What programming language is this?

It is essentially C, using a couple pre-programmed libraries standard for the Arduino IDE. You can bypass the Arduino dialect of C entirely, if you want.

Arduino IDE:

```
digitalWrite(D13, ...
    HIGH);
```

C function:

```
void ...
    digitalWrite(uint8_t
    pin, uint8_t val)
{
uint8_t timer = ...
    digitalPinToTimer(pi
uint8_t bit = ...
    digitalPinToBitMask(
uint8_t port = ...
    digitalPinToPort(pin
volatile uint8_t
* out;
if (port ==
```

What programming language is this?

- Controlling microcontroller peripherals is performed using special memory locations called registers.
- Registers can be read and written.
- Imagine them as banks of 8 switches, connecting and disconnecting wires inside the chip.

Arduino IDE:

```
pinMode(D13, OUTPUT)
digitalWrite(D13, ...
    HIGH);
```

Writing to a register in C (preprocessor definitions):

```
DDRB  = B00100000;
PORTB = B00100000;
```

What just happened?

- The IDE looks simple, but there is a lot going on.
- The Arduino IDE is not the only way of programming the microcontroller.
- The bootloader - bypassing programmers.

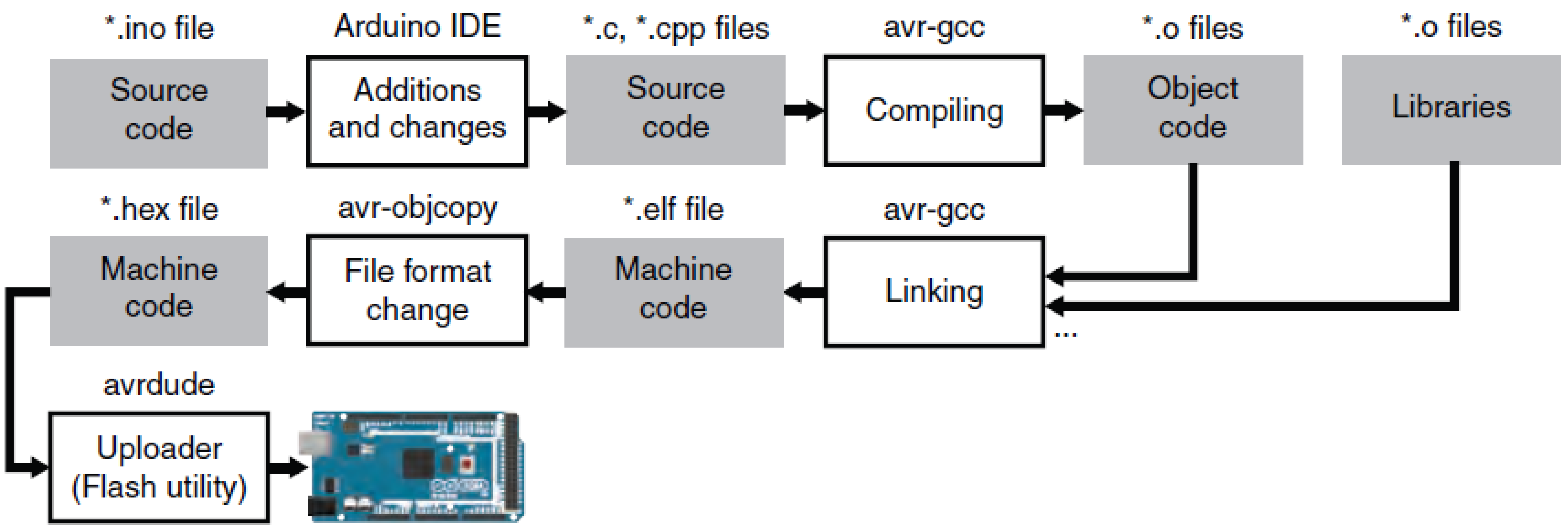


Figure: The hidden compilation process in the Arduino IDE.

Data types

Data types in C refers to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.

The following table provides all the data types that you will use during Arduino programming.

void	Boolean	char	Unsigned char	byte	int	Unsigned int	word
long	Unsigned long	short	float	double	array	String-char array	String-object

void

The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

```
Void Loop ( )
{
   // rest of the code
}
```

Data types

boolean	single bit FALSE/TRUE (really uses a whole byte of memory)	
Unsigned Integers		
byte	eight bits	0 to 255
word	two bytes	0 to 65535
unsigned long	4 bytes	0 to 4,294,967,295
Signed Integers		
char	eight bits	-128 to 127
short	two bytes	-32,768 to 32,767
long	4 bytes	-2,147,483,648 to 2,147,483,647.

int Unsigned integer, the number of bytes used depends on the particular hardware used. For us int is the same as short.

Real Numbers

float Floating point number uses 4 bytes. Used for non-integers has 6-7 decimal point precision. -3.4028235E+38 to 3.4028235E+38

Data types

A single variable can store an array of values.

The index is contained in square brackets. Arrays are zero indexed (start at zero).

```
int threeints[3];   // ‘threeints’ is an array (0-2)
threeints[0]=15;
threeints[1]=10;
threeints[2]=threeints[0]-threeints[1];
```

Declaring/Initializing Variables

Before using a variable it must be declared.

int a; // creates an integer with the name ‘a’

When a variable is declared it can also be initialized.

int a=34;   // creates an integer with the name ‘a’ and assigns it the value 34.

The char data type is used to represent characters using ASCII encoding.
Single character constants are indicated with single quotes.

```
char A;
A='B';
```

'B' is encoded with ASCII and the resulting value of 66 is stored in A.

Boolean

A Boolean holds one of two values, true or false. Each Boolean variable occupies one byte of memory.

```
boolean  val = false ;  // declaration of variable with type boolean and initialize it with false

boolean   state = true ;   //    declaration of variable with type boolean and initialize it with false
```

Unsigned char

Unsigned char is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

```
Unsigned Char   chr_y = 121 ;       // declaration of variable with type Unsigned char and initialize it with character   y
```

Data types

An array of characters is called a string

char examplestring[8];

examplestring=“arduino”;

The last element of a sting is always the ‘null string’ which has an ASCII value of zero

