Computer Organization and Assembly Language

Prepared by:
Nestor R. Valdez

Microcontroller

- Micro – small, Controller – one that is capable of control.
- A miniaturized computer that can sense and control more of the physical world than your average desktop.
- Control and intercepts data in forms of both input and output.
- Receives instructions via code, executes solutions in a step-by-step format.

What is Arduino

- Arduino is an open-source electronics platform based on easy-to-use hardware and software.
- Open-source physical computing platform based on a simple microcontroller board and a development environment for writing software for the board.
- Arduino comes with its own open-source Integrated Development Environment (IDE) used to create programs that the board will execute.
- Design to easily connect to your computer for easy and quick programming and interfacing.

What is Arduino

- An Arduino board is a one type of microcontroller based kit. The first Arduino technology was developed in the year 2005 by David Cuartielles and Massimo Banzi.
- The designers thought to provide easy and low cost board for students, hobbyists and professionals to build devices.
- Arduino was born at the **Ivrea Interaction Design Institute** as an easy tool for fast prototyping, aimed at students without a background in electronics and programming.

Why Arduino?

- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.

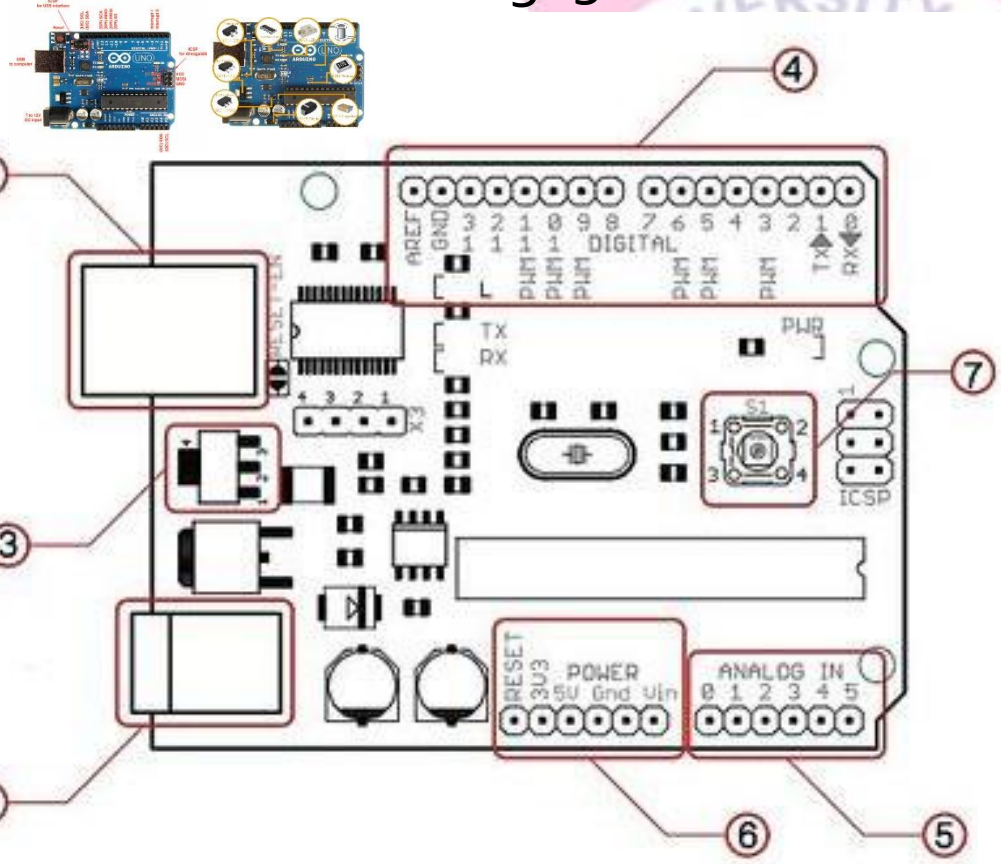
Why Arduino?

- **Open source and extensible software -** The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

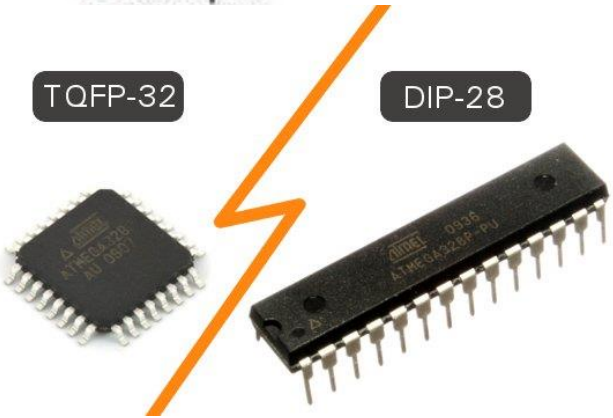
Parts of Arduino UNO

- A typical example of the Arduino board is Arduino UNO. It includes an Atmega328 microcontroller and it has 28-pins



The most important parts on the Arduino board high lighted in red:

