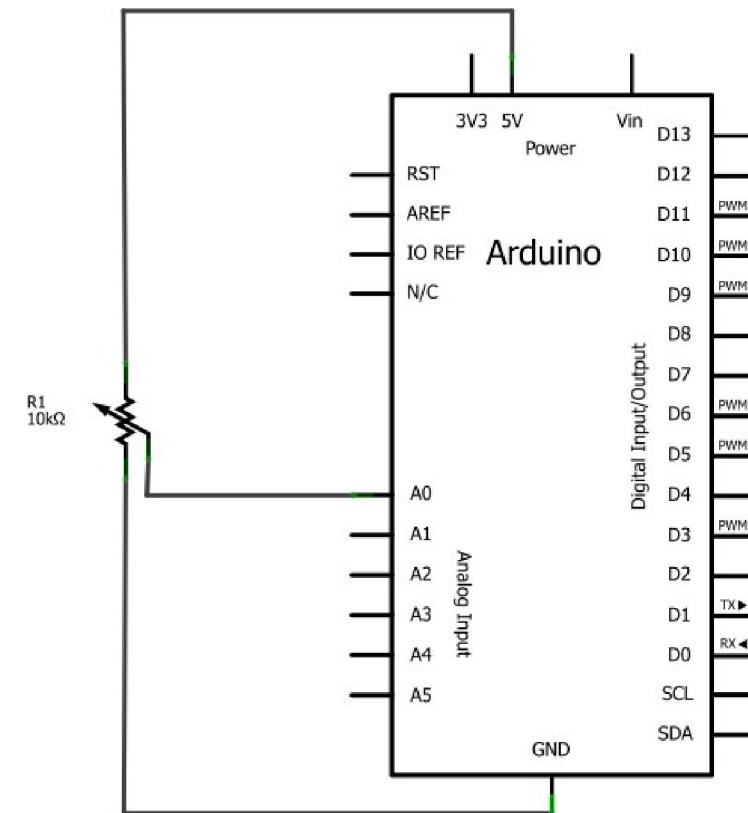
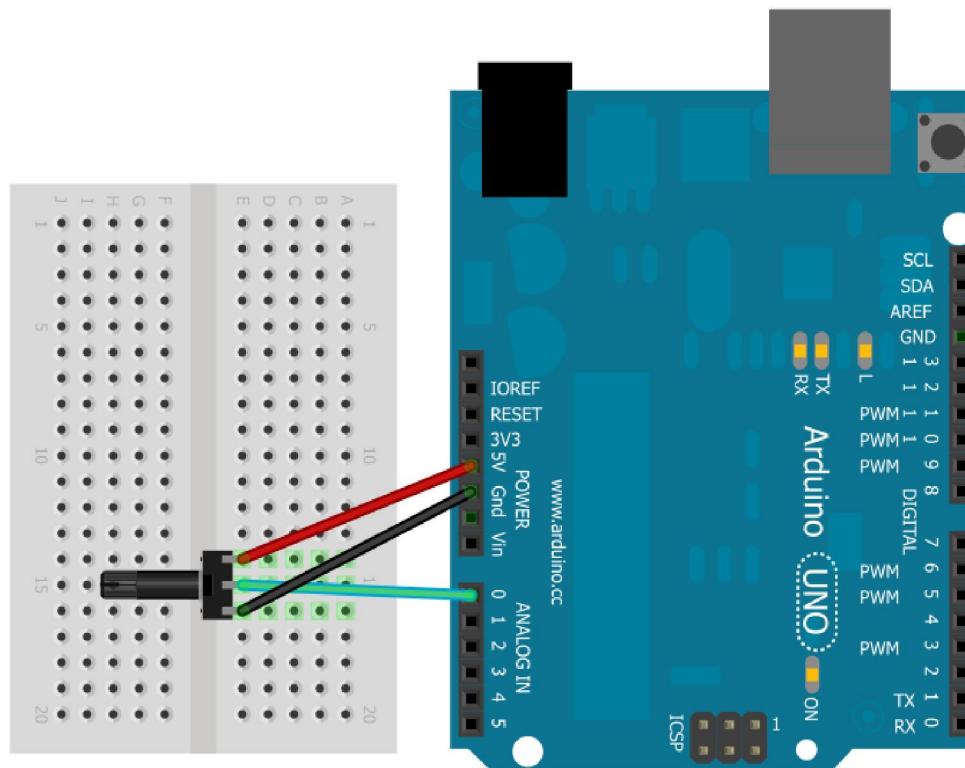
Note: Do NOT use `pinMode(A0, INPUT)` unless you want to use A0 for **DIGITAL** input.

