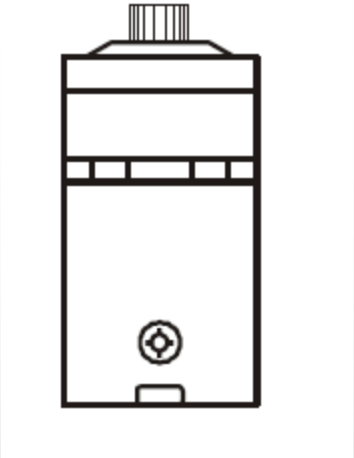
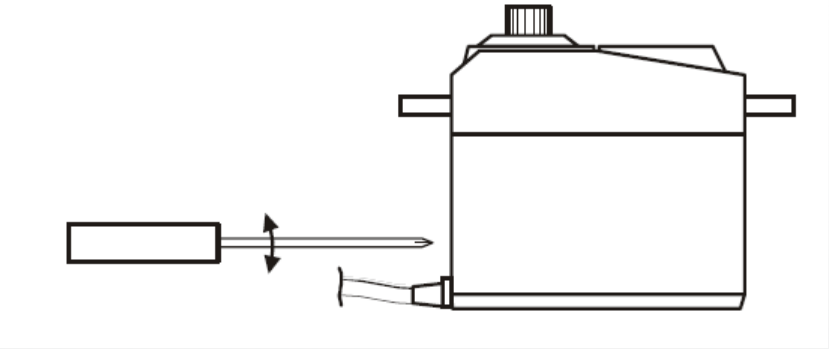
Arduino- Digital Output - Continuous Rotation- Adjustment

As opposed to standard Servo that its rotation is limited to 180 degrees both ways, a continuous rotation servo can keep rotating unlimitedly-again both ways- based on the frequency that is pulsed out to it. There is a specific frequency at which the Servo motor should be static and beyond and before which the servo will change in its rotation direction.
There is a pin on the servo motor that enables us to adjust the servo for its static frequency.





Arduino-Digital Output - Continuous Rotation-Adjustment

Upload the following code to the board and while the servo is connected, try to adjust the pin until the servo motor is static.

Once the servo is adjusted to this code any pulse grater than 1500 will result in rotation in one direction while any pulse less than 1500 will result in rotation in the other direction

```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500); // 1.5ms This is the frequency at which the servo motor should be static
digitalWrite(5,LOW);
delay(20); // 20ms
}
}
```

```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500); // 1.5ms This is the frequency at which the servo motor should be static
digitalWrite(5,LOW);
delay(20); // 20ms
}
}
```

Arduino- Digital Output - Continuous Rotation- Direction Change

```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{

//Rotating in One direction
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(20); // 20ms
}

//Stop
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500);
digitalWrite(5,LOW);
delay(20); // 20ms
}

//Rotating in the other direction
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1200);
digitalWrite(5,LOW);
delay(20); // 20ms
}

//Stop
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500);
digitalWrite(5,LOW);
delay(20); // 20ms
}
}
```

Once the servo is adjusted to this code any pulse grater than 1500 will result in rotation in one direction while any pulse less than 1500 will result in rotation in the other direction

Arduino- Digital Output - Continuous Rotation- Direction Change

```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{
//Rotating in One direction
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(20); // 20ms
}
//Stop
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500);
digitalWrite(5,LOW);
delay(20); // 20ms
}
//Rotating in the other direction
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1200);
digitalWrite(5,LOW);
delay(20); // 20ms
}
//Stop
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500);
digitalWrite(5,LOW);
delay(20); // 20ms
}
}
```

Once the servo is adjusted to this code any pulse grater than 1500 will result in rotation in one direction while any pulse less than 1500 will result in rotation in the other direction

Arduino- Digital Output - Continuous Rotation- Delayed Steps

```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{
//Continious Rotation
for (int i = 0; i <= 20; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(1);
}
//Rotating with delayed steps
for (int i = 0; i <= 20; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(100);
}
//More Delay
for (int i = 0; i <= 20; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(200);
}
//More Delay
for (int i = 0; i <= 20; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(400);
}
//More Delay
for (int i = 0; i <= 20; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(800);
}
//More Delay
for (int i = 0; i <= 20; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(1800);
}
}
```

Playing with delay() gives us pauses between rotation steps

Arduino- Digital Output - Continuous Rotation- Controlling Rotation Angle

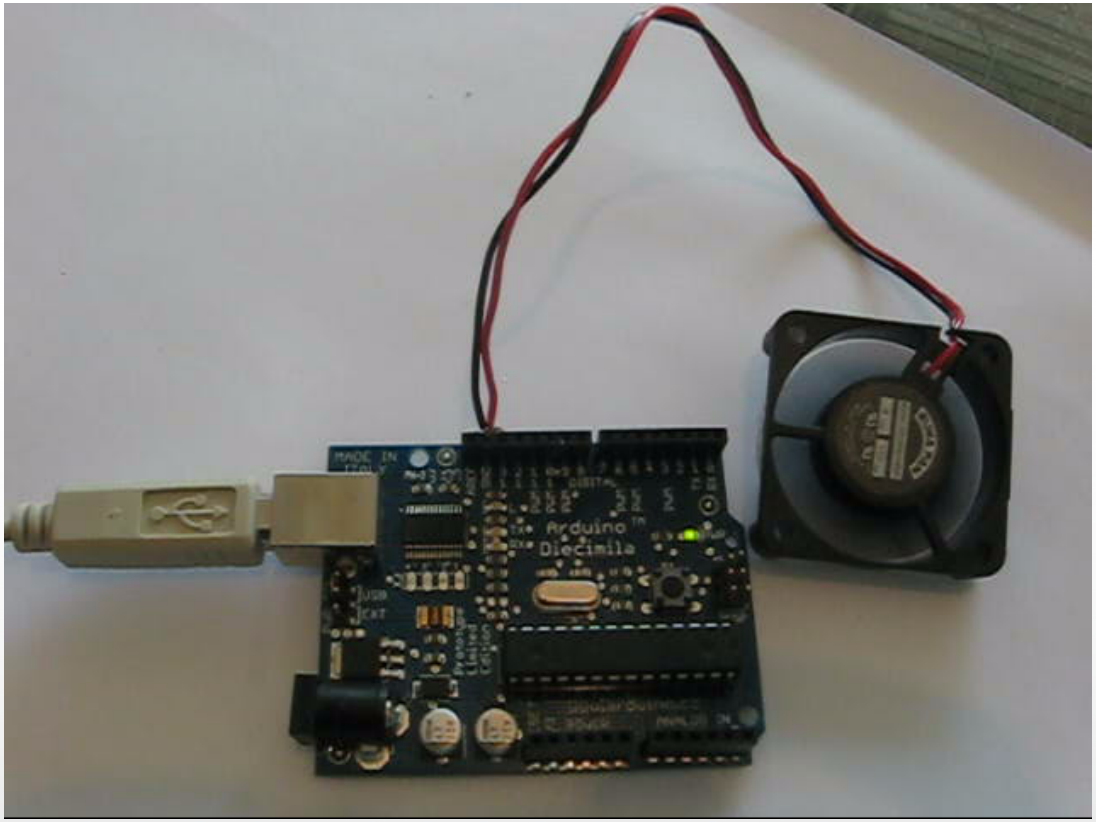
```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{
for (int i = 0; i <= 10; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(20);
}
delay(1000);
for (int i = 0; i <= 20; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(20);
}
delay(1000);
for (int i = 0; i <= 30; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(20);
}
delay(1000);
for (int i = 0; i <= 40; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(20);
}
delay(1000);
}
```

Playing with the number of steps in the for loop gives us variations in the span /Angle of the rotation

Arduino-Digital Output – Wind –Controlling a Fan

```
// Connect the fan to Pin 13 and Ground
void setup(){
pinMode(13, OUTPUT); // Specify Arduino Pin number
and output/input mode
}
void loop(){
digitalWrite(13, HIGH); // Turn on Pin 13 sending a
HIGH Signal
delay(1000);          // Wait for one second
digitalWrite(13, LOW);  // Turn off Pin 13 sending a
LOW Signal
delay(3000);          // Wait for Three second
}
```

Controlling a Fan is as easy as sending a HIGH or LOW Signal to the Pin that the fan is connected to.