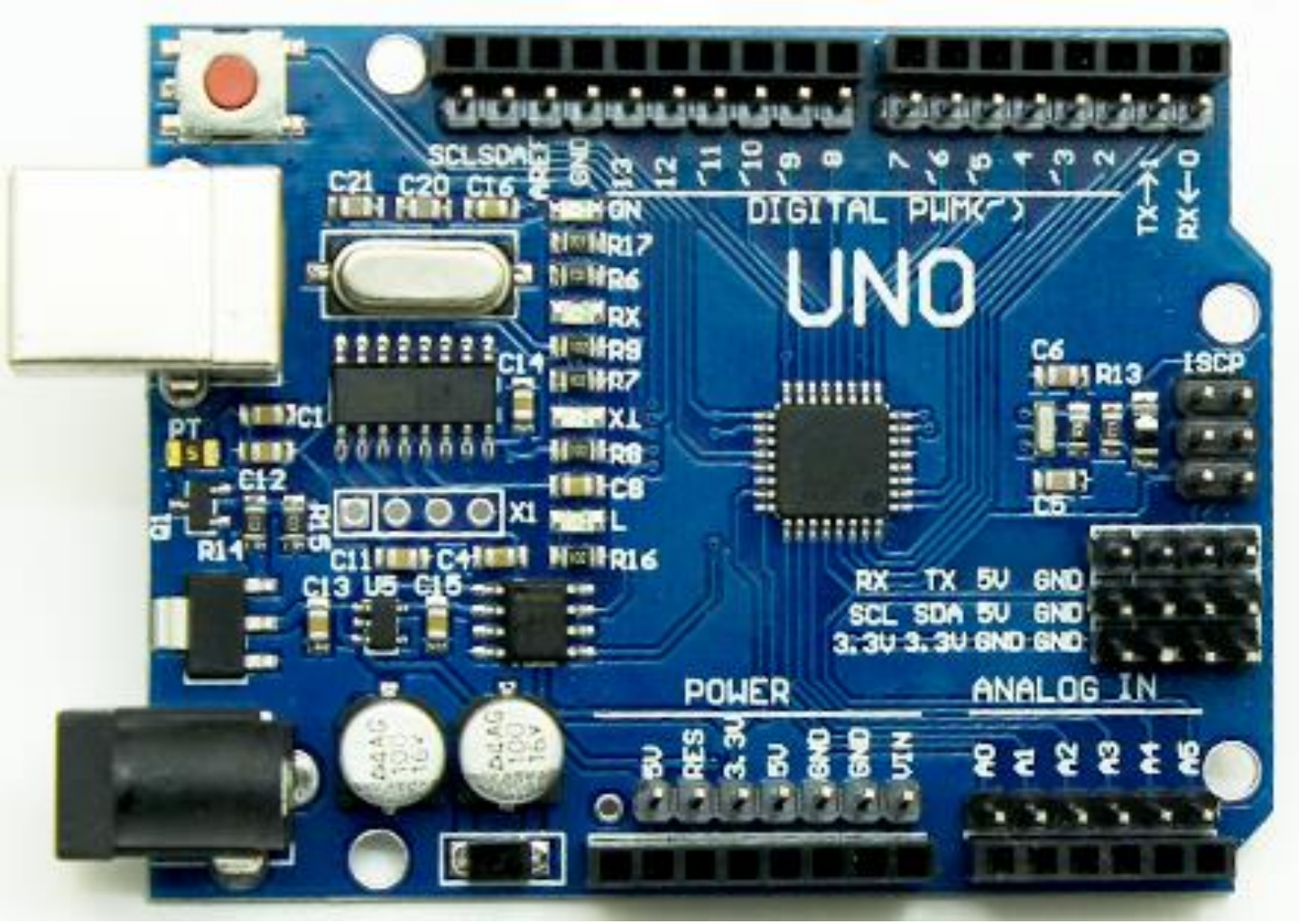
1: USB connector
2: Power connector
3: Automatic power switch
4: Digital pins
5: Analog pins
6: Power pins



Arduino UNO Specifications

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6 (A0-A5)
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB of which 0.5 KB used by bootloader
- SRAM: 2KB
- EEPROM: 1KB
- Clock Speed: 16 MHz