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

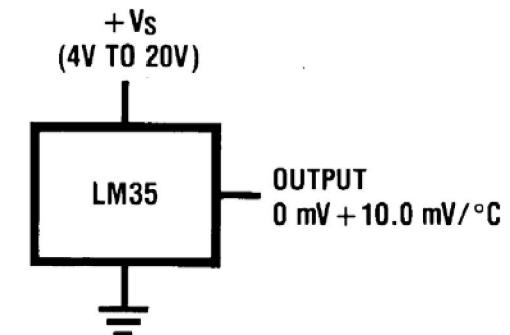
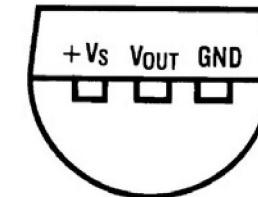
LM35

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 $\pm\frac{1}{4}^{\circ}\text{C}$ 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

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

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

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

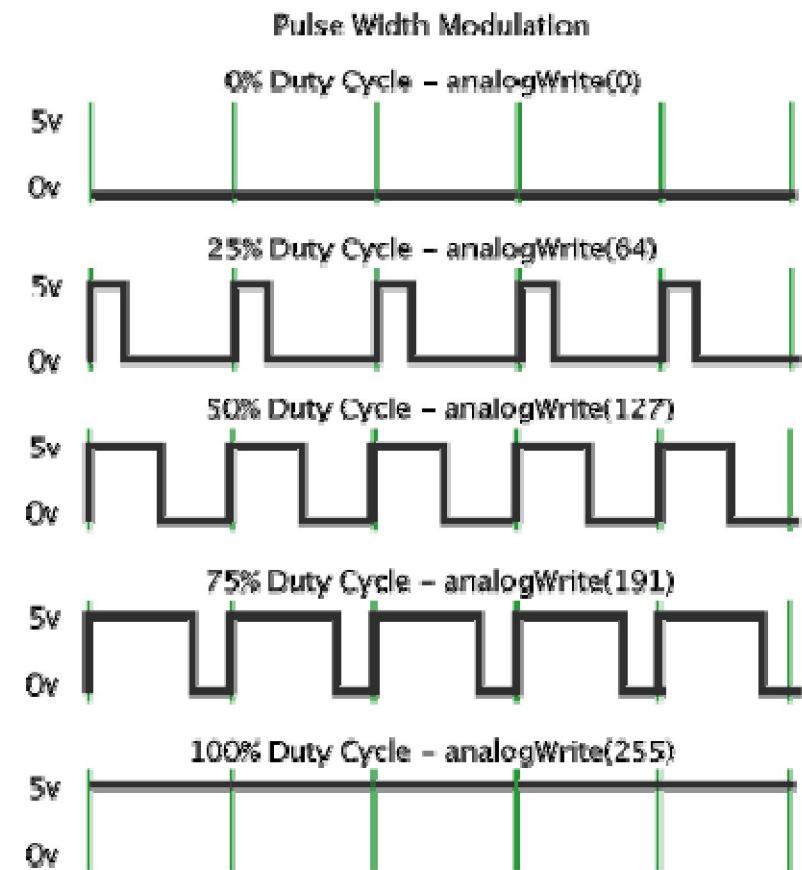


- Build a circuit and write a sketch that turns
on an LED whenever it gets dark
Hint: connect the photoresistor in a voltage
divider