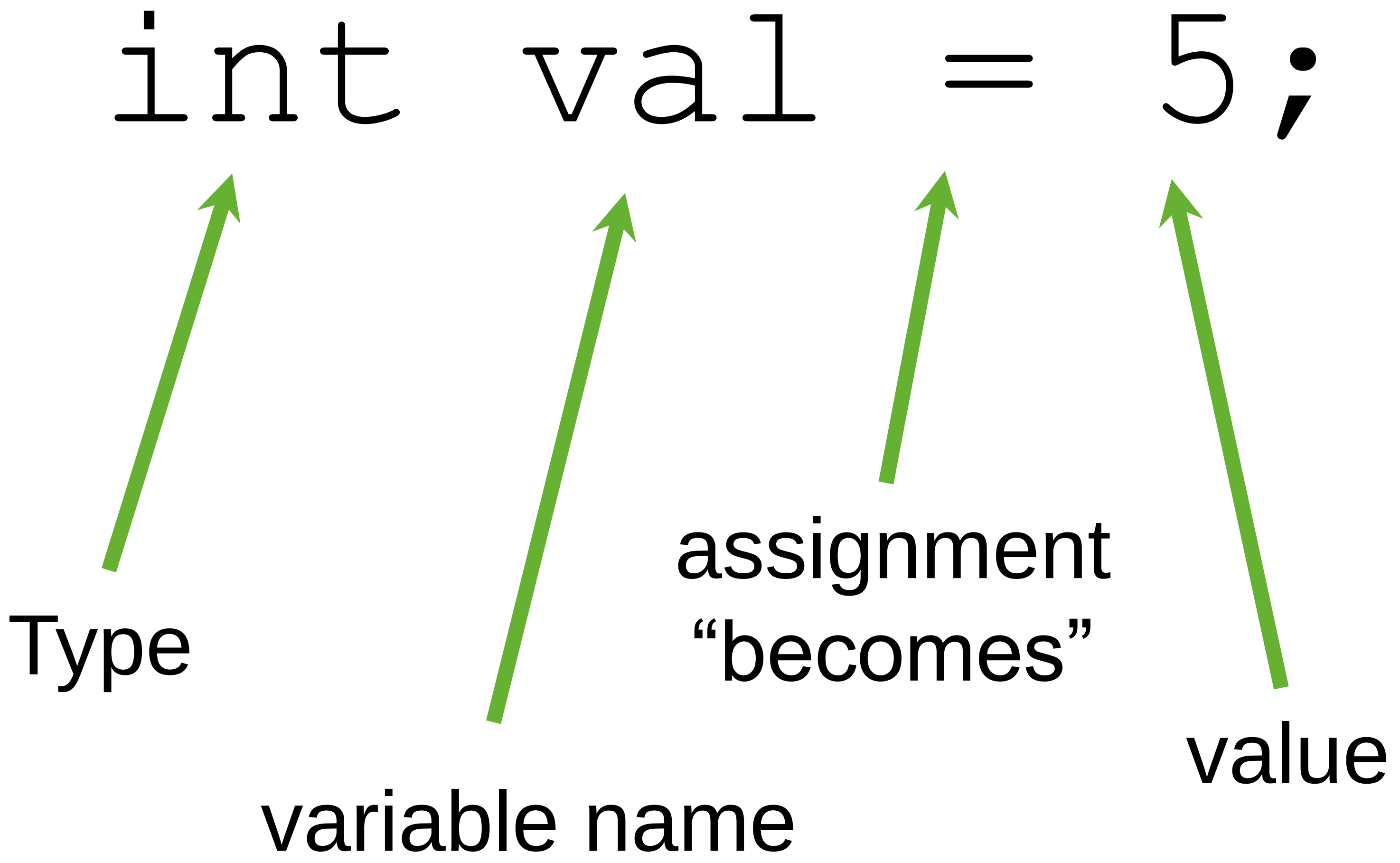
Stings can also be stored as objects using the String class.

Using String objects rather than character arrays uses more memory but adds functionality.

Character arrays are referred to as strings with a small s, and instances of the String class are referred to as Strings with a capital S.

Constant strings, specified in "double quotes" are treated as char arrays, not instances of the String class.

Declaring a variable



```
int val = 5;
```

Using variables

```
int delayTime = 2000;
int greenLED = 9;
void setup() {

     pinMode(greenLED, OUTPUT);

}

void loop() {

     digitalWrite(greenLED, HIGH);
     delay(delayTime);
     digitalWrite(greenLED, LOW);
     delay(delayTime);

}
```

Declare delayTime Variable

Use delayTime Variable

Using Variables

```
int delayTime = 2000;
int greenLED = 9;

void setup() {
      pinMode(greenLED, OUTPUT);
}
void loop() {
      digitalWrite(greenLED, HIGH);
      delay(delayTime);
      digitalWrite(greenLED, LOW);
 delayTime = delayTime - 100;
      delay(delayTime);
}
```

subtract 100 from delayTime to gradually increase LED's blinking speed

Serial vs. parallel communication

Classical asynchronous serial is robust, simple, widely used and the Uno can handle it!

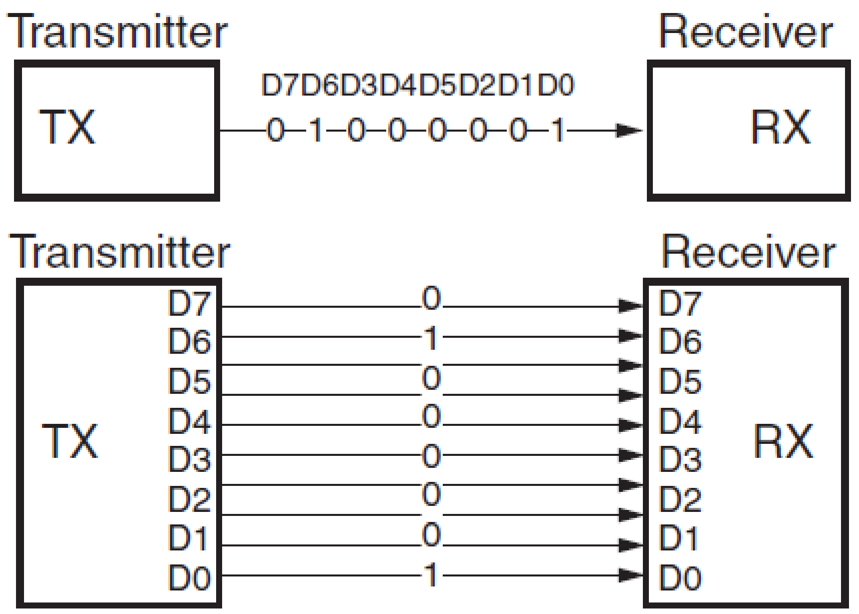


Figure: Serial (top) and parallel (bottom) modes of digital communication.

Serial ports

There is 1 pair of serial ports on the Uno (TX0,RX0) and 4 pairs on the Mega.

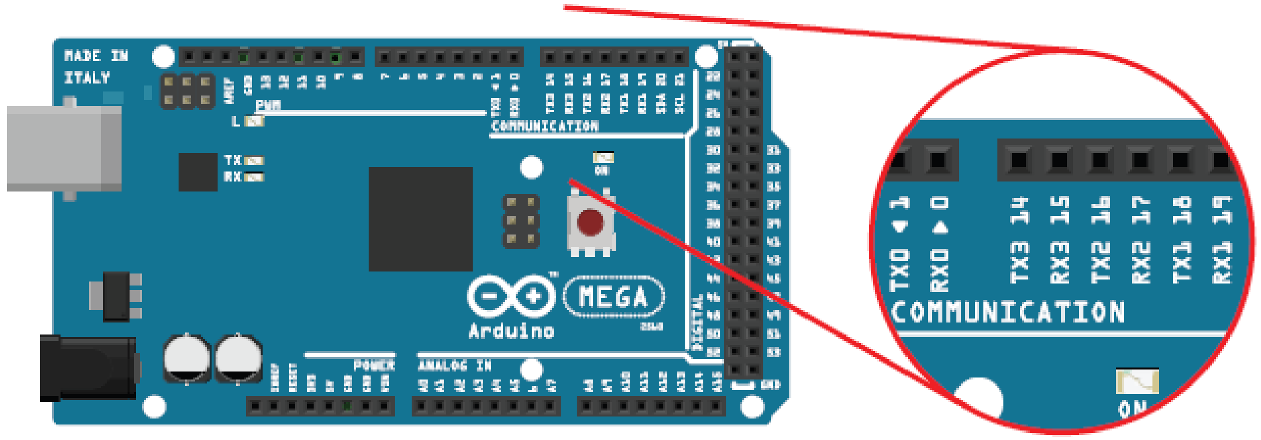


Figure: The 4 pairs of UART pins on an Arduino Mega 2560.