Arduino-Digital Output – Rotation –Controlling a DC Motor

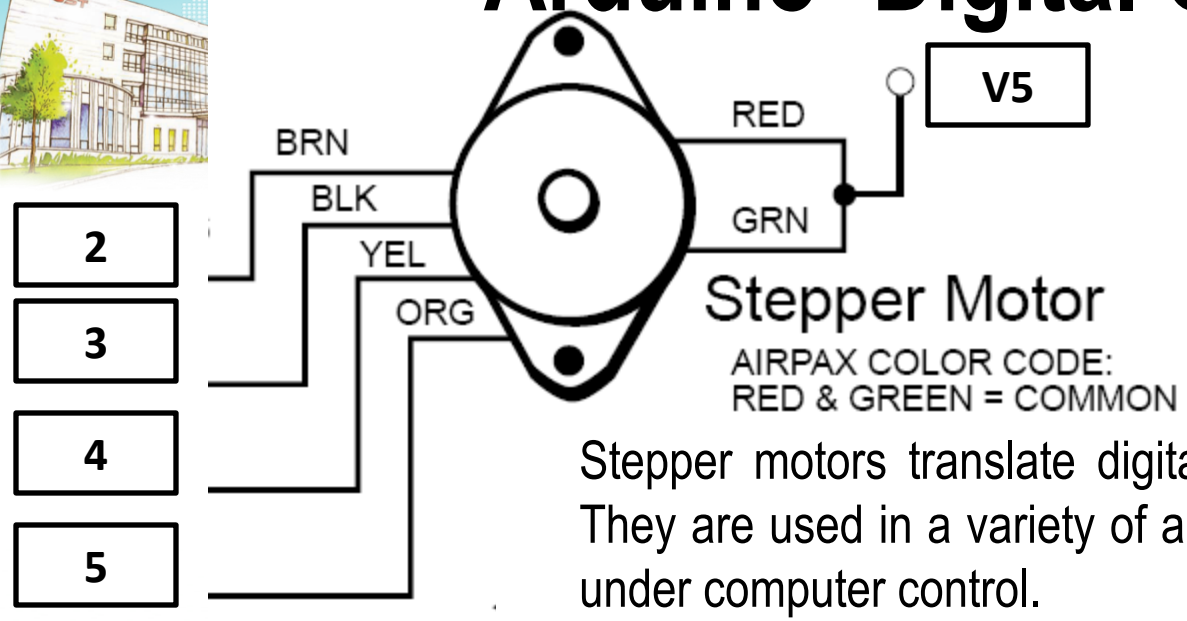
```
// Connect to Pin 13 and Ground
void setup(){
pinMode(13, OUTPUT); // Specify Arduino Pin number
and output/input mode
}
void loop(){
digitalWrite(13, HIGH); // Turn on Pin 13 sending a
HIGH Signal
delay(1000);          // Wait for one second
digitalWrite(13, LOW);  // Turn off Pin 13 sending a
LOW Signal
delay(3000);          // Wait for Three second
}
```

Code for Rotation/No Rotation

```
// Connect to Pin 13 and 12
void setup(){
pinMode(13, OUTPUT); // Specify Arduino Pin number
and output/input mode
pinMode(12, OUTPUT);
}
void loop(){
digitalWrite(13, HIGH); // Turn on Pin 13 sending a
HIGH Signal
digitalWrite(12, LOW);  //Make Pin 12 a Ground
delay(1000);          // Wait for one second
digitalWrite(13, LOW);  // Make Pin 13 a Ground
digitalWrite(12, HIGH); // Turn on Pin 12 sending a
HIGH Signal
delay(3000);          // Wait for Three second
}
```

Code for CW and CCW Rotation

Arduino- Digital Output–Rotation–Stepper Motor



Stepper motors translate digital switching sequences into motion. They are used in a variety of applications requiring precise motions under computer control.

Unlike ordinary dc motors, which spin freely when power is applied,steppers require that their power source be continuously pulsed in specific patterns. These patterns, or step sequences, determine the speed and direction of a stepper's motion.

For each pulse or step input, the stepper motor rotates a fixed angular increment; typically 1.8 or 7.5 degrees.

Steppers are driven by the interaction (attraction and repulsion) of magnetic fields. The driving magnetic field "rotates" as strategically placed coils are switched on and off. This pushes and pulls at permanent magnets arranged around the edge of a rotor that drives the output shaft.

Unipolar Stepper Motor

Full Step, Low Torque

Full Step, High Torque (standard application)

Half Step (best precision):

Stepper Motor

When the on-off pattern of the magnetic fields is in the proper sequence, the stepper turns (when it's not, the stepper sits and quivers).

The most common stepper is the four-coil unipolar variety. These are called unipolar because they require only that their coils be driven on and off. Bipolar steppers require that the polarity of power to the coils be reversed.

	Step Sequence				
	1	2	3	4	1
coil 1	1	1	0	0	1
coil 2	0	0	1	1	0
coil 3	1	0	0	1	1
coil 4	0	1	1	0	0

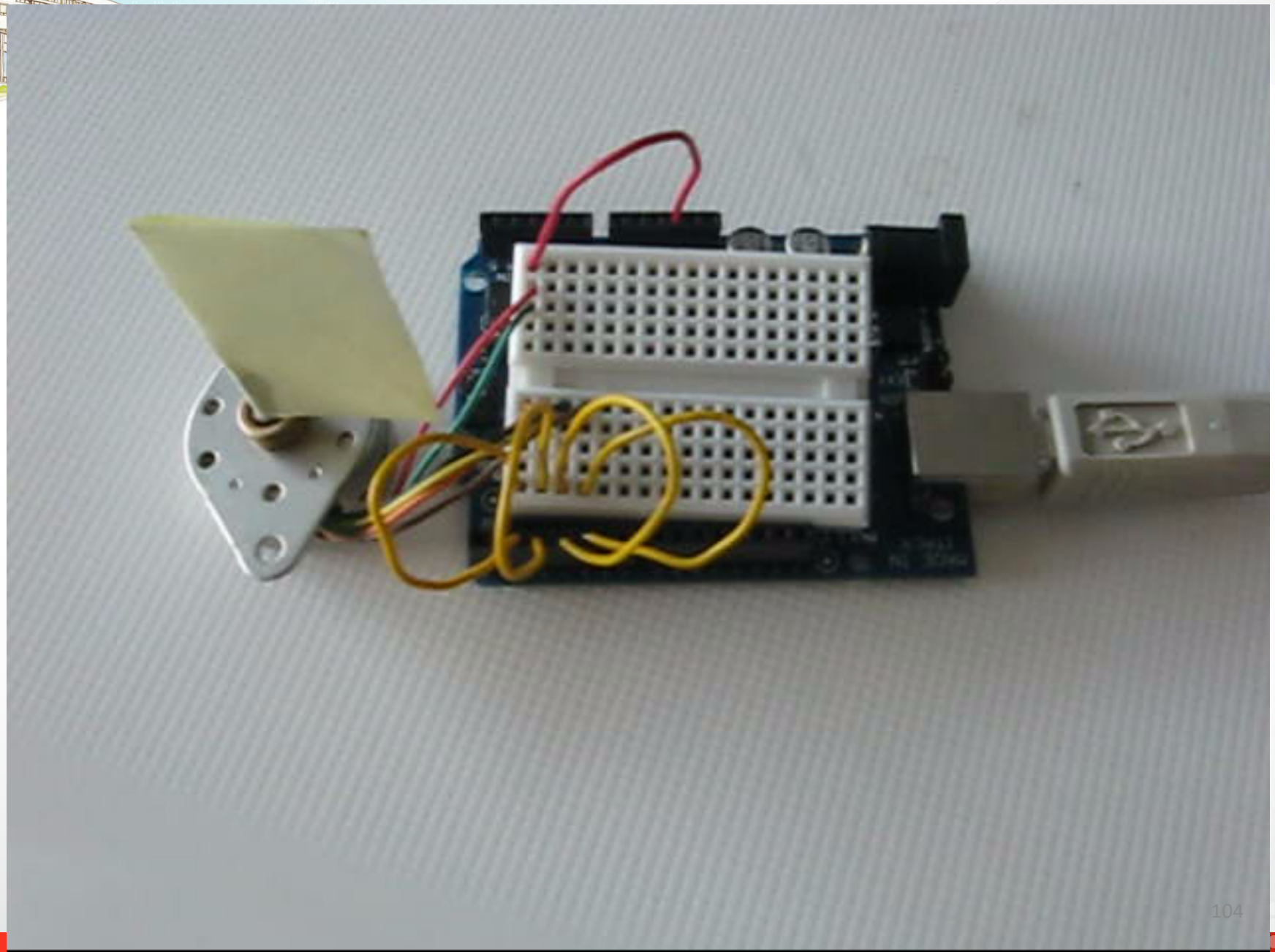
Figure 2. Normal stepping sequence.

The normal stepping sequence for four-coil unipolar steppers appears in the figure. If you run the stepping sequence in the figure forward, the stepper rotates clockwise; run it backward, and the stepper rotates counterclockwise.

The motor's speed depends on how fast the controller runs through the step sequence. At any time the controller can stop in mid sequence.