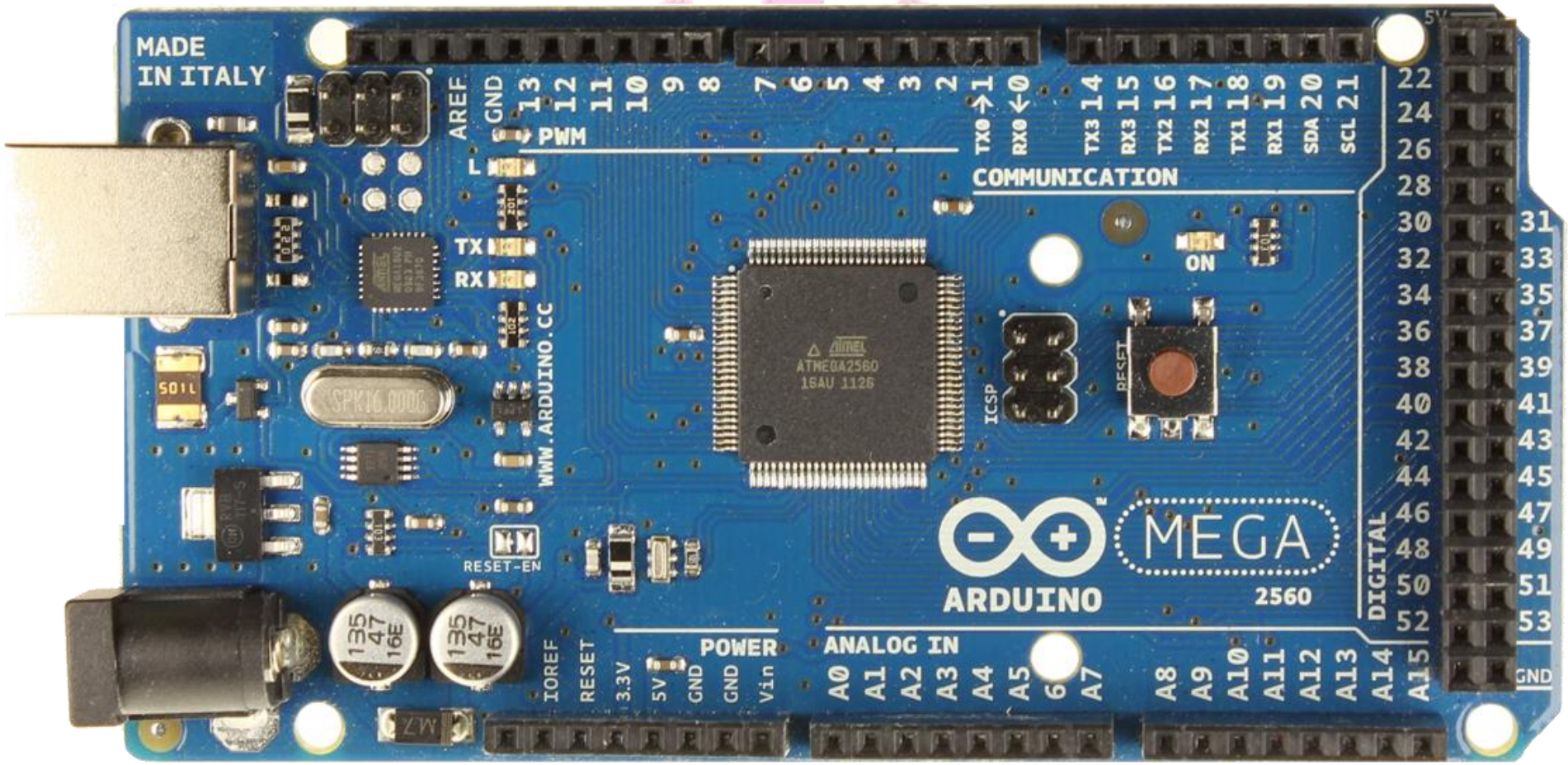
Arduino UNO


UNO
DIGITAL PWM(~)
POWER
ANALOG IN
ICSP
RX TX 5V GND
SCL SDA 5V GND
3.3V 3.3V GND GND

Arduino MEGA Specifications

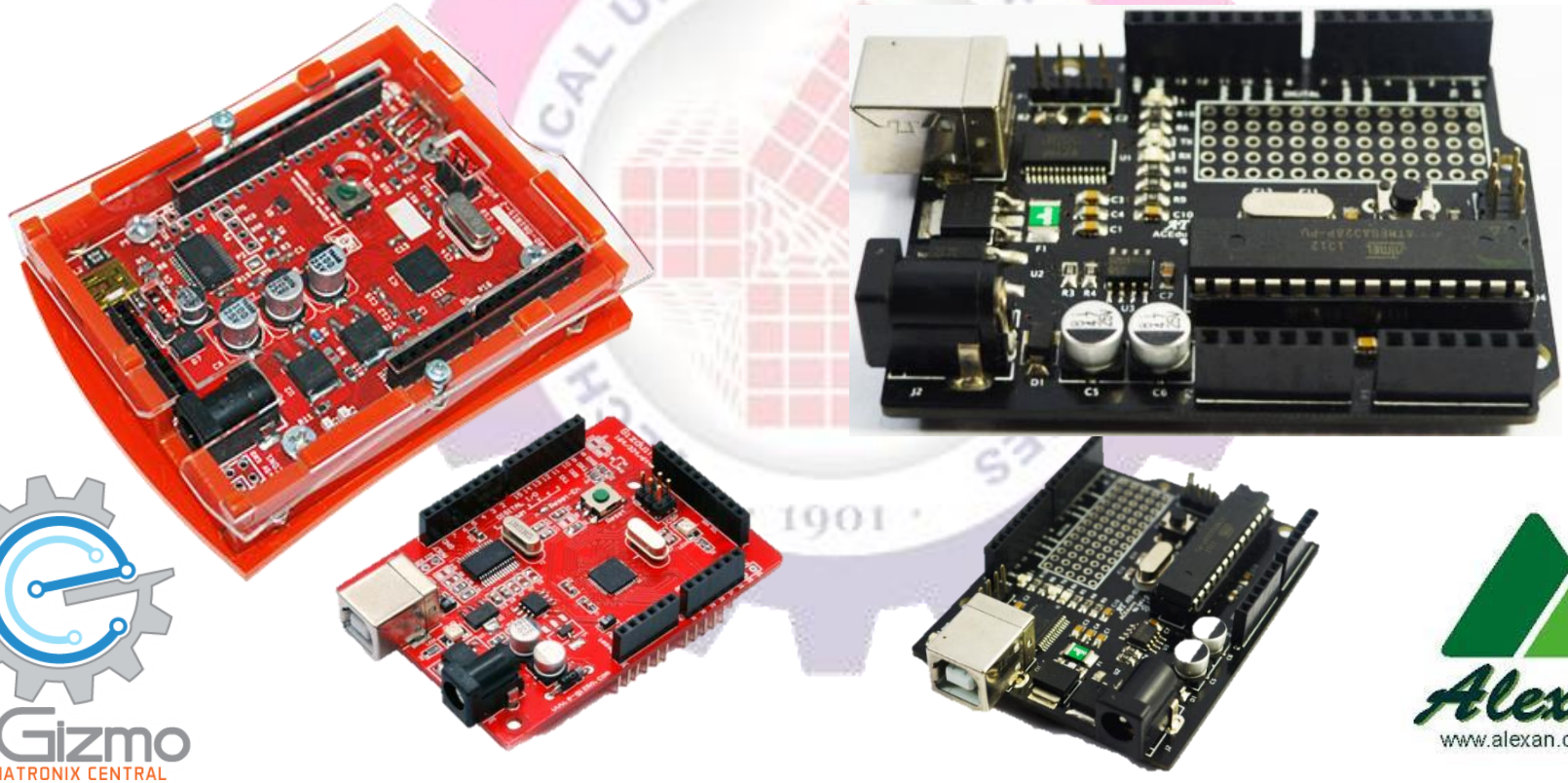
- Microcontroller: ATmega2560
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 54 (of which 15 provide PWM output)
- Analog Input Pins: 16 (A0-A15)
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 256 KB of which 8 KB used by bootloader
- SRAM: 8KB
- EEPROM: 4 KB
- Clock Speed: 16 MHz