Connecting serial devices

- Cross connect wires
- Need 1 wire each direction + ground
- Not a bus, end-to-end communication

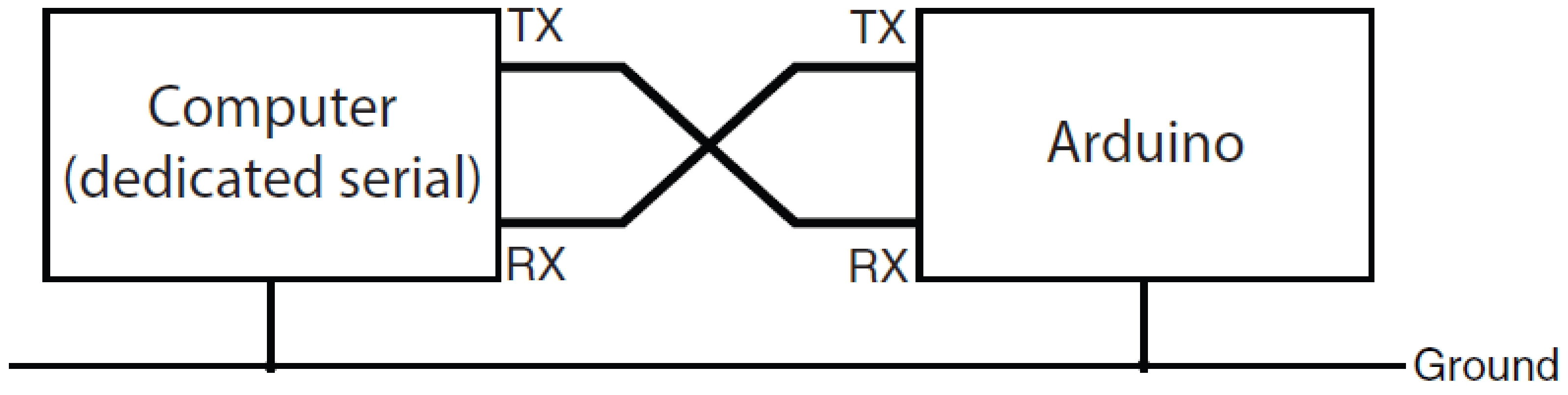


Figure: Connection scheme for serial communication.

Serial communication example

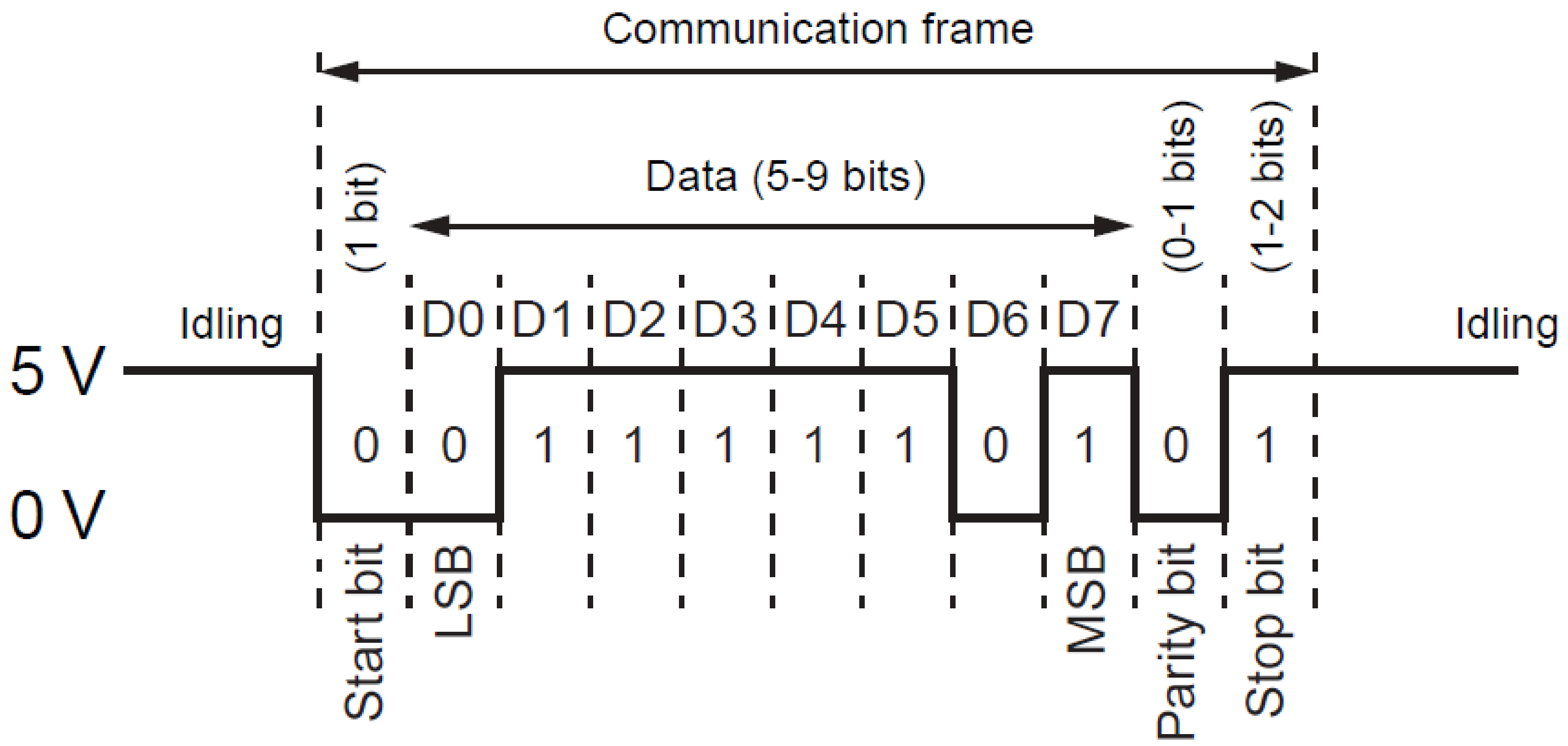


Figure: An example of serial communication over the line with 8 data bits, an even parity bit and one stop bit.

Serial communication example

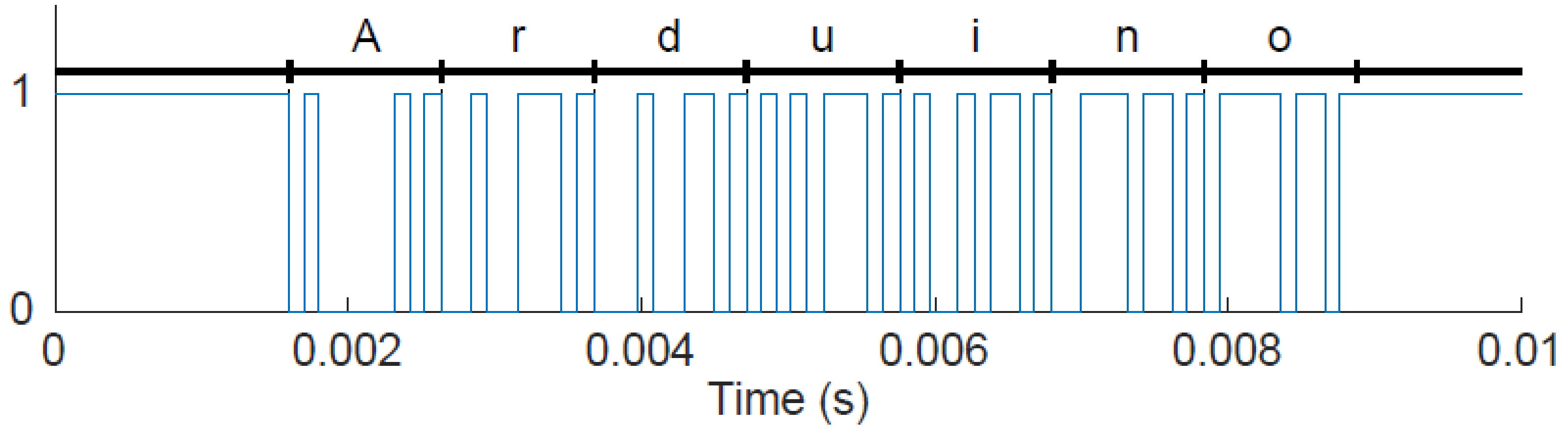


Figure: Asynchronous serial transfer of the string of characters “Arduino”.

Serial communication in the Arduino

Set the speed in *baud* (communication rate) once:

```
Serial.begin(9600);
\pause
```

Print an arbitrary text:

```
Serial.print("My text.");
```

This is the same as:

```
Serial.print("My ");
Serial.print("te");
Serial.print("xt.");
```

Meaning that you can separate one line of text to different parts, comes in handy when you write variables.