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

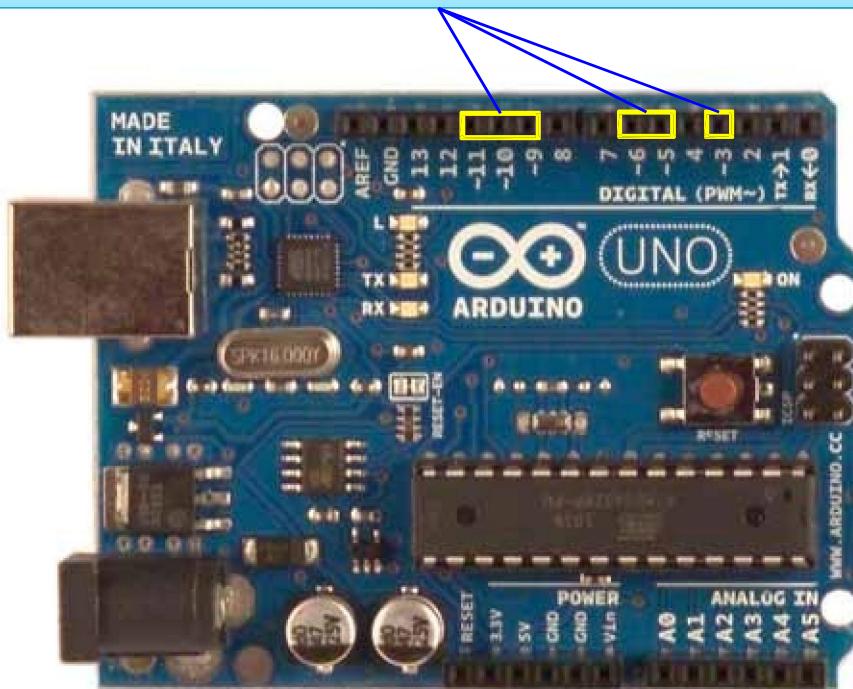


<http://arduino.cc/en/Tutorial/PWM>



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:
 - $16 \text{ MHz} / 2^{15} = 488.28125 \text{ Hz}$

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:

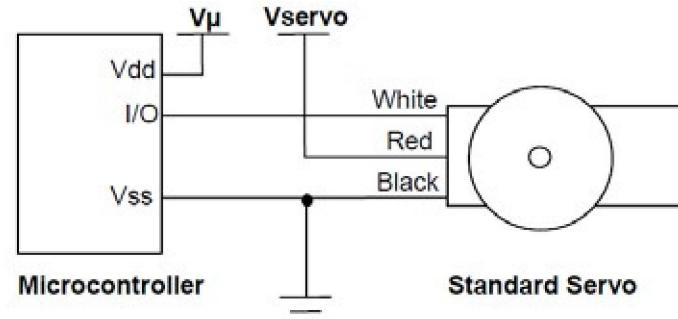
- `newValue = map(oldValue, 0, 1023, 0, 255);`

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



V_μ = microcontroller voltage supply

V_{servo} = 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 **V_{servo}** + 0.2 V

Pin	Name	Description	Minimum	Typical	Maximum	Units
1 (White)	Signal	Input; TTL or CMOS	3.3	5.0	$V_{servo} + 0.2$	V
2 (Red)	V_{servo}	Power Supply	4.0	5.0	6.0	V
3 (Black)	V_{ss}	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