Arduino MEGA



Arduino compatible boards

- Gizduino – by e-Gizmo
- Aceduino – by Alexan



e-Gizmo
MECHATRONIX CENTRAL

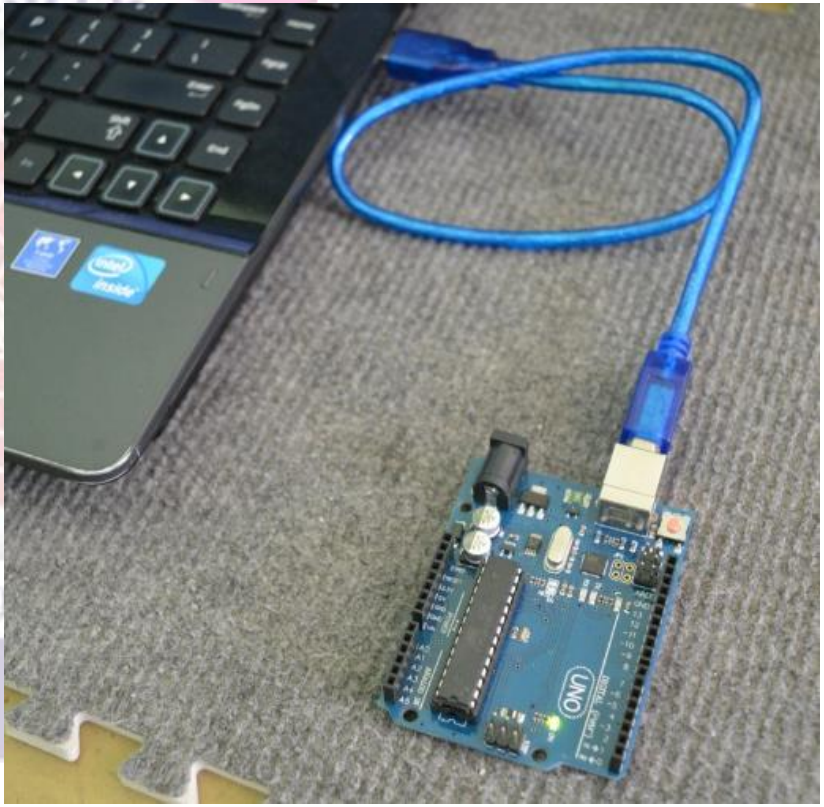
Alexan
www.alexan.com.ph

Arduino Shields

Arduino Shields

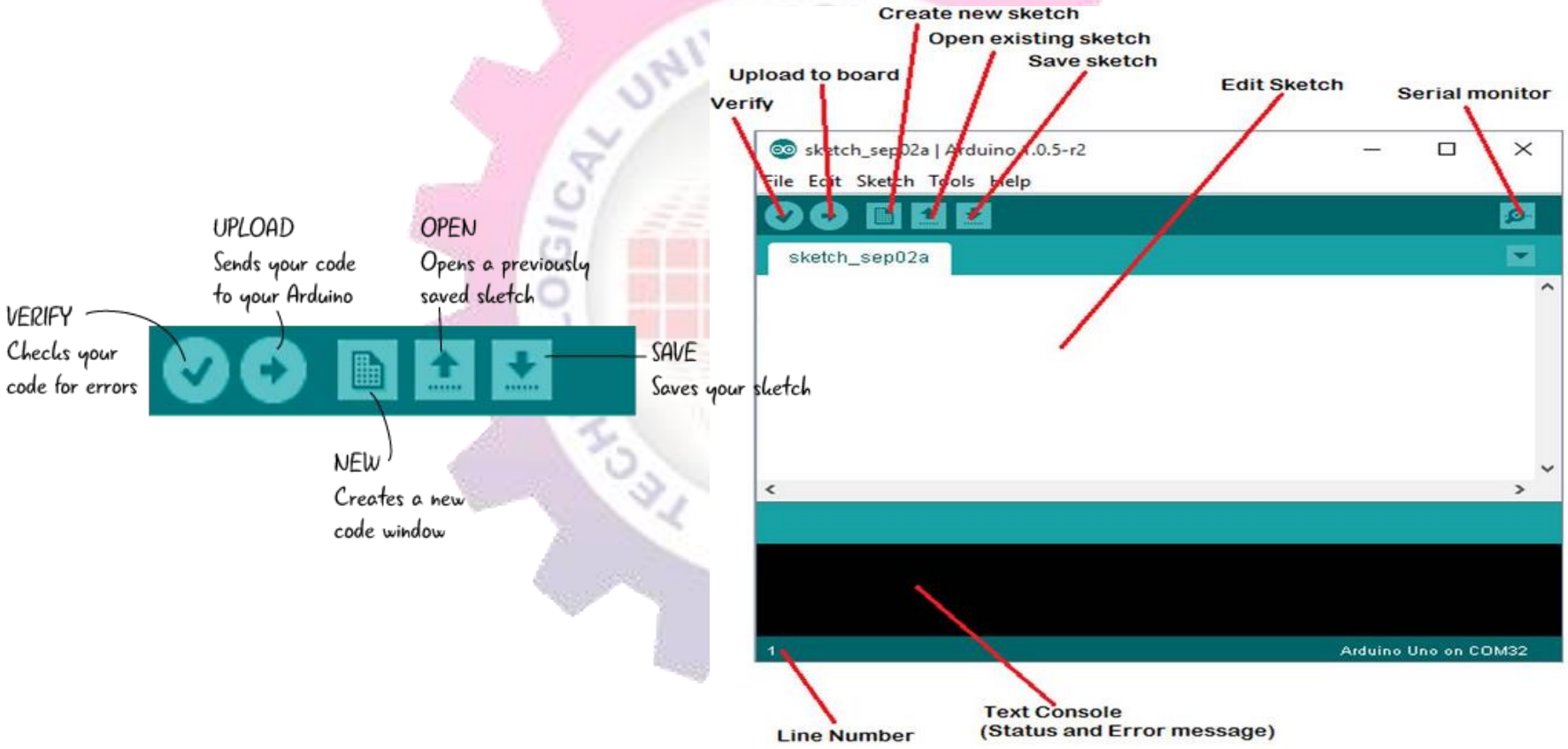
Connecting Arduino to PC

- Connect the Arduino board to your computer using the USB cable.
- Type A to PC
- Type B to Arduino
- LED power indicator should lit.