Serial communication in the Arduino

Print a variable:

```
Serial.print(myVariable);
```

Number of decimal places:

```
Serial.print(myVariable,4);
```

Format the data: `BIN`, `HEX`, `OCT` and `DEC`:

```
Serial.print(myVariable,BIN);
```

Insert a line ending and move to next line

```
Serial.println(myVariable);
```

Serial communication

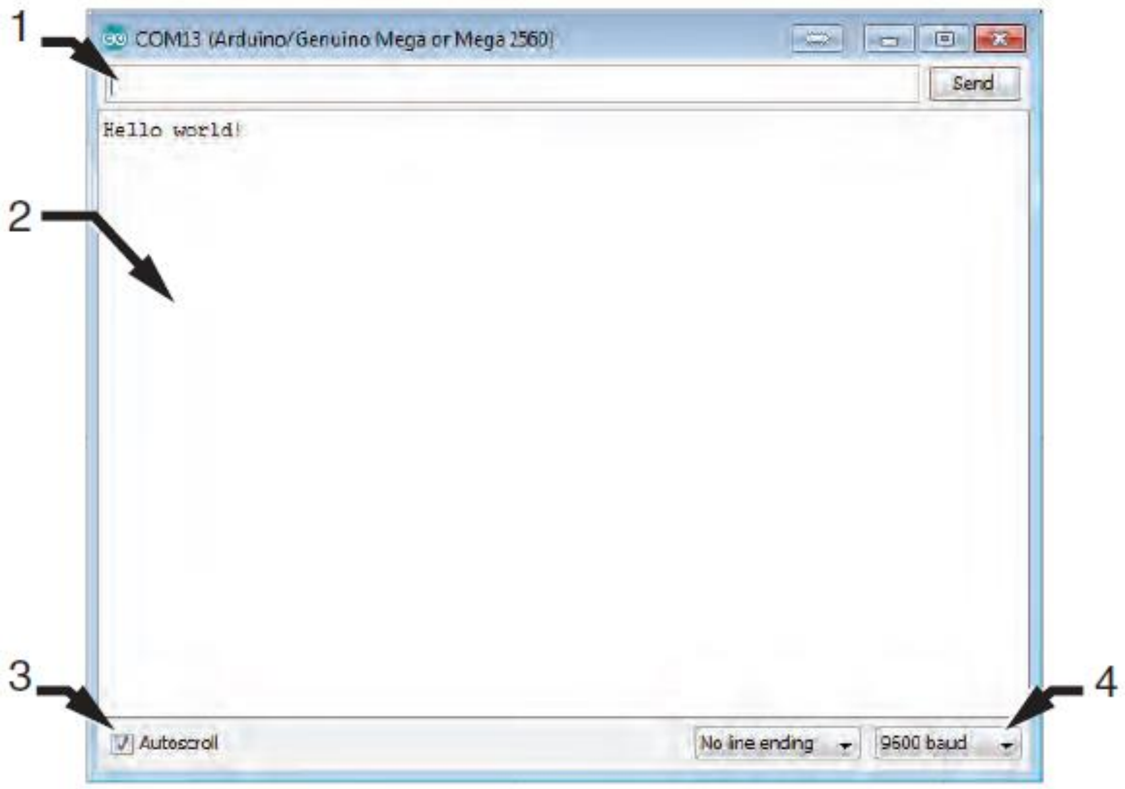
The goal of this example is to send "Hello world!" through serial communication to your computer and display it in the serial console.

```
void setup() {
Serial.begin(9600);                 // Open port
Serial.println("Hello world!");     // Write
}

void loop() {
}
```

Upload this to your board, open the *Serial Monitor* of the Arduino IDE

Serial monitor



```
Hello world!
```

Figure: The built-in serial monitor of the Arduino IDE.

Some analog sensors

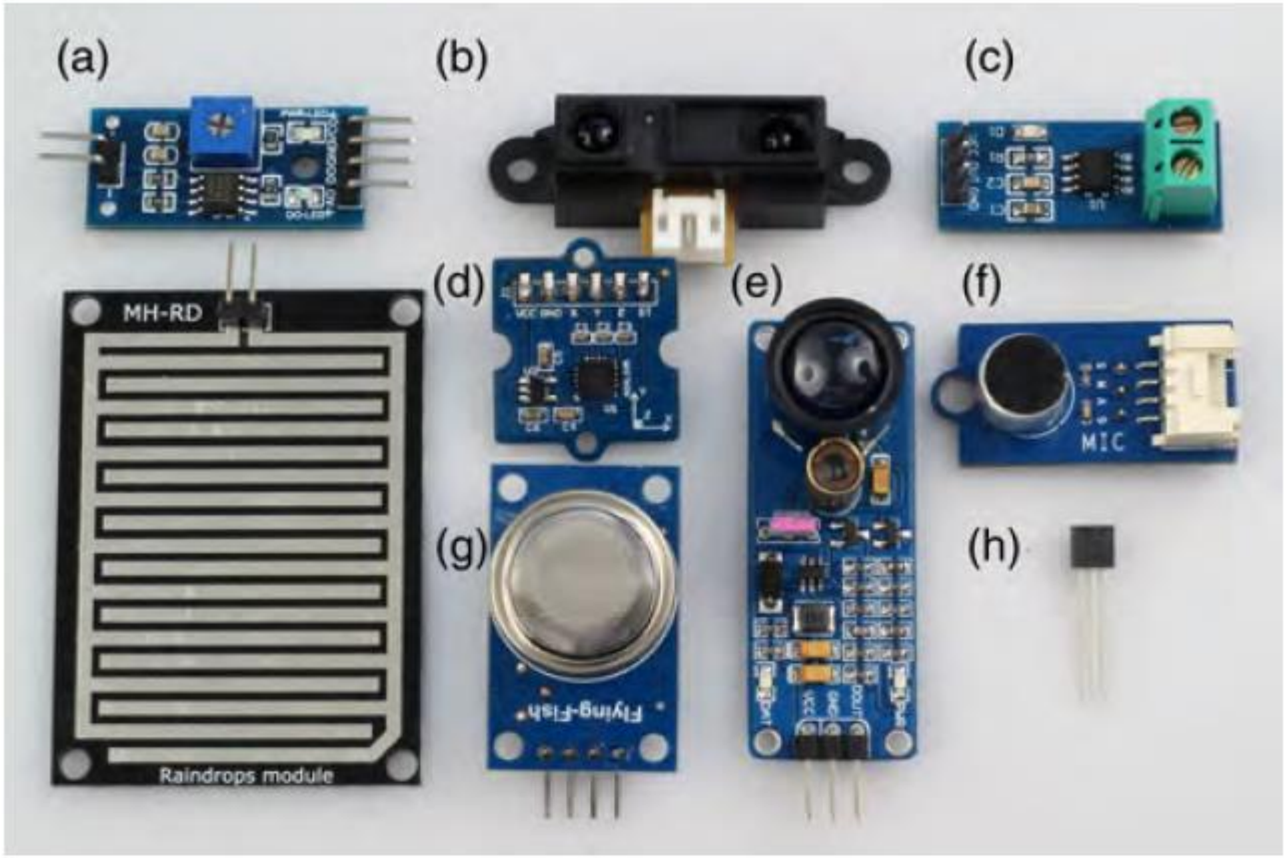


Figure: A variety of sensors with analog output, including sensors for: (a) rain; (b) infrared distance; (c) current; (d) acceleration; (e) laser distance; (f) sound; (g) gas; (h) temperature.