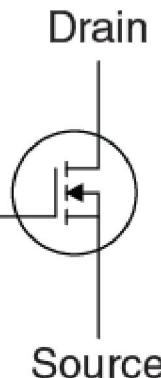
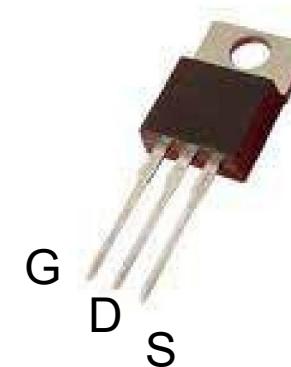
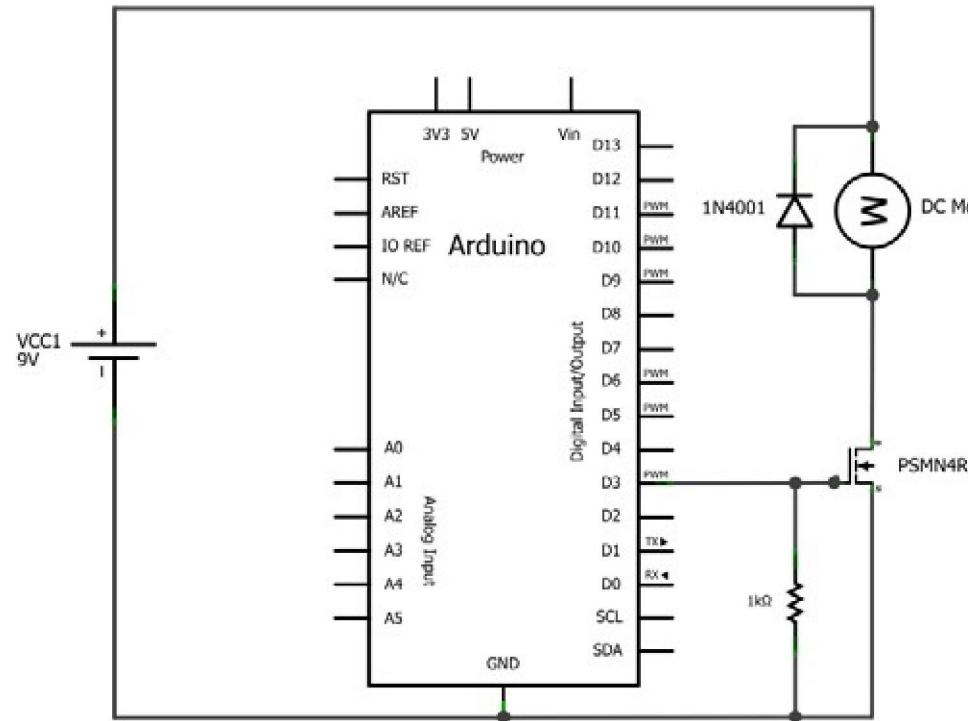


Activity 9 – Servomotor Control

- 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!

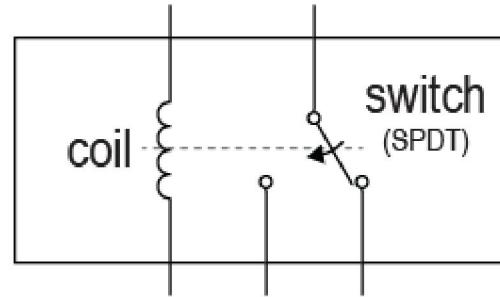


Activity 10 – PWM Speed Control

- 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



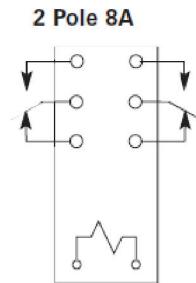
- Electromechanically -actuated switch
- Provides electrical isolation
- Typically few ms response time



Note: Arduino cannot supply enough current to drive relay coil

RT series (DC Coil)
16 Amp PC Board Miniature Relay

File E22575
 File LR15734
 NR 6106



Coil Data @ 25°C

Voltage: 5 to 110VDC.
Nominal Power @ 25°C: 400mW.
Duty Cycle: Continuous.