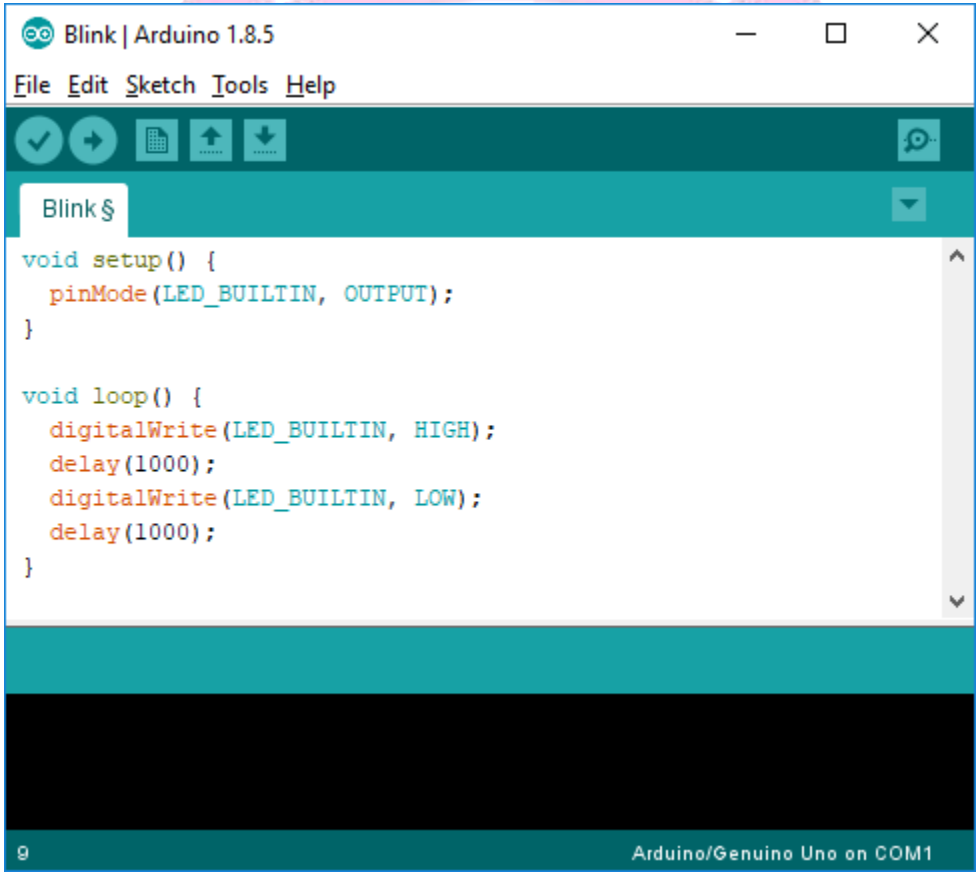
Launching Arduino IDE

- Double click the Arduino application and wait for it to be loaded.

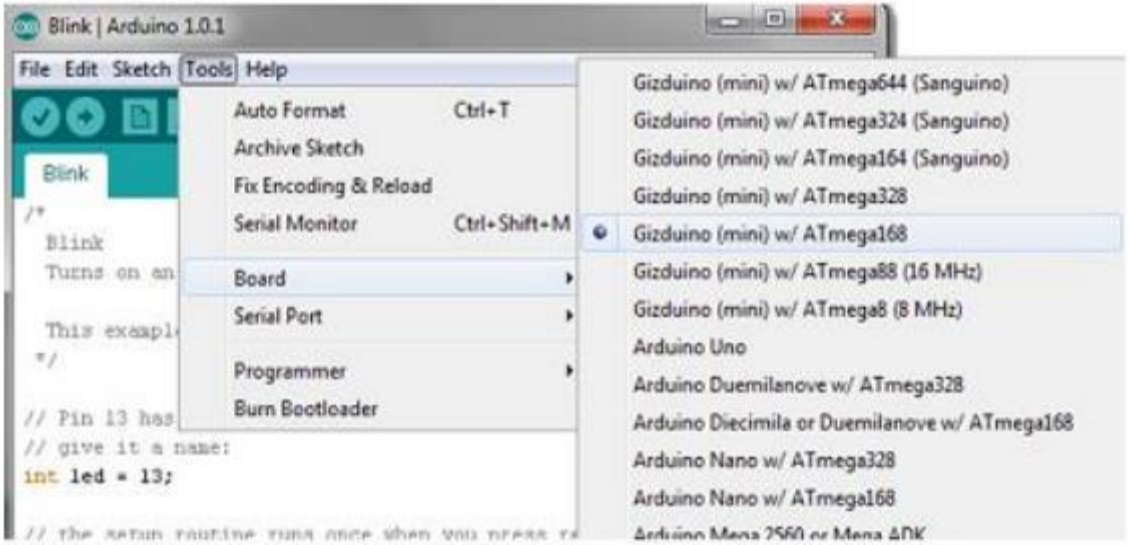


What is a sketch?

- It is the unit of code that is uploaded to and run on an Arduino board.
  - Ex.