The role of the ADC

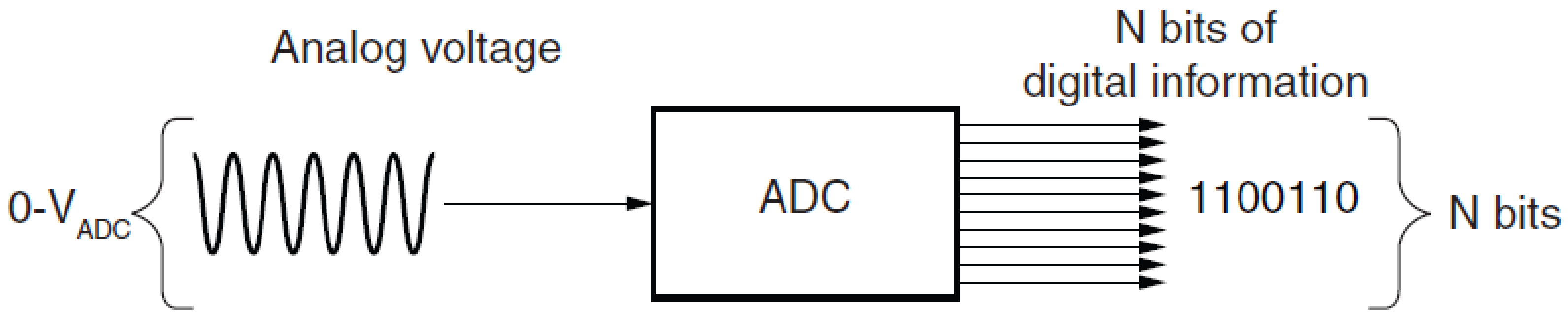


Figure: The ADC turns an analog voltage span to N bits of digital information.

Quantization example

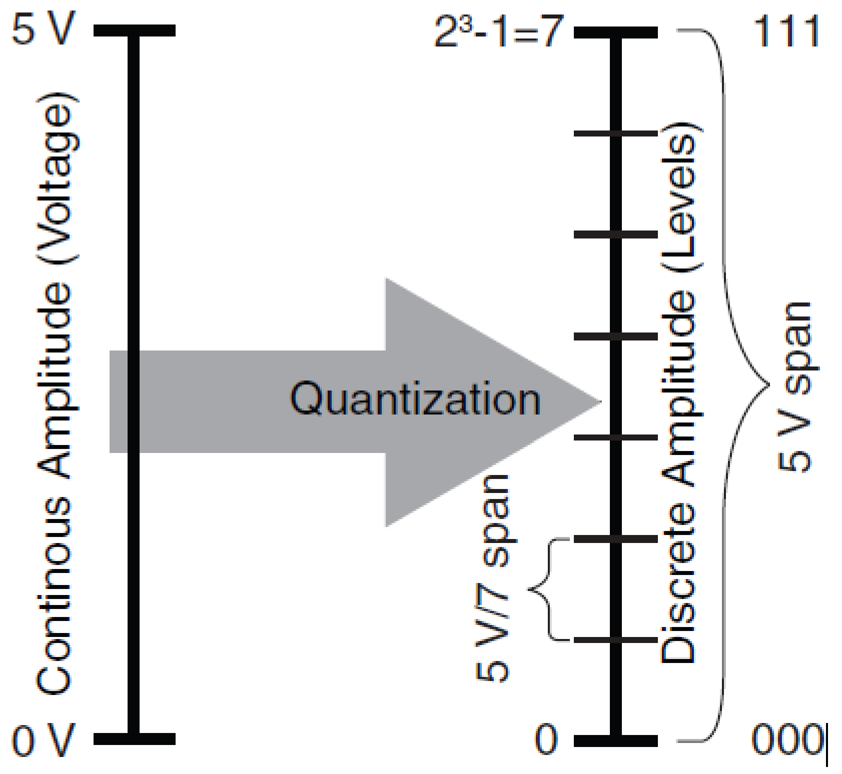


Figure: Quantization with a bit depth of 3.

Uno ADC

```
val = analogRead(aPin);
```

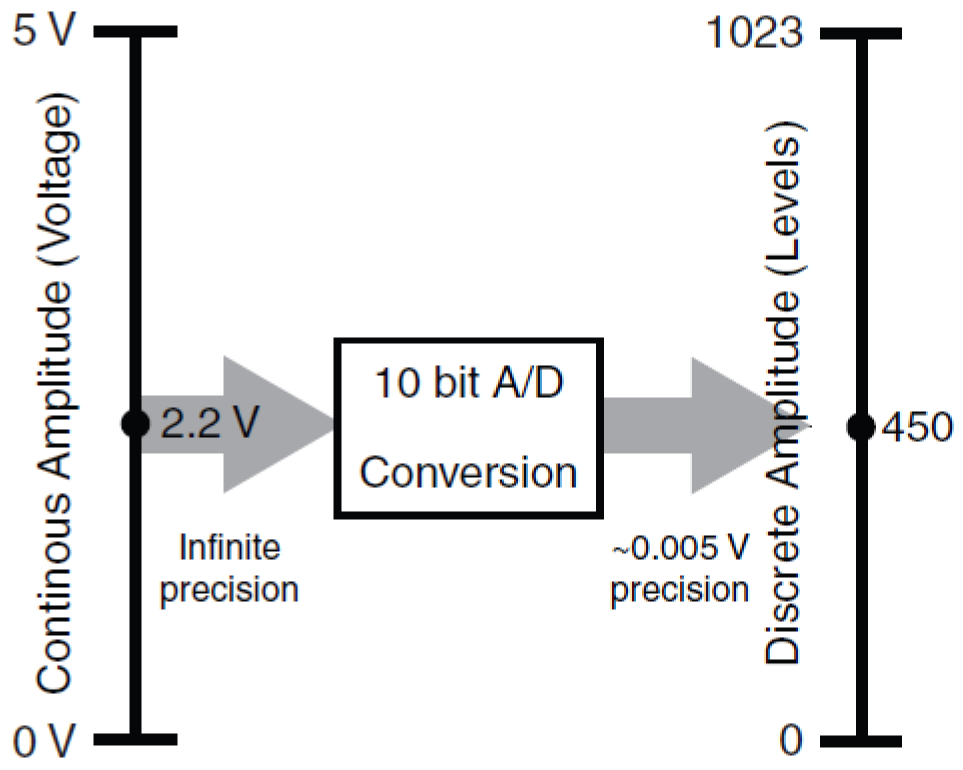
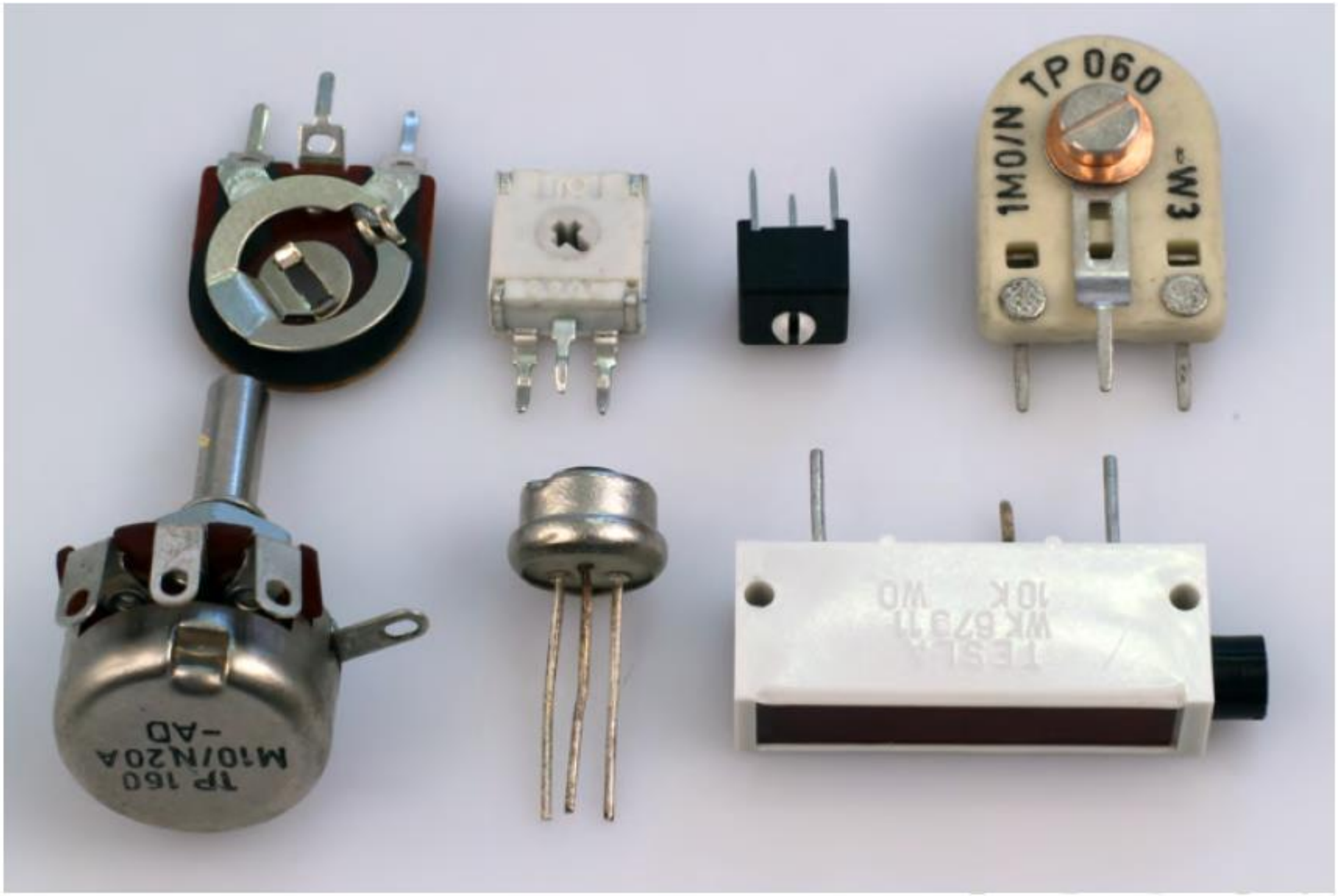