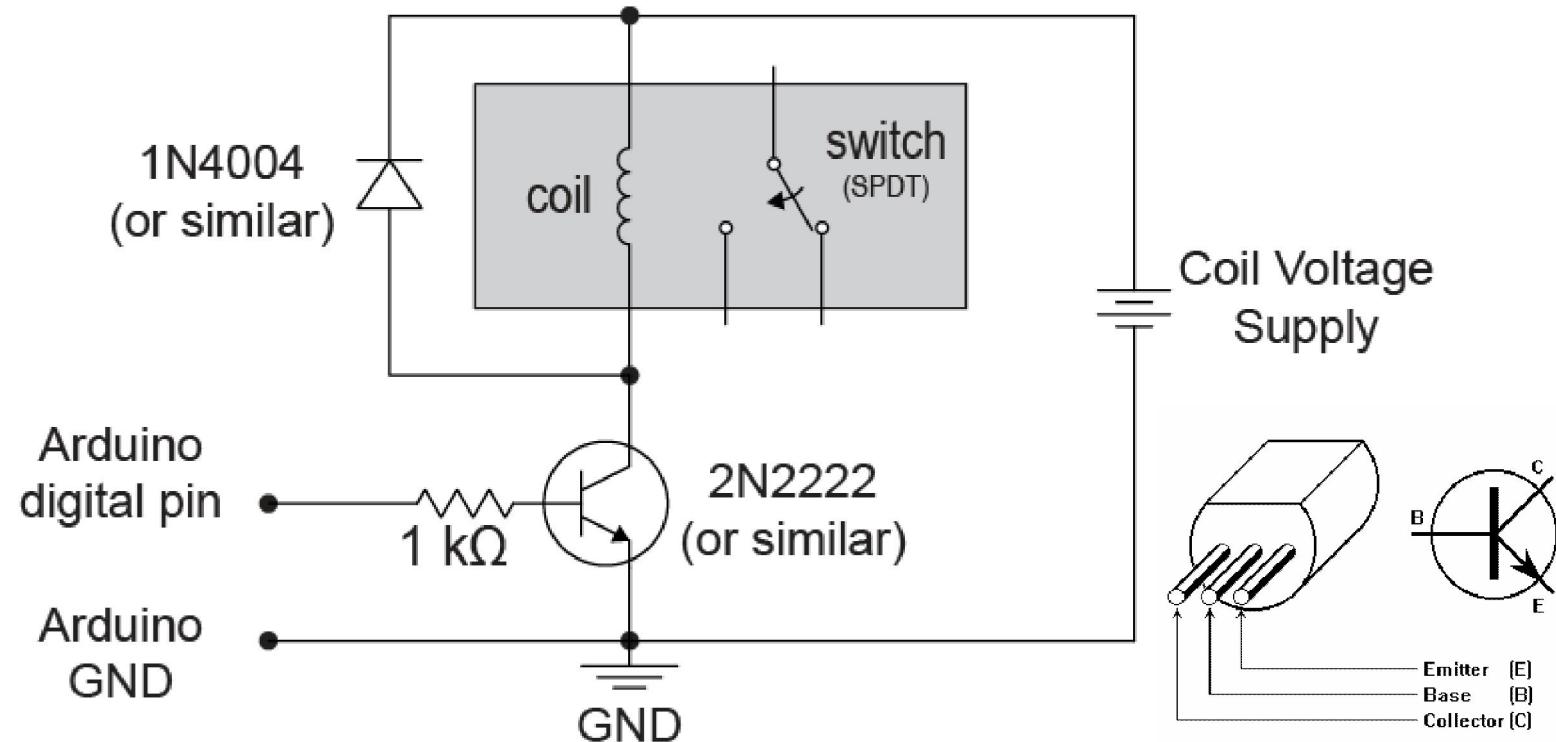
Initial Insulation Resistance: 10,000 megohms, min., at 25°C, 500VDC and 50% rel. humidity.