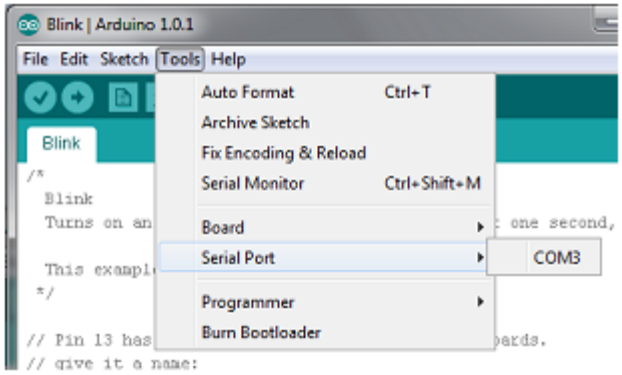


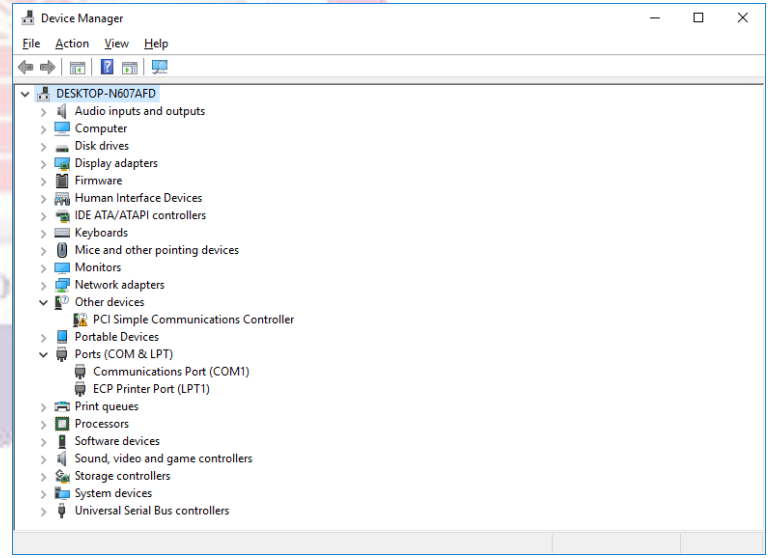
Setting-up Arduino IDE



Setting-up Arduino IDE

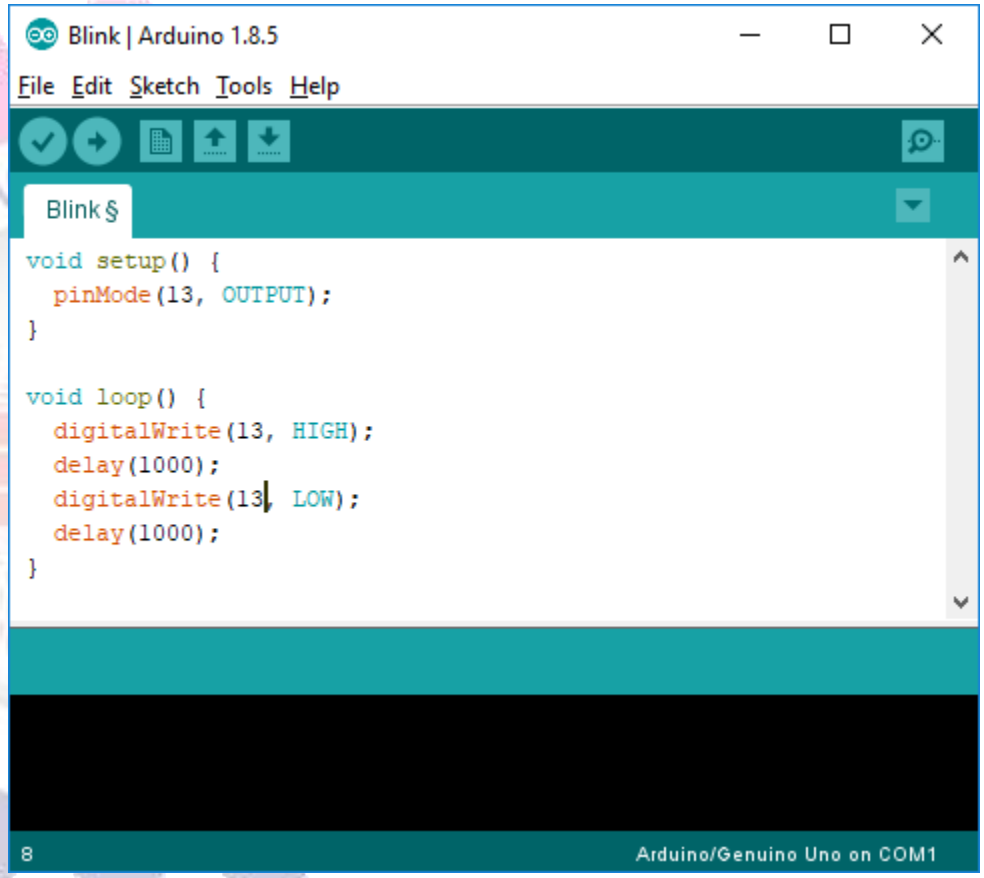
- Select the serial device of the Arduino board from the Tools > Serial Port menu
- Disconnect-reconnect the USB of the Arduino to find out which port to pick, or use the Device Manager