If it leaves power to any pair of energized coils on, the motor is locked in place by their magnetic fields. This points out another stepper motor benefit: built-in brakes.

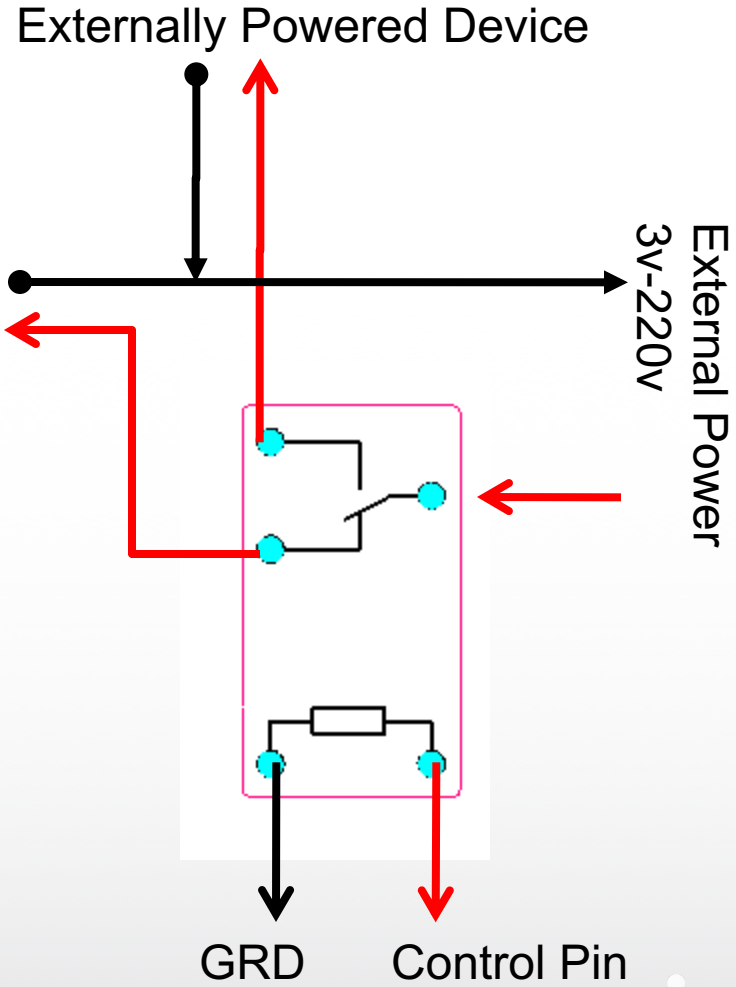
Stepper Motor



Stepper Motor-Direction and Speed

```
void setup(){
 pinMode(2,OUTPUT);
 pinMode(3,OUTPUT);
 pinMode(4,OUTPUT);
 pinMode(5,OUTPUT);
}
void loop(){
 // Pause between the types that determines the speed
 int stepperSpeed=200;// Change to change speed
 int dir=1;// change to -1 to change direction
 if (dir==1){  //Running Clockwise
 digitalWrite(2,HIGH);//Step 1
 digitalWrite(3,LOW);
 digitalWrite(4,HIGH);
 digitalWrite(5,LOW);
 delay(stepperSpeed);// Pause between the types that determines the speed
 digitalWrite(2,HIGH);//Step 2
 digitalWrite(3,LOW);
 digitalWrite(4,LOW);
 digitalWrite(5,HIGH);
 delay(stepperSpeed);// Pause between the types that determines the speed
 digitalWrite(2,LOW);//Step 3
 digitalWrite(3,HIGH);
 digitalWrite(4,LOW);
 digitalWrite(5,HIGH);
 delay(stepperSpeed);// Pause between the types that determines the speed
 digitalWrite(2,LOW);//Step 4
 digitalWrite(3,HIGH);
 digitalWrite(4,HIGH);
 digitalWrite(5,LOW);
 delay(stepperSpeed);// Pause between the types that determines the speed
 }
 if (dir==-1){  //Running CounterClockwise
 digitalWrite(2,LOW);//Step 4
 digitalWrite(3,HIGH);
 digitalWrite(4,HIGH);
 digitalWrite(5,LOW);
 delay(stepperSpeed);// Pause between the types that determines the speed
 digitalWrite(2,LOW);//Step 3
 digitalWrite(3,HIGH);
 digitalWrite(4,LOW);
 digitalWrite(5,HIGH);
 delay(stepperSpeed);// Pause between the types that determines the speed
 digitalWrite(2,HIGH);//Step 2
 digitalWrite(3,LOW);
 digitalWrite(4,LOW);
 digitalWrite(5,HIGH);
 delay(stepperSpeed);// Pause between the types that determines the speed
 digitalWrite(2,HIGH);//Step1
 digitalWrite(3,LOW);
 digitalWrite(4,HIGH);
 digitalWrite(5,LOW);
 delay(stepperSpeed);// Pause between the types that determines the speed
 }
}
```

Arduino-Digital Output – Controling any Electrical Device with any power needs using a relay




```
// Connect to Pin 13 and Ground
void setup(){
pinMode(13, OUTPUT); // Specify Arduino Pin number and
output/input mode
}
void loop(){
digitalWrite(13, HIGH); // Turn on Pin 13 sending a HIGH Signal
delay(1000);          // Wait for one second
digitalWrite(13, LOW);  // Turn off Pin 13 sending a LOW Signal
delay(3000);          // Wait for Three second
}
```

IO Pins

Two states (binary signal) vs. multiple states (continuous signal)

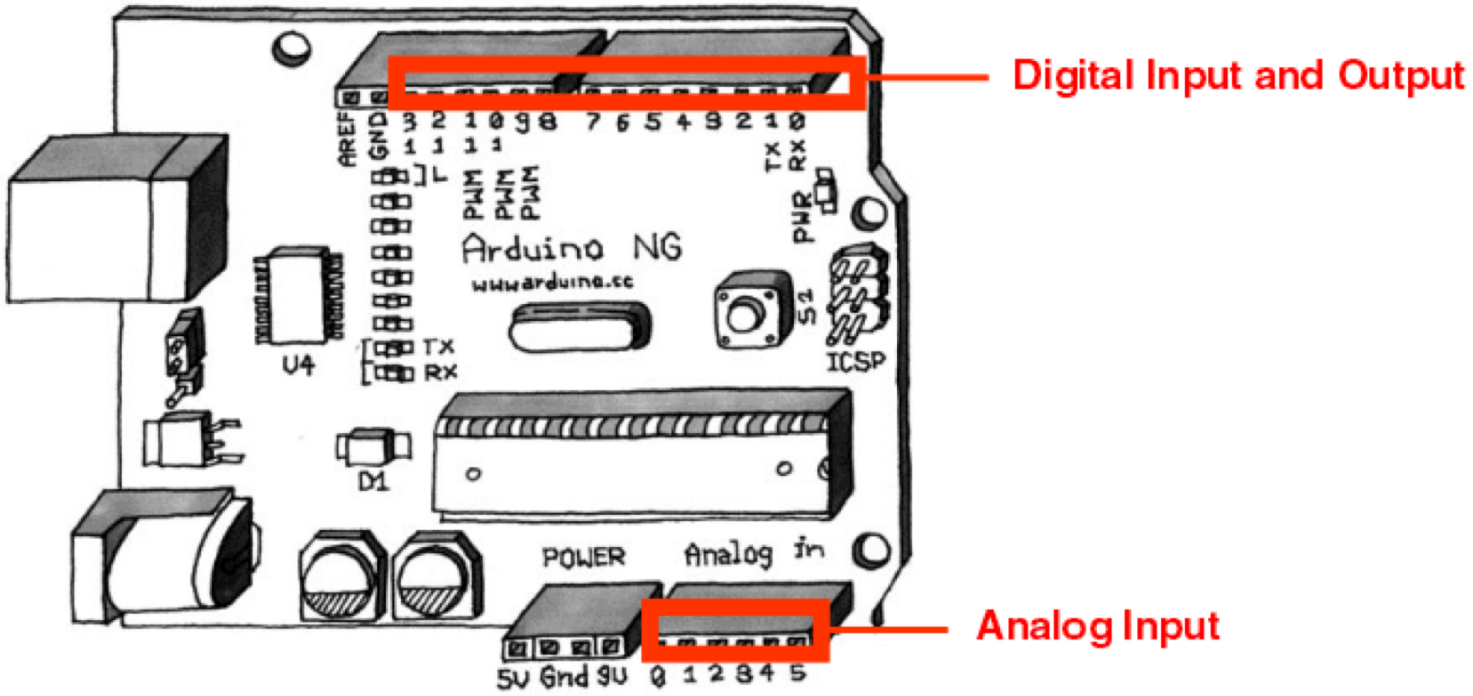
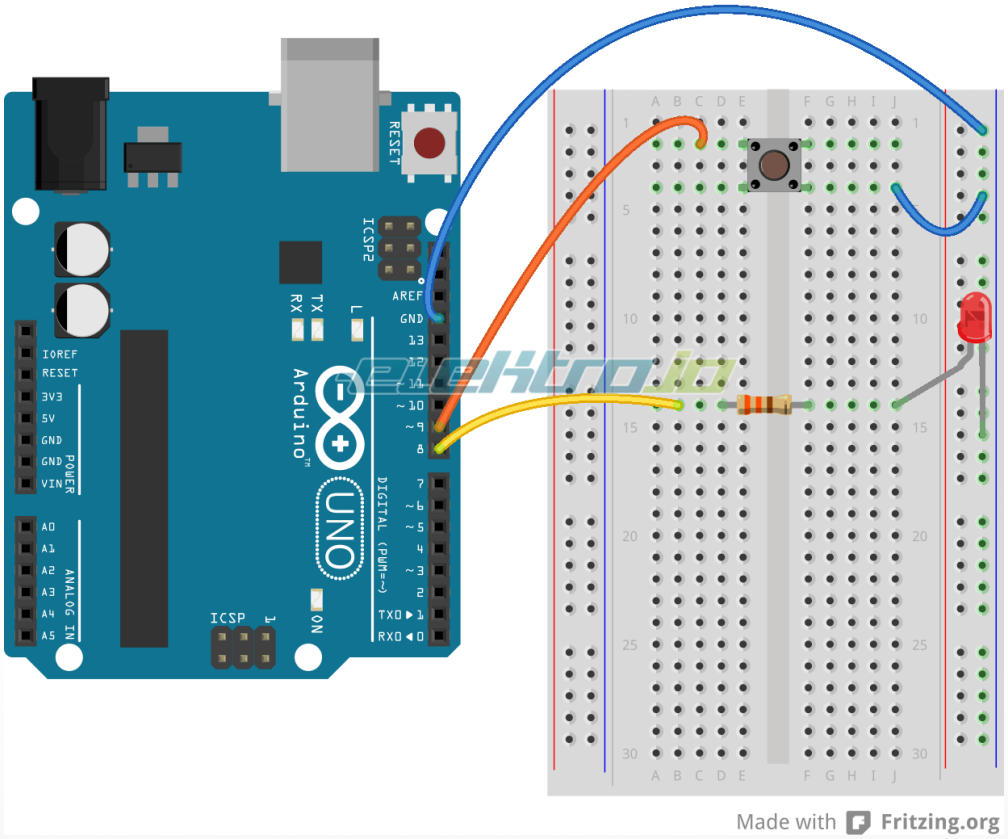


Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley

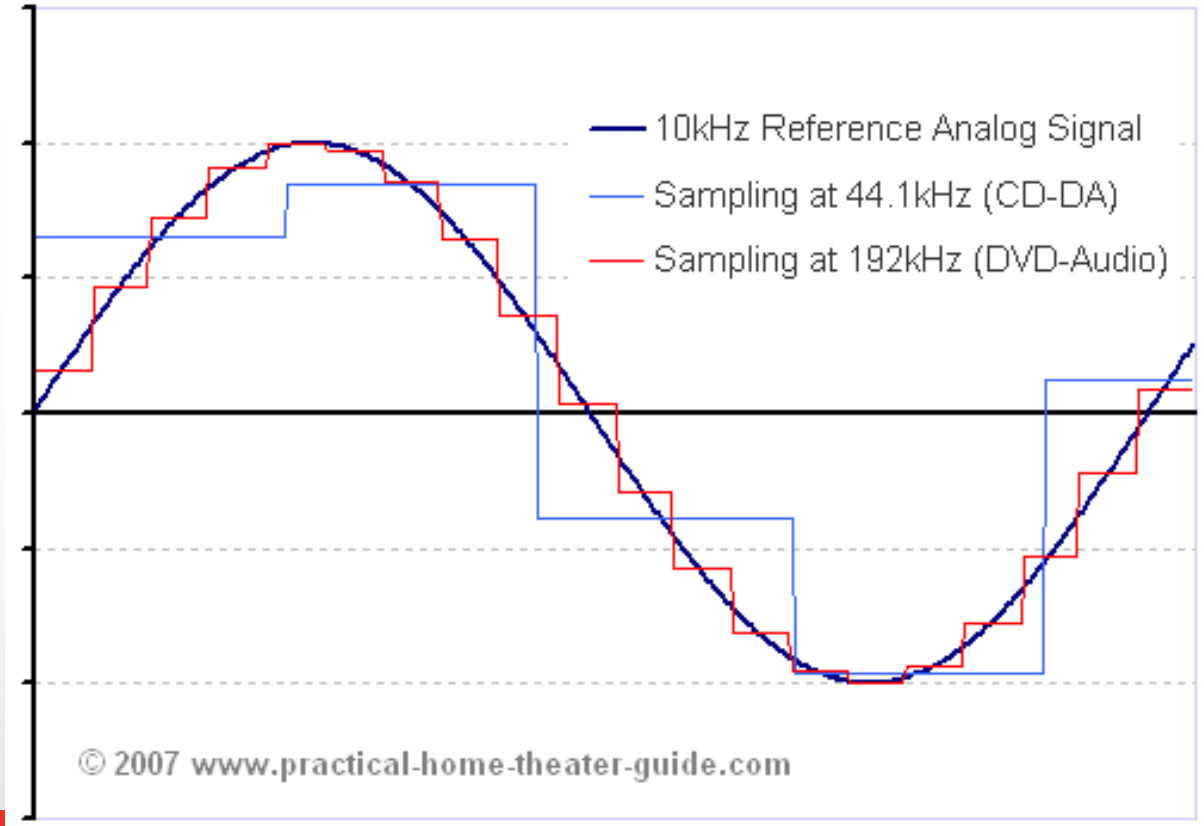
In-class Exercise 1: Digital IO



- Use a push-button to turn ON/Off LED

Topic 3: Analog Input

- Think about music stored on a CD---an analog signal captured on digital media
  - Sample rate
  - Word length