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

Relay Driver Circuit



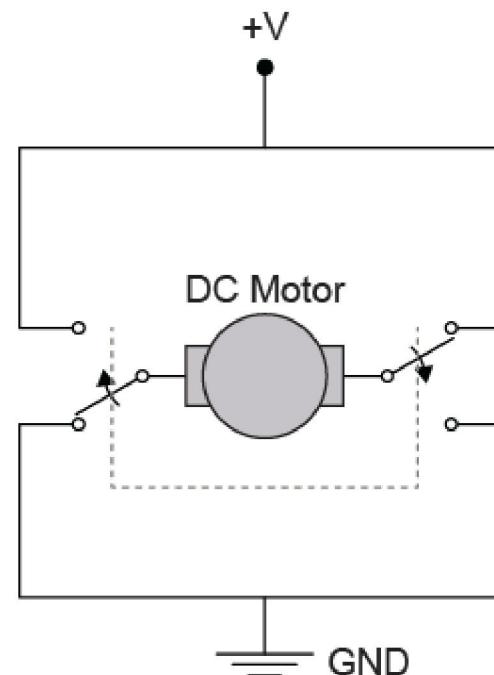
- NPN transistor: acts like a current-controlled switch
- MOSFET will also work
- Diode prevents back-EMF (associated with inductive loads)
- Coil voltage supply and Arduino share common GND



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)



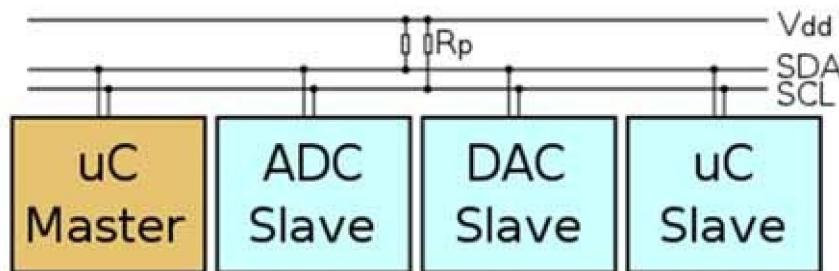
Communication: I²C, SPI

- I²C (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)



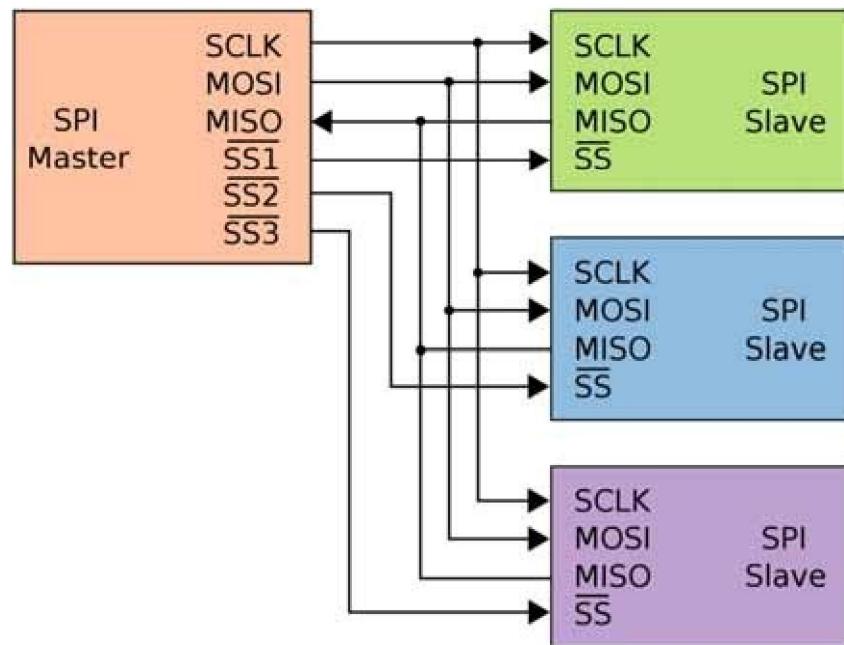
Connecting Multiple Devices (I²C and SPI)

Master (μ C) with three I²C slaves:

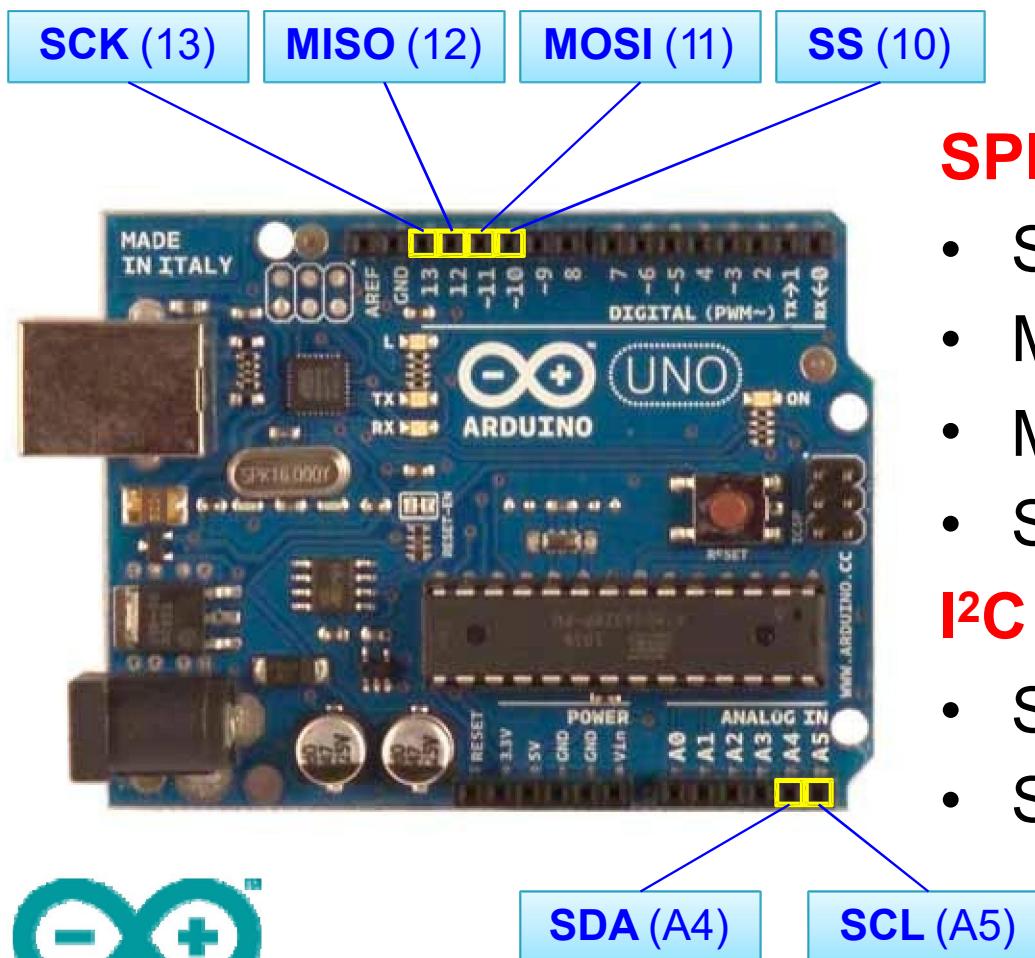


<http://en.wikipedia.org/>

Master with three SPI slaves:



SPI and I²C on the Arduino



SPI pins:

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

I²C pins:

- SDA = data line
- SCL = clock line



Basic Arduino I²C Commands

COMMAND	EXPLANATION
<code>Wire.begin()</code>	Join the I ² C bus as master (usually invoked in <code>setup</code>)
<code>Wire.beginTransmission(address)</code>	Begin communicating to a slave device
<code>Wire.write(byte)</code>	Write one byte to I ² C bus (after request)
<code>Wire.endTransmission(address)</code>	End transmission to slave device