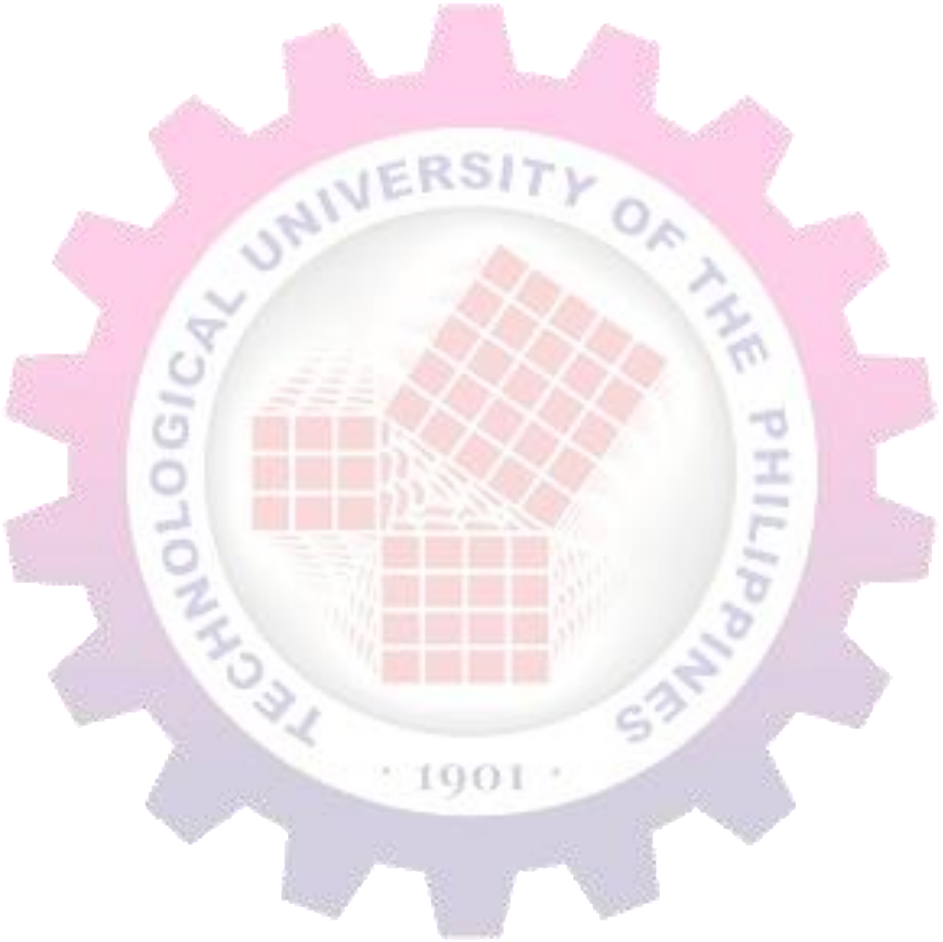
Blinks Sketch

```
int pin;
void setup() {
  pinMode(pin, OUTPUT);
}

void loop() {
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(1000);
}
```



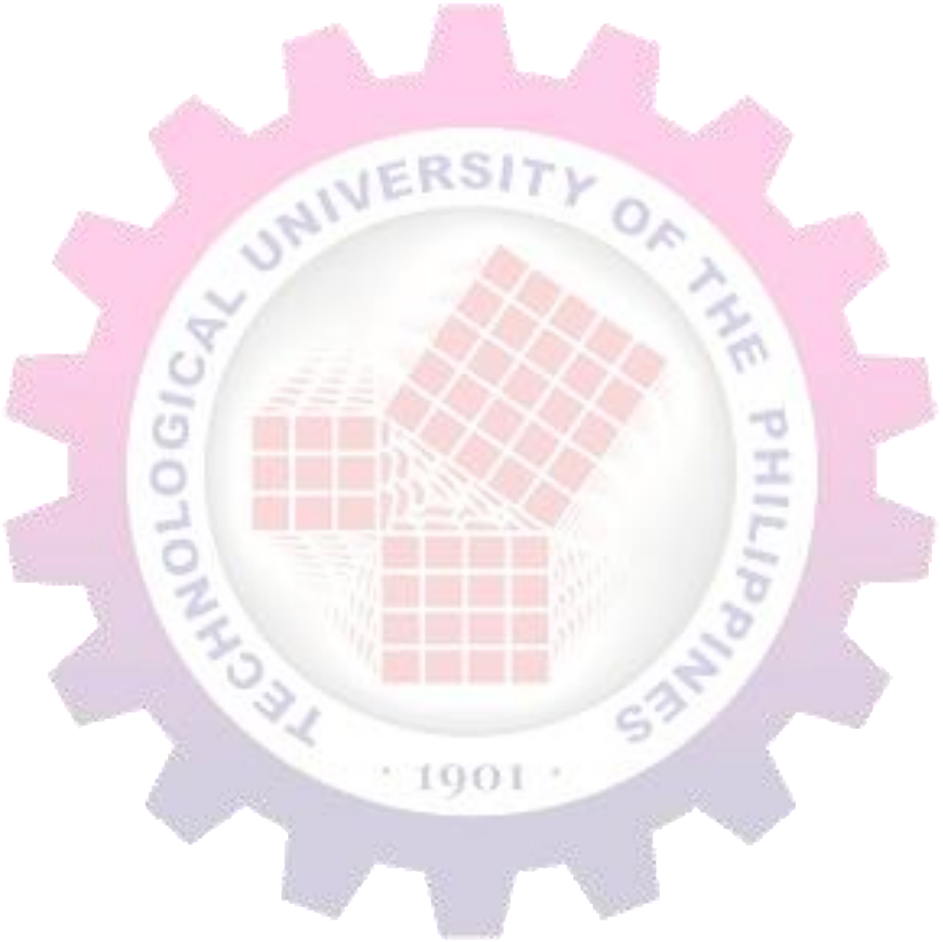
Uploading the Program

- Demo



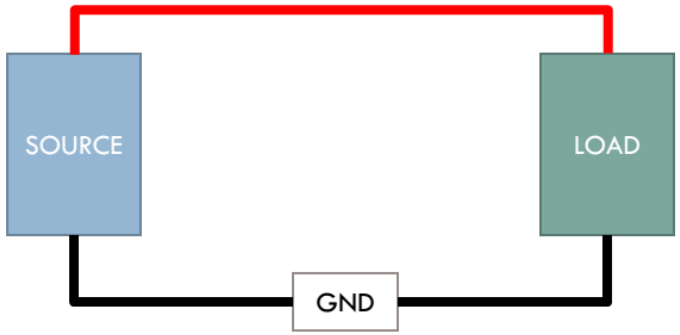
Program Execution

- Demo



Electronic Fundamentals

- CURRENT – measure of flow of electric charges
- VOLTAGE – measure of work done in making the electric charges flow
- RESISTANCE – property of an object to oppose the flow of electric charges
- LOAD – any device or component that will use electrical energy to operate.
- OHM'S LAW
  - Voltage = Current x Resistance
- POWER EQUATION
  - Power = Voltage x Current
- GND / GROUND – electronic circuit termination.