Arduino Analog Input

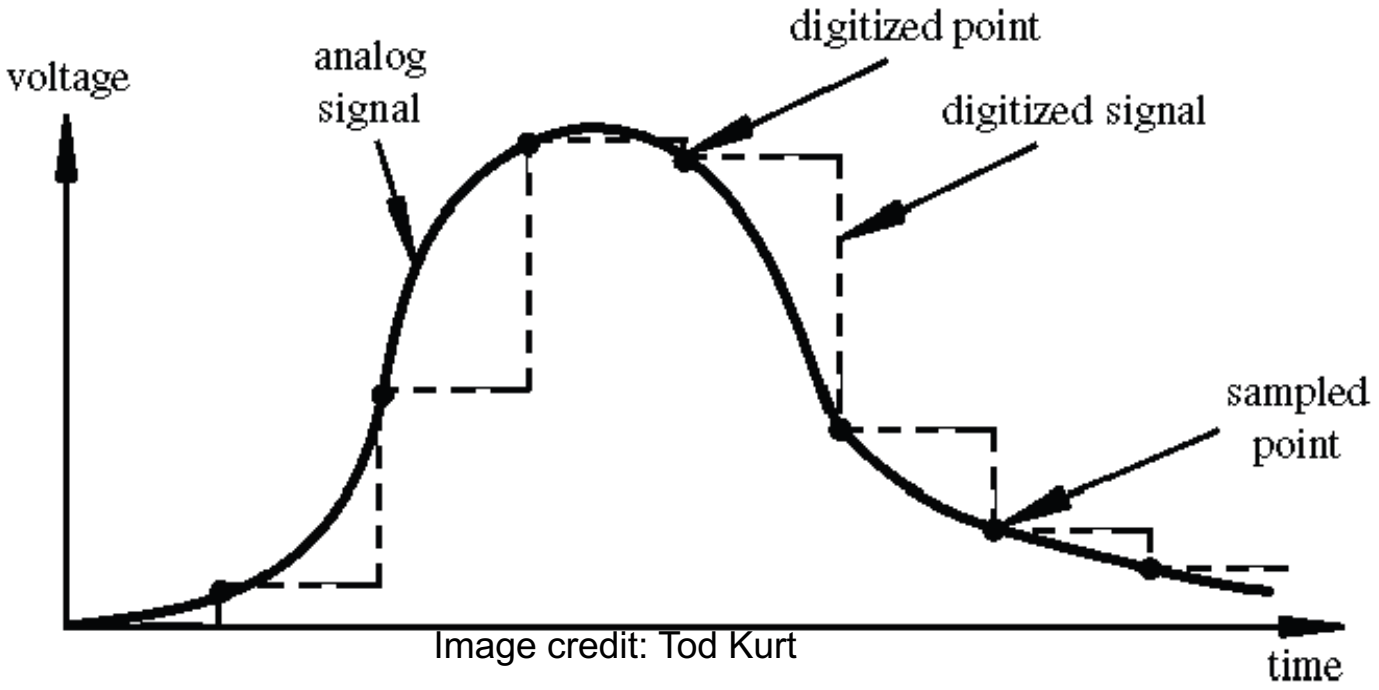
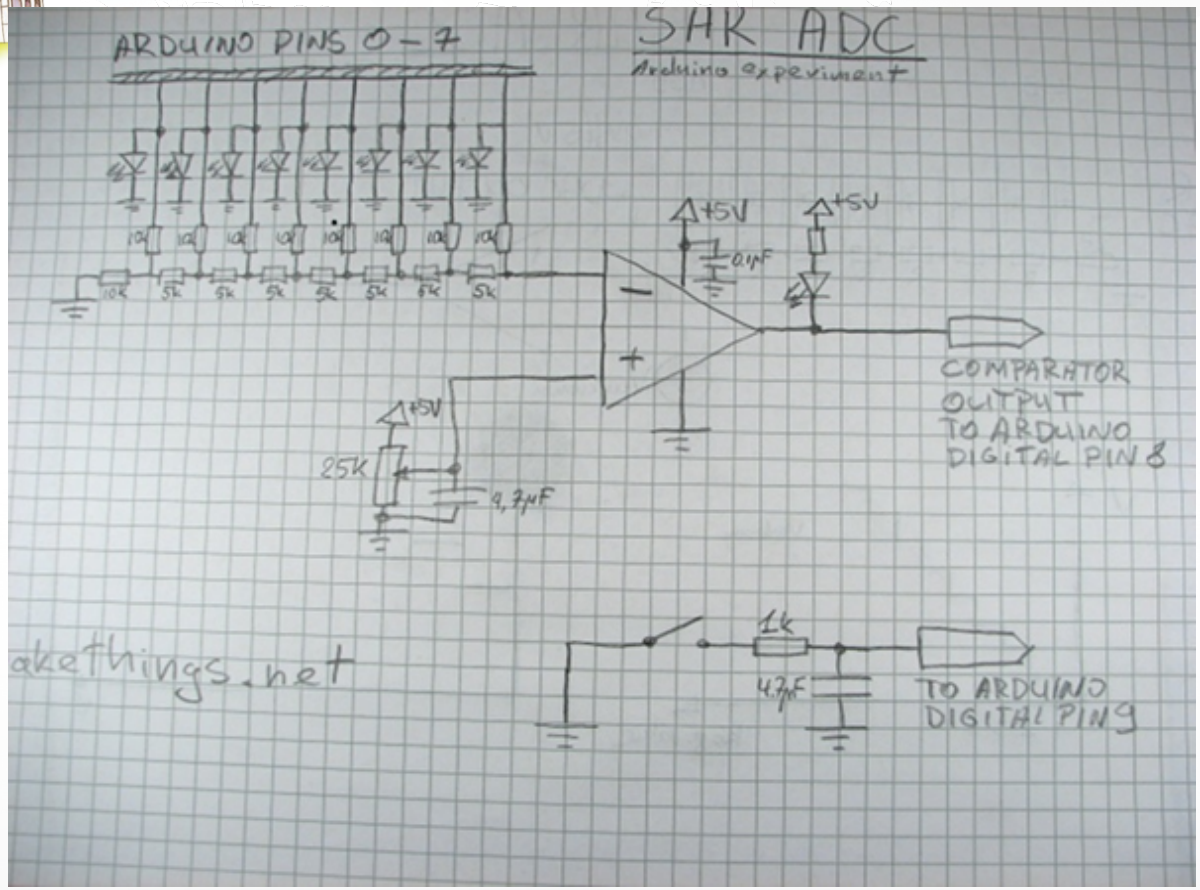


Image credit: Tod Kurt

- *Resolution*: the number of different voltage levels (i.e., *states*) used to discretize an input signal
- Resolution values range from 256 states (8 bits) to 4,294,967,296 states (32 bits)
- The Arduino uses 1024 states (10 bits)
- Smallest measurable voltage change is 5V/1024 or 4.8 mV
- Maximum sample rate is 10,000 times a second

How does ADC work?



- How does ADC work
- Excel Demonstration

Topic 3: Analog Output

- Can a digital devise produce analog output?

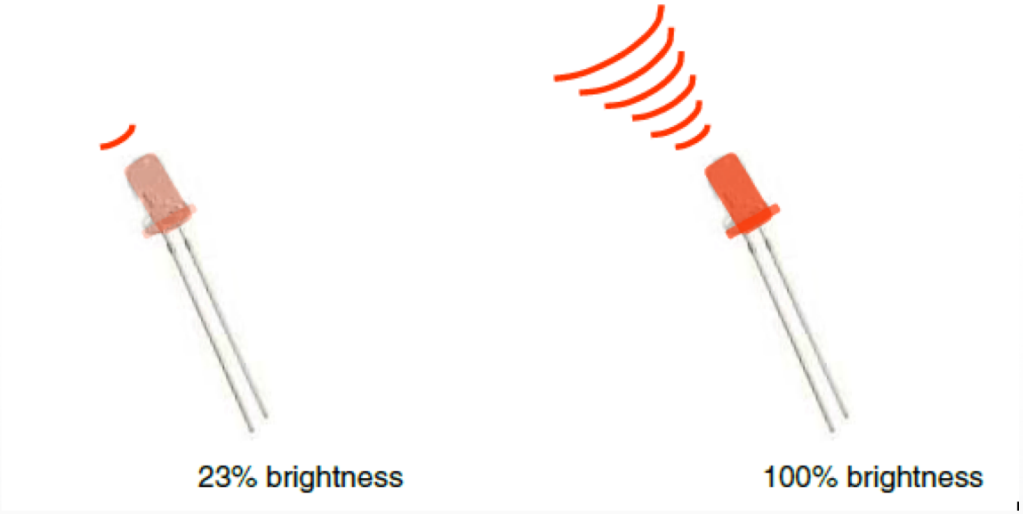


Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley

- Analog output can be simulated using pulse width modulation (PWM)

Pulse Width Modulation

- Can't use digital pins to directly supply say 2.5V, but can pulse the output on and off really fast to produce the same effect
- The on-off pulsing happens so quickly, the connected output device "sees" the result as a reduction in the voltage