Figure: Sampling 5 V by the 10 bit ADC on the Arduino.

Potentiometers



Schematic view

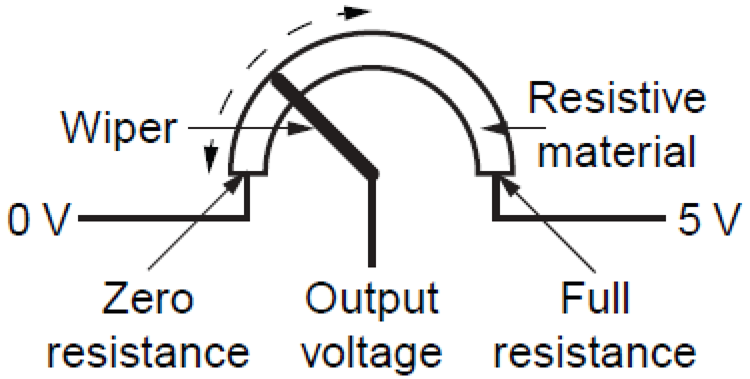


Figure: Schematic view of a potentiometer.

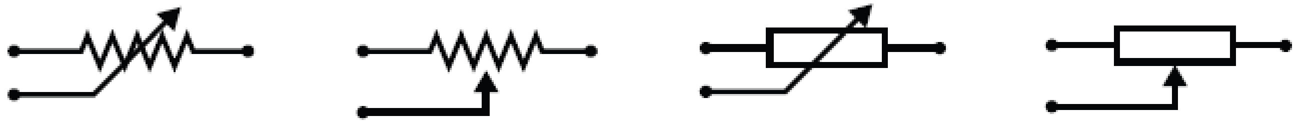


Figure: Electrical symbol of a potentiometer.

Connecting a pot to the Arduino

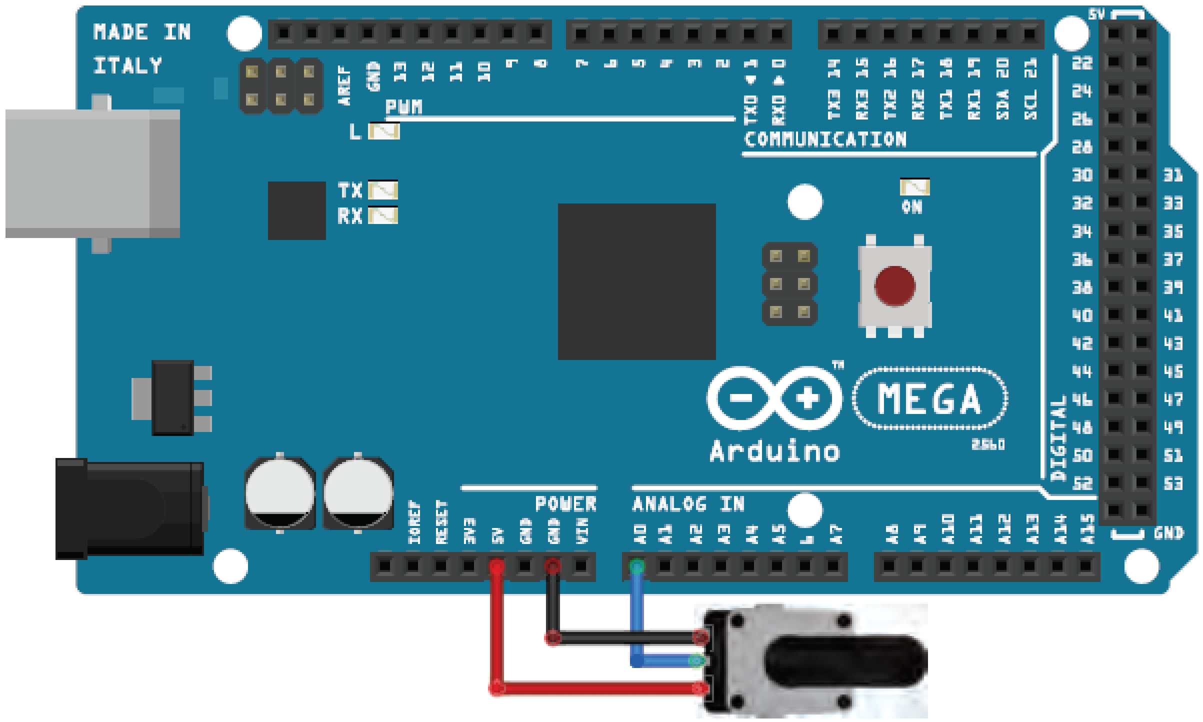


Figure: Connecting a potentiometer to the Arduino analog input ("A5" on the Flexy).

Read the ADC output and display it in the Serial Monitor

```
#define REF A5
void setup() {
  Serial.begin(9600);}
void loop() {
 int adc = analogRead(REF);
 Serial.println(adc);
 delay(500);}
```

Recompute ADC voltages and display it in the Serial Monitor

```
#define REF A5
void setup() {
  Serial.begin(9600);}
void loop() {
 int adc = analogRead(REF);
 Serial.println((float)adc*5.0/1024.0);
 delay(500);}
```

How would you connect something that only changes resistance to an Arduino? Create a voltage divider!