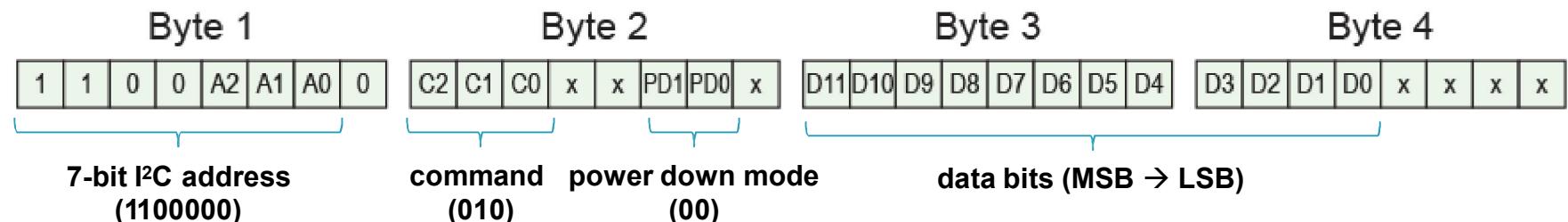


Note: you must include the Wire library:
`#include <Wire.h>`

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

Example: MCP4725 12-bit DAC

MCP4725 write command (taken from data sheet)



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 I²C Commands

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



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

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

Basic Arduino SPI Commands

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



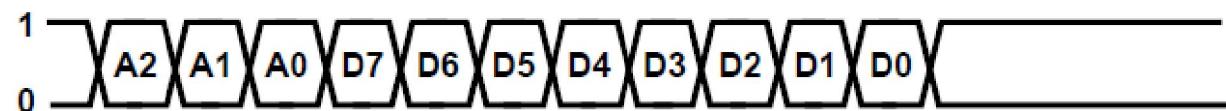
Note: you must include the SPI library:
`#include <SPI.h>`

Note: `pinMode()` not needed

SPI Write Sequence

Note: same as
MOSI (master
out slave in)

SDI

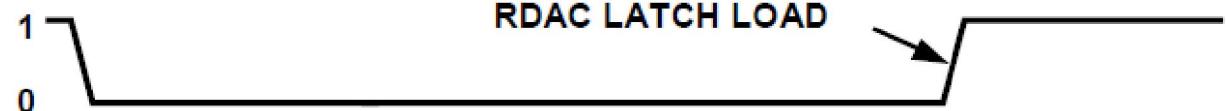


Note: same as SS
(slave select)

CLK



cs



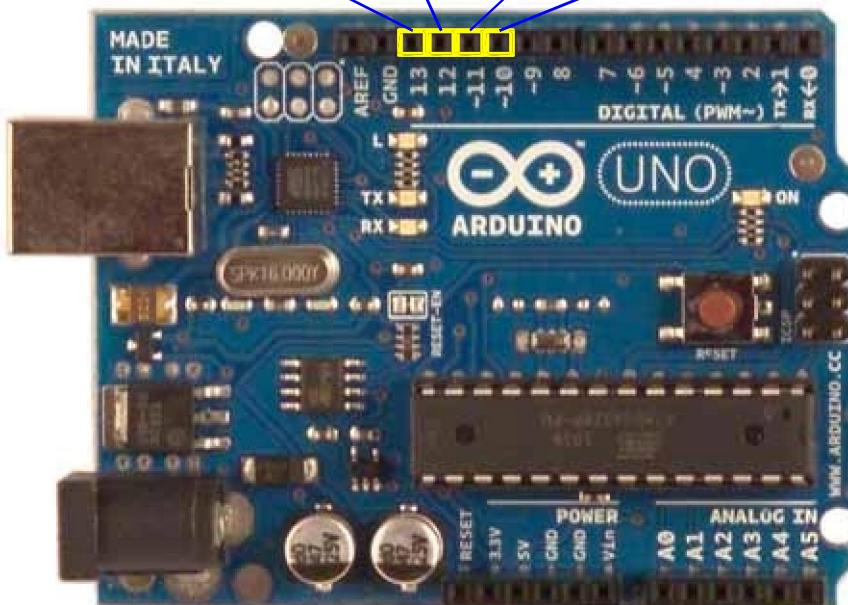
Arduino program segment:

```
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
```



Summary of 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
```