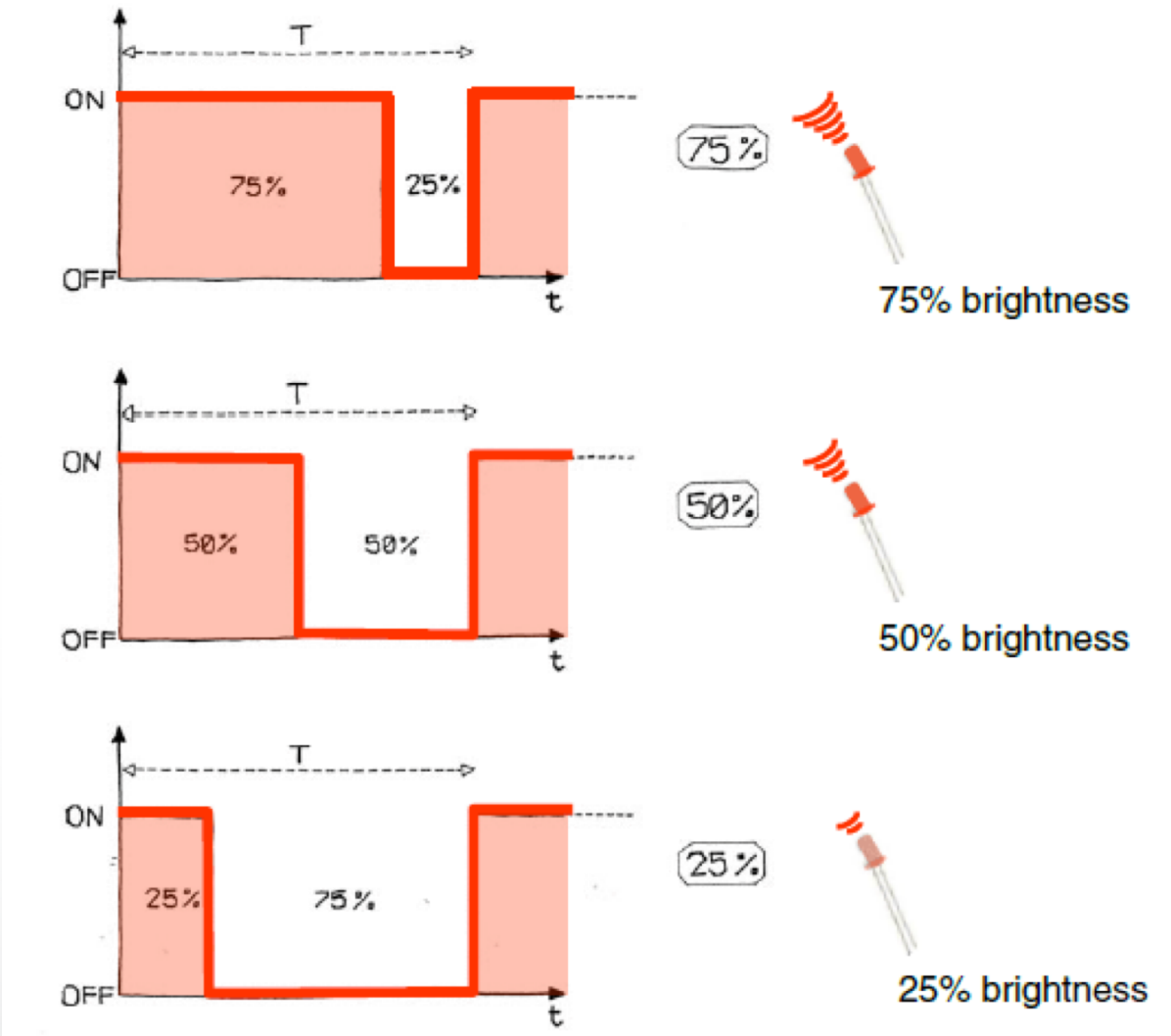


Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley

PWM Duty Cycle

output voltage = (on_time / cycle_time) * 5V

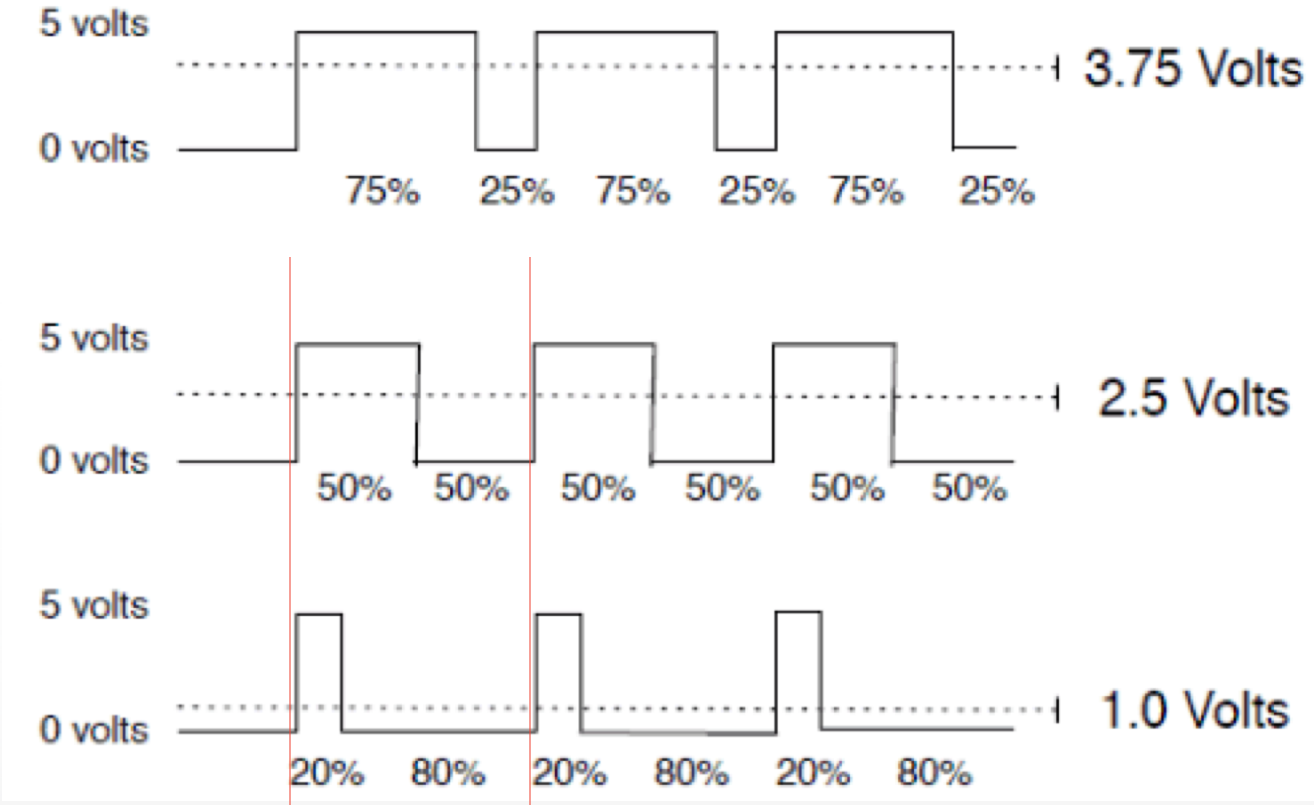
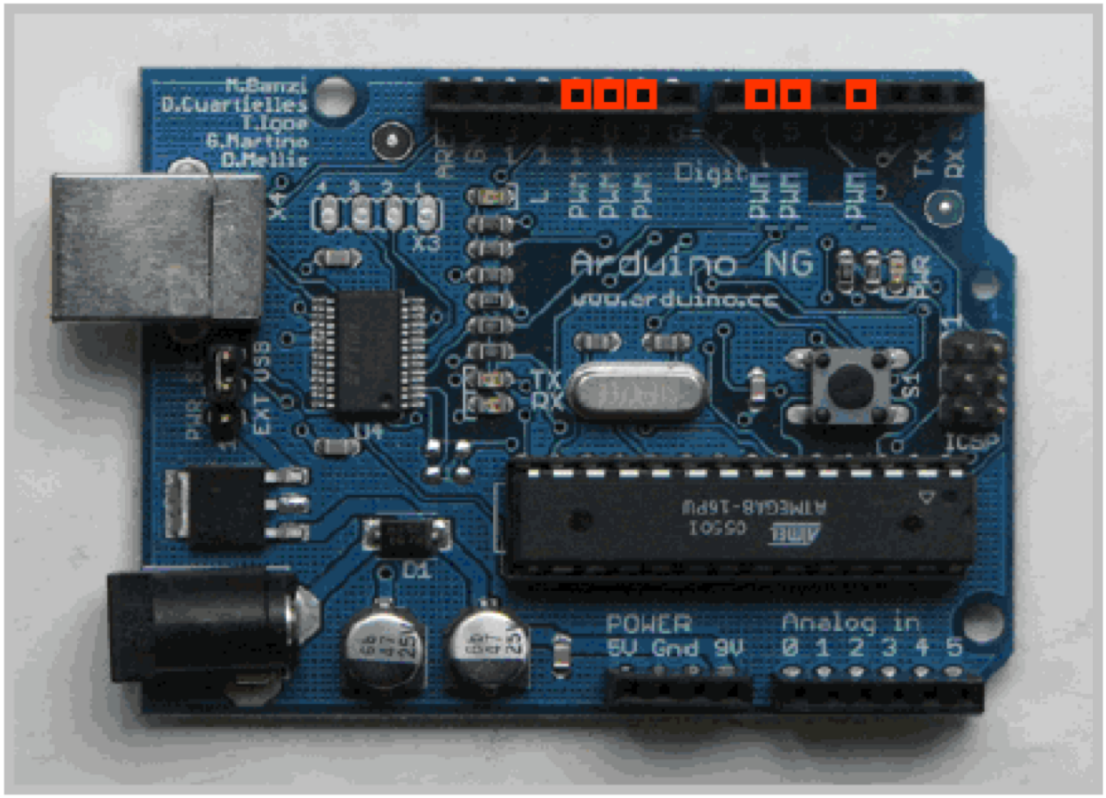