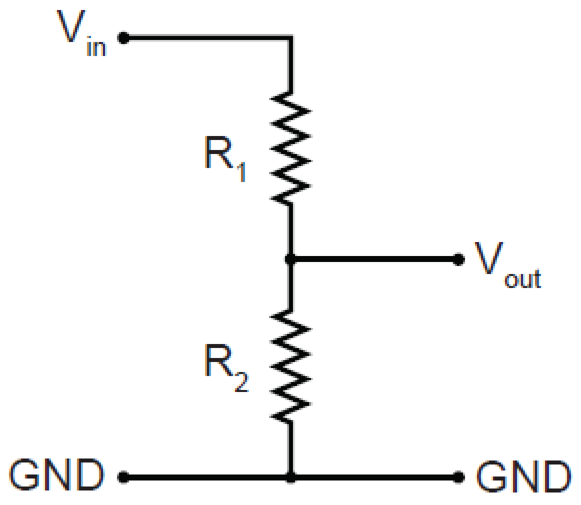
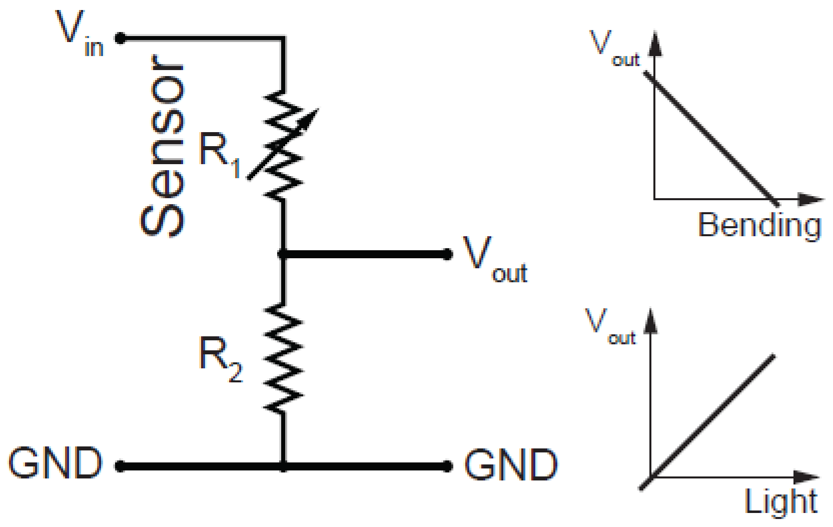


Figure: Voltage divider.

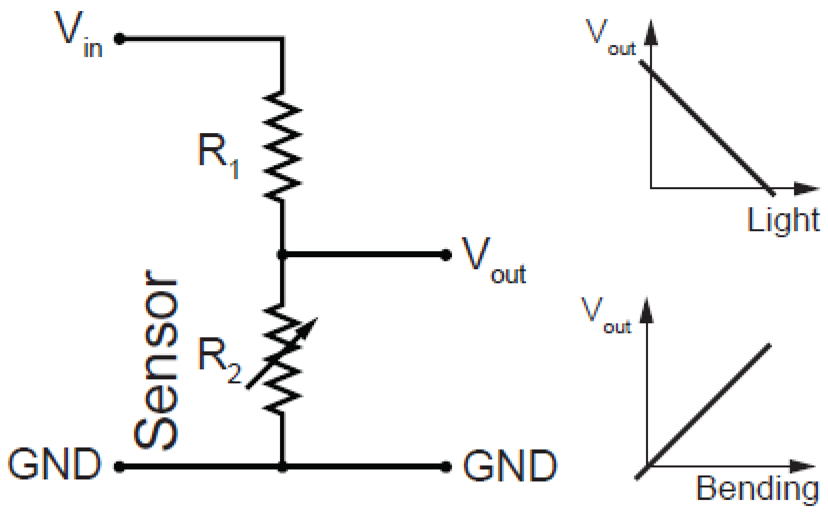
$$V_{out} = V_{in} \frac{R_2}{R_1 + R2}.$$

Using voltage dividers to connect resistive sensors

- Does not matter which is the sensor, but will change the direction of voltage variation
- Use roughly matching resistors



(a) Sensor as R_1.



(b) Sensor as R_2.

Figure: The relationship of output voltage depends on the placement and proportionality of the resistance to the measured phenomena.

Using voltage dividers to connect resistive sensors

- Impedance matching - use an operational amplifier
- Operational amplifier is in non-inverting voltage follower mode
- Can ignore the opamp in the model

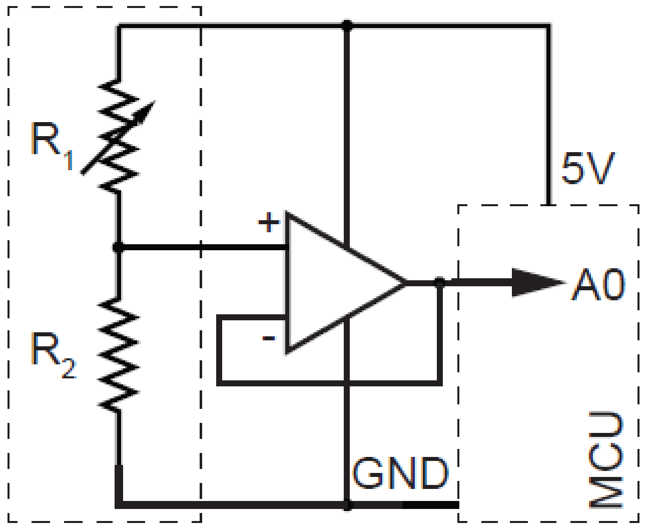


Figure: Connecting a resistive flexure sensor to the Arduino.

```
#define SENSOR A4
void setup() {
  Serial.begin(115200);}
void loop() {
 int adc = analogRead(SENSOR);
 int perc = map(adc,0,1023,0,100);
 Serial.println(perc);
 delay(100);}
```