Image credit: Tod Kurt

Fixed cycle length; constant number of cycles/sec

PMW Pins

Your Arduino board has built in PWM circuits, on pins 3, 5, 6, 9, 10, and 11



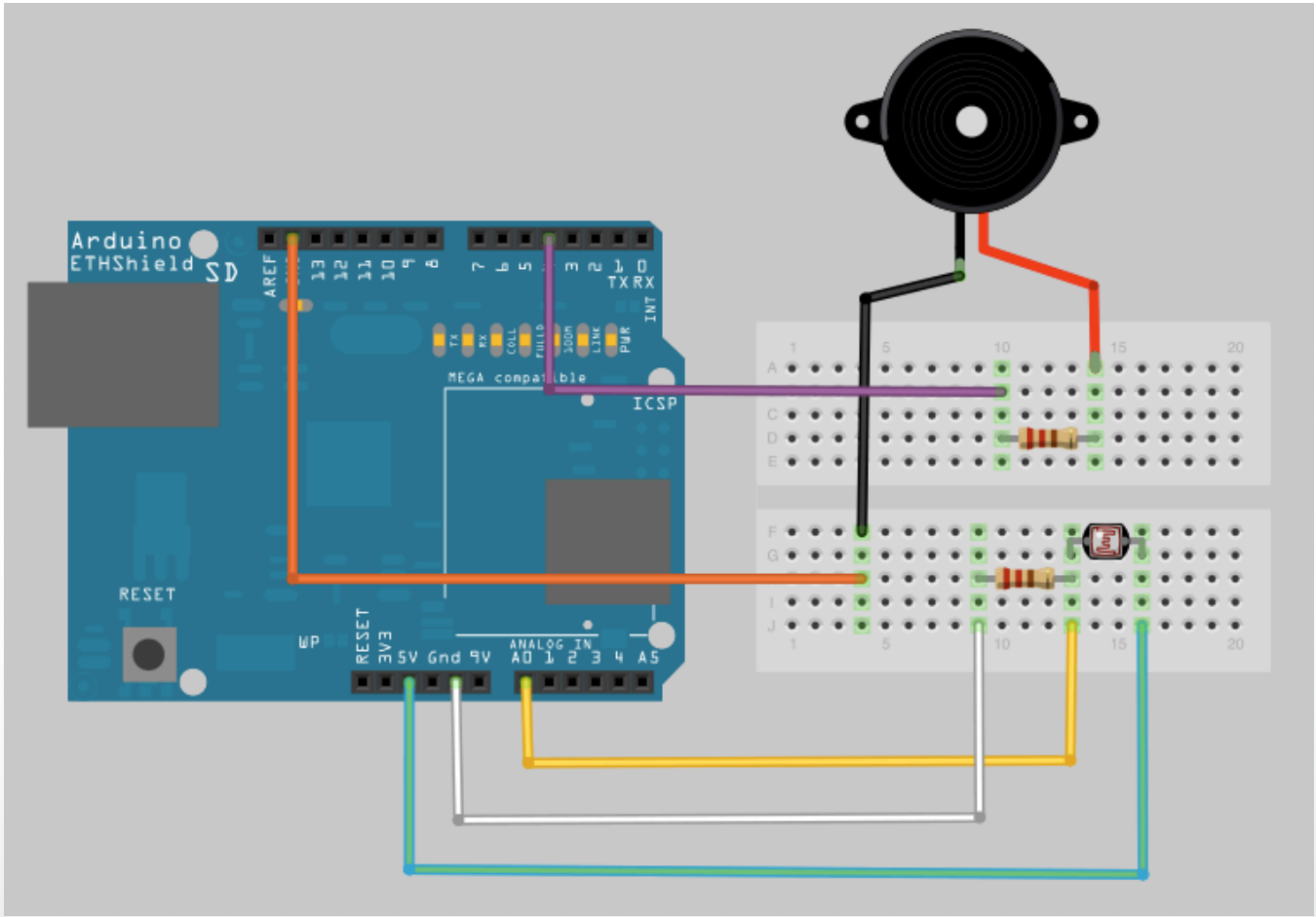
- Command: **analogWrite(pin,value)**
- value is duty cycle: between 0 and 255
- Examples:
  analogWrite(9, 128)
  for a 50% duty cycle

  analogWrite(11, 64)
  for a 25% duty cycle

Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley

In-class Exercise 2: Analog IO

Part 1:



A light theremin

In-class Exercise 2: Analog IO

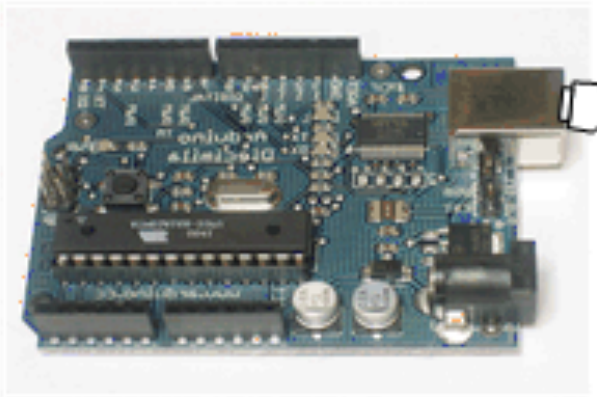
Part 2: Add an LED

- Add a 330 ohm resistor and an LED to pin 9
- Using the analogWrite() command, set the intensity of the LED as a function of the value of prReading

Topic 4: Serial Communication

Cable

1 0 0 1 0 1 0 0 1 1 0 . . .



Information passes between the computer and Arduino through the USB cable. Information is transmitted as zeros ('0') and ones ('1')… also known as **bits**!

Image from http://www.ladyada.net/learn/arduino/lesson4.html

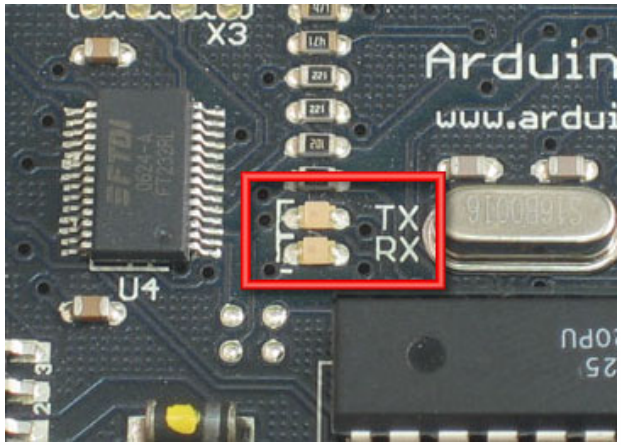
Serial Communications

- "Serial" because data is broken down into bits, each sent one after the other down a single wire.
- The single ASCII character 'B' is sent as:

```
'B' =  0 1 0 0 0 0 1 0
    =  L H L L L L H L
    =
```

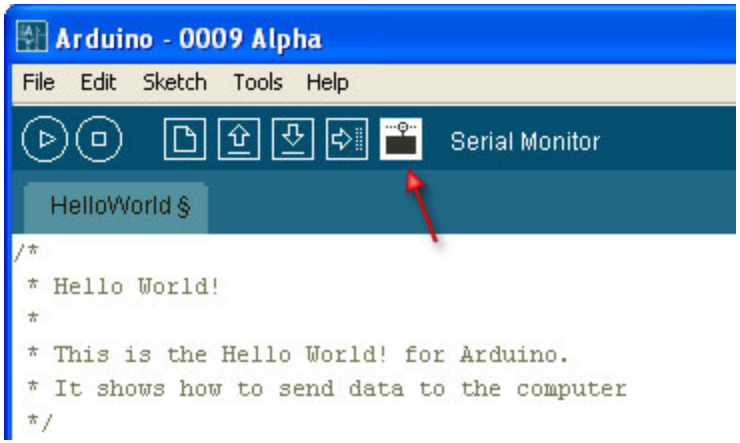
- Toggle a pin to send data, just like blinking an LED
- You could implement sending serial data with `digitalWrite()` and `delay()`
- A single data wire needed to send data. One other to receive.

Serial Communication



- ***Compiling*** turns your program into binary data (ones and zeros)
- ***Uploading*** sends the bits through USB cable to the Arduino
- The two LEDs near the USB connector blink when data is transmitted
  - **RX** blinks when the Arduino is receiving data
  - **TX** blinks when the Arduino is transmitting data