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

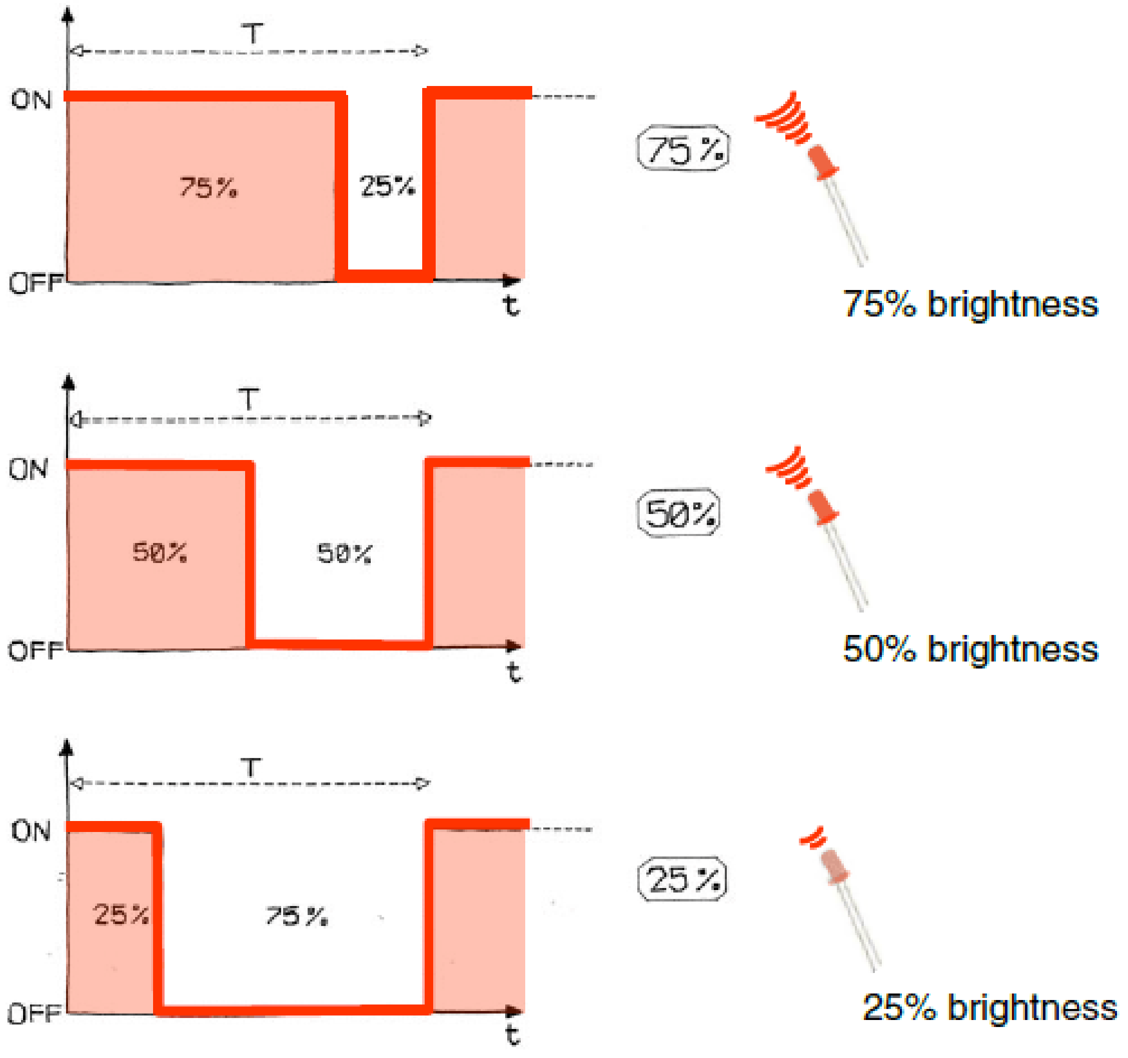


Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley

PWM Duty Cycle

output voltage = (on_time / cycle_time) * 5V

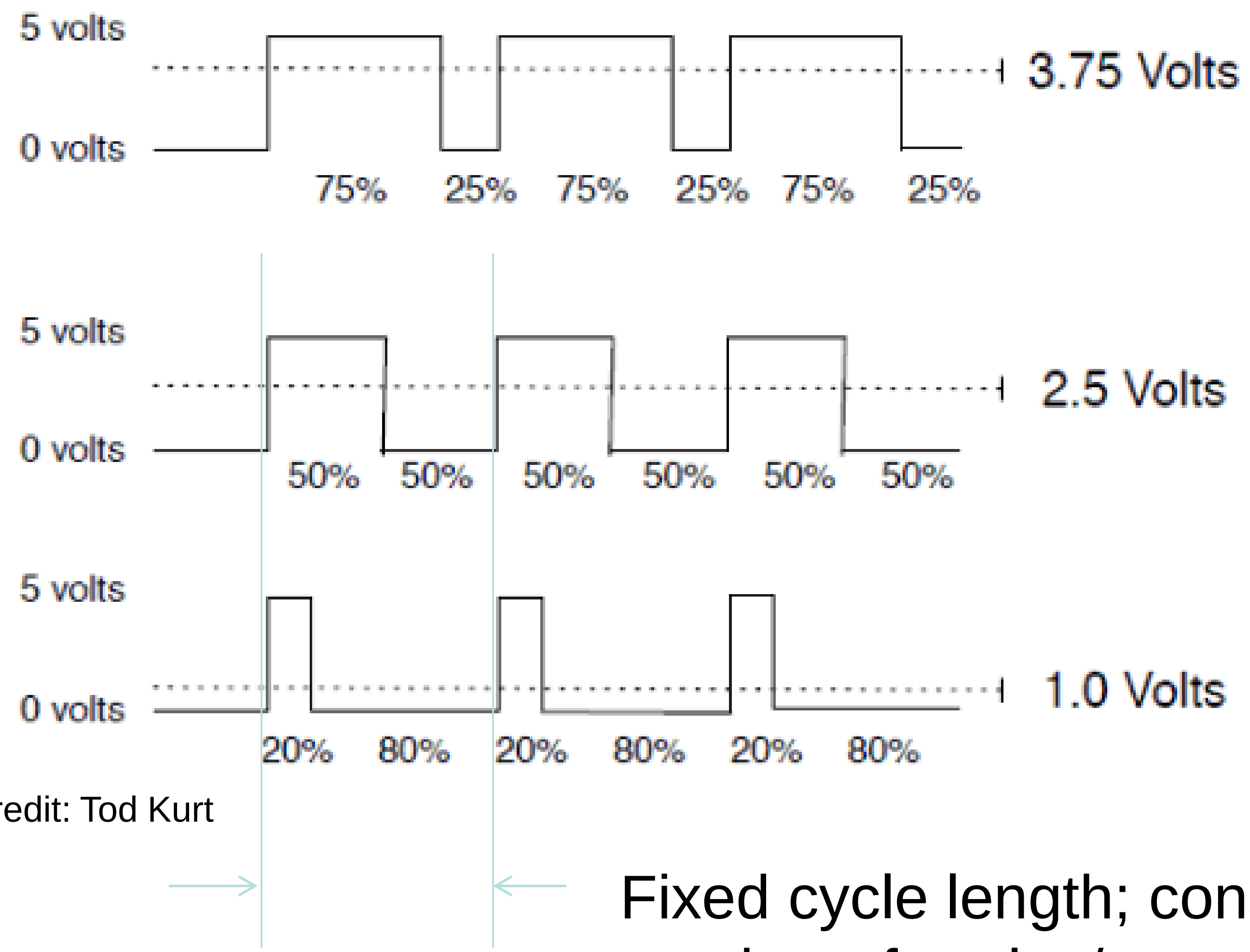
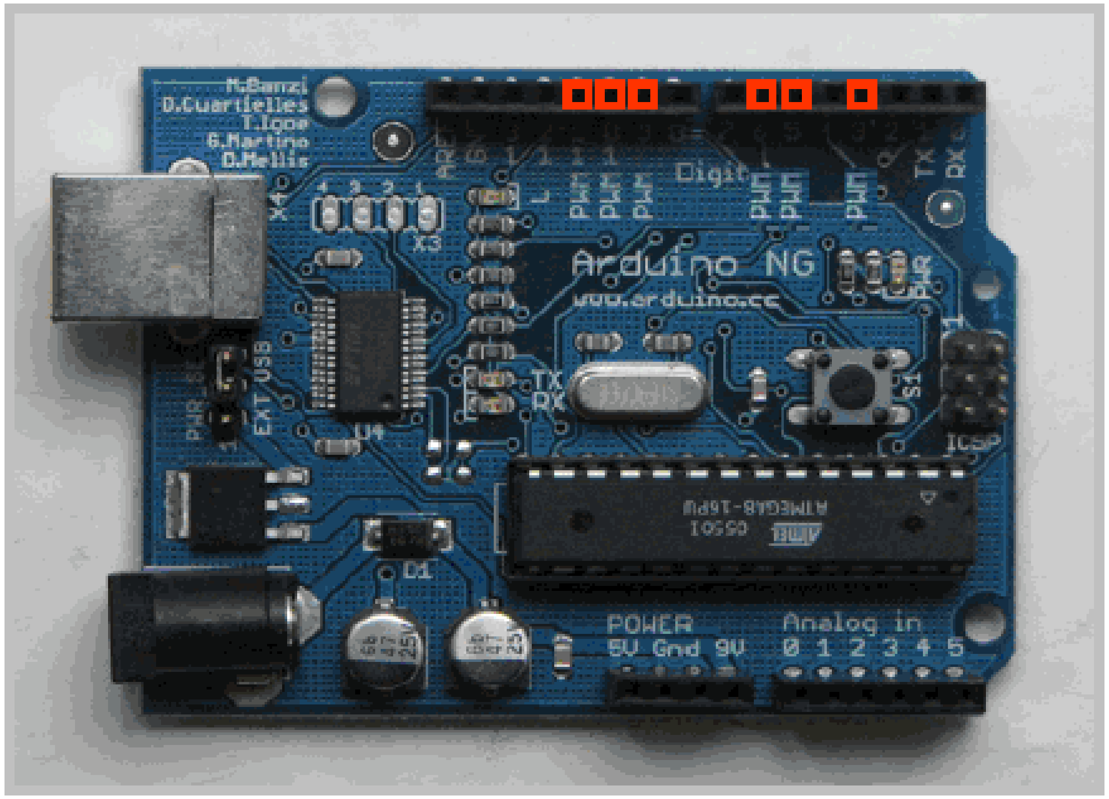


Image credit: Tod Kurt

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