Open the Serial Monitor and Upload the Program

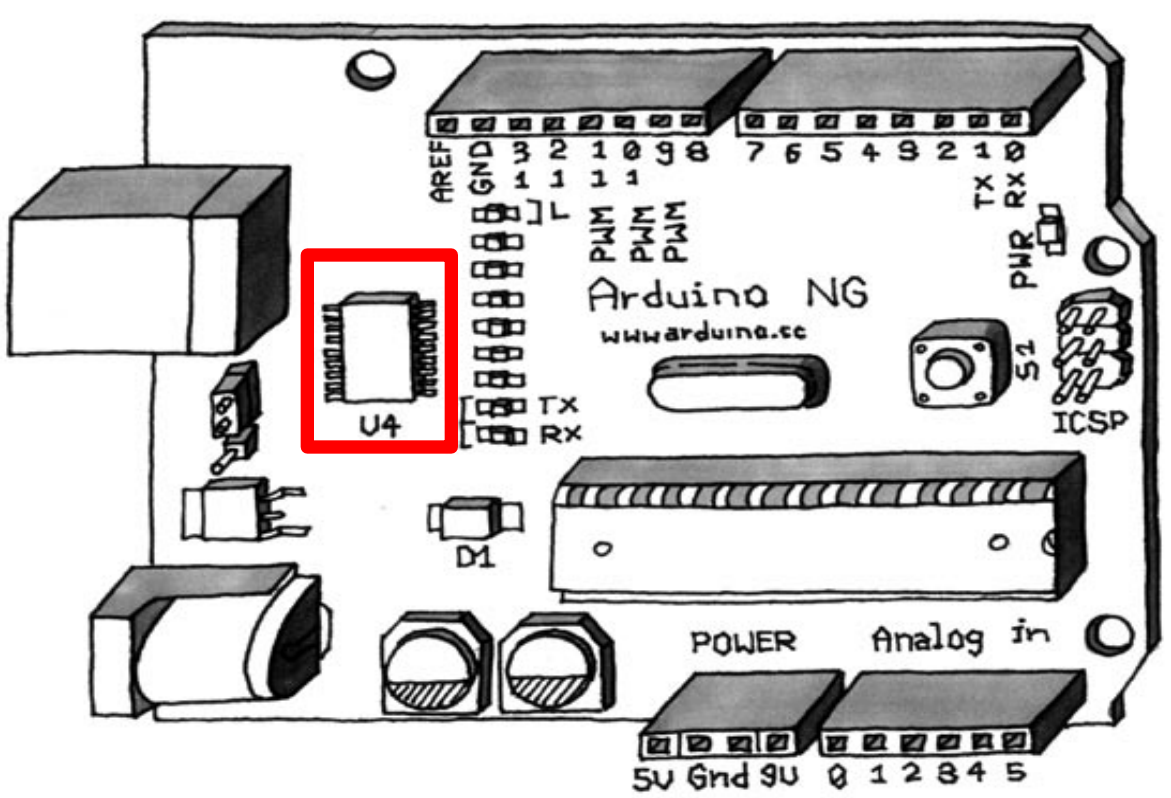


Some Commands

- Serial.begin()
  - e.g., Serial.begin(9600)
- Serial.print() or Serial.println()
  - e.g., Serial.print(value)
- Serial.read()
- Serial.available()
- Serial.write()
- Serial.parseInt()

- Example Program

Serial-to-USB chip---what does it do?



The LilyPad and Fio Arduino require an external USB to TTY connector, such as an FTDI “cable”.
In the Arduino Leonardo a single microcontroller runs the Arduino programs and handles the USB connection.

Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley

Two different communication protocols

Serial (TTL):

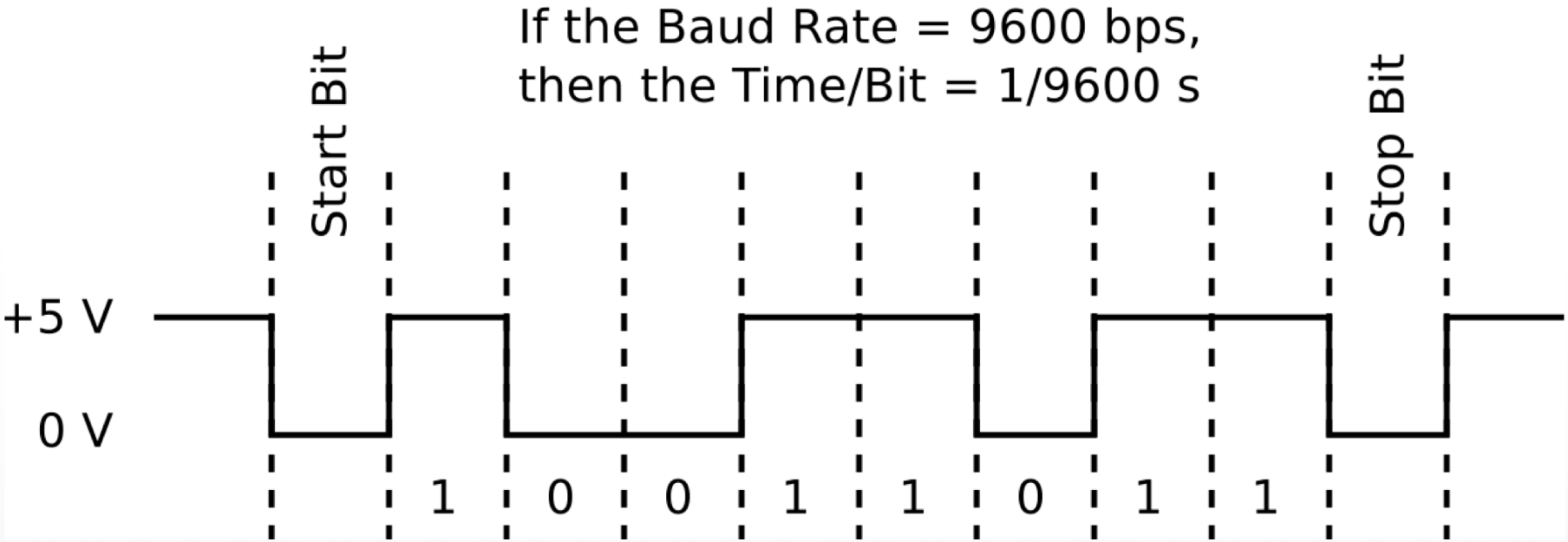


Image from http://www.fiz-ix.com/2013/02/introduction-to-arduino-serial-communication/

USB Protocol

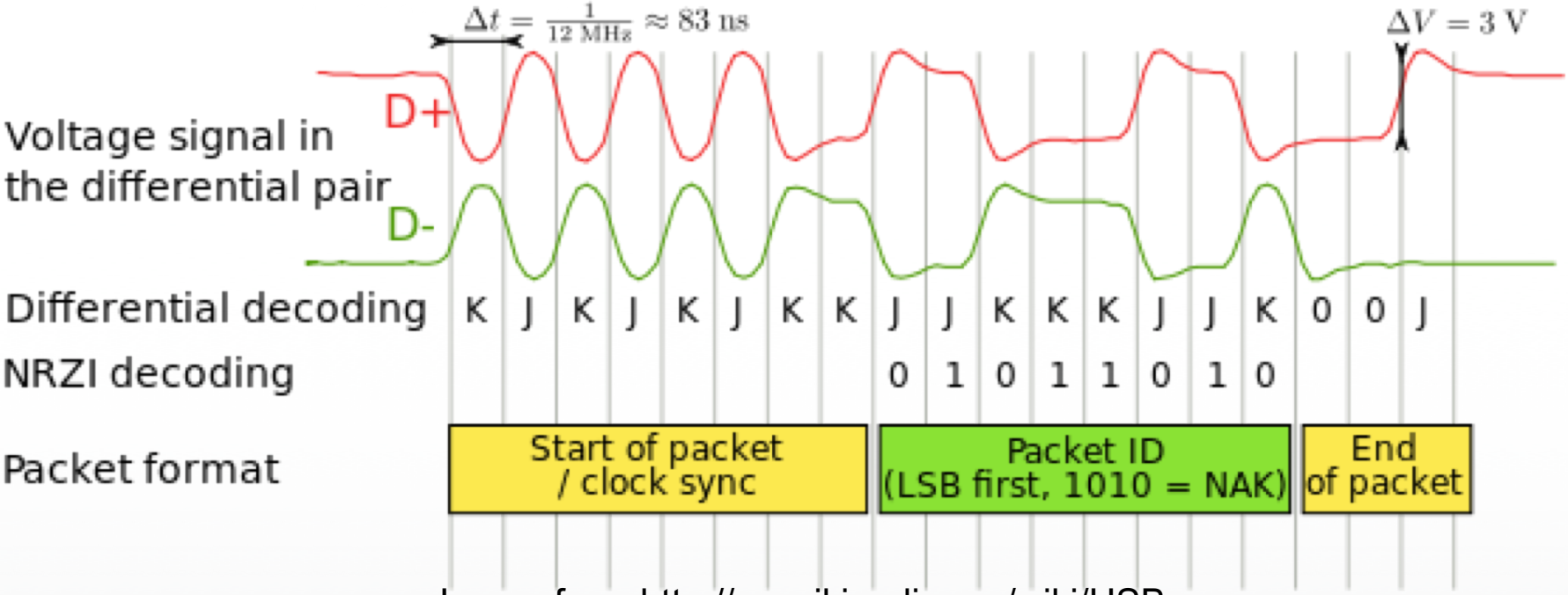


Image from http://en.wikipedia.org/wiki/USB

- Much more complicated

In-class Exercise 3: Serial Communication

Modify your program from in-class exercise “LED” to control the intensity of the LED attached to pin 9 based on keyboard input.

Use the Serial.parseInt() method to read numeric keyboard input as an integer.

An input of 9 should produce full intensity and an input of 0 should turn the LED off.