Programming Arduino

Syntax and Symbols

Syntax and Symbols

- Grouping symbols
  - ( )
  - { }
  - “ “
- Arithmetic symbols
  - +
  - -
  - *
  - /
  - %
- Punctuation symbols
  - ;
  - ,
  - .
- Comparators
  - =
  - <
  - >
  - &&
  - ||

Comments

- Statements ignored by the Arduino when it runs the sketch.
- Used to give information about the sketch, and for people reading the code.
- Multi-line comment: /* <statement> */
- Single-line comment: //

Variables

- Place for storing a piece of data
- Has a name, value, and type
  - int, char, float
- Variable names may not start with a numerical character
- Example:
  - int iamVariable = 9;
  - char storeChars;
  - float saveDecimal;

Variable Types

Basic Data Types

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
short	2 bytes	-32,768 to 32,767
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
float	4 bytes	1.2E-38 to 3.4E+38 (6 decimal places)
double	8 bytes	2.3E-308 to 1.7E+308 (15 decimal places)
Long double	10 bytes	3.4E-4932 to 1.1E+4932 (19 decimal places)

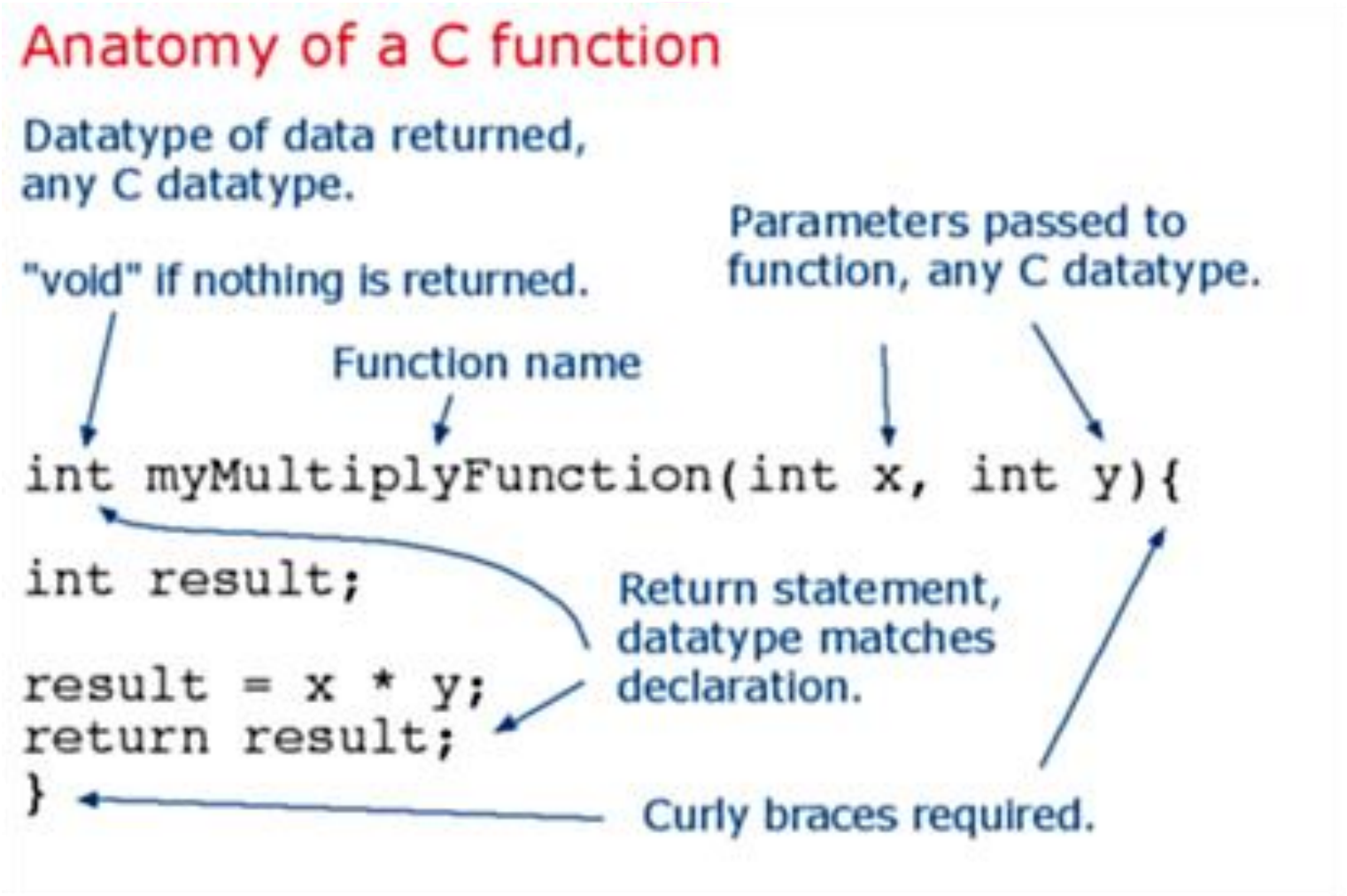
Variables

- You may now refer to this variable by its name, at which point its value will be looked up and used
- Example:
  - In a statement: digitalWrite(13, HIGH);
  - Declaring: int led = 13;
  - We can instead write: digitalWrite(led, HIGH);

Functions

- Modular pieces of code that perform a defined task outside of the main program
- Very useful to reduce repetitive lines of code due to multiple execution of commands
- Has a name, type, and value… same as variables?!

Functions


Anatomy of a C function
Datatype of data returned, any C datatype.
"void" if nothing is returned.
Function name
Parameters passed to function, any C datatype.
int myMultiplyFunction(int x, int y){
int result;
result = x * y;
return result;
}
Return statement, datatype matches declaration.
Curly braces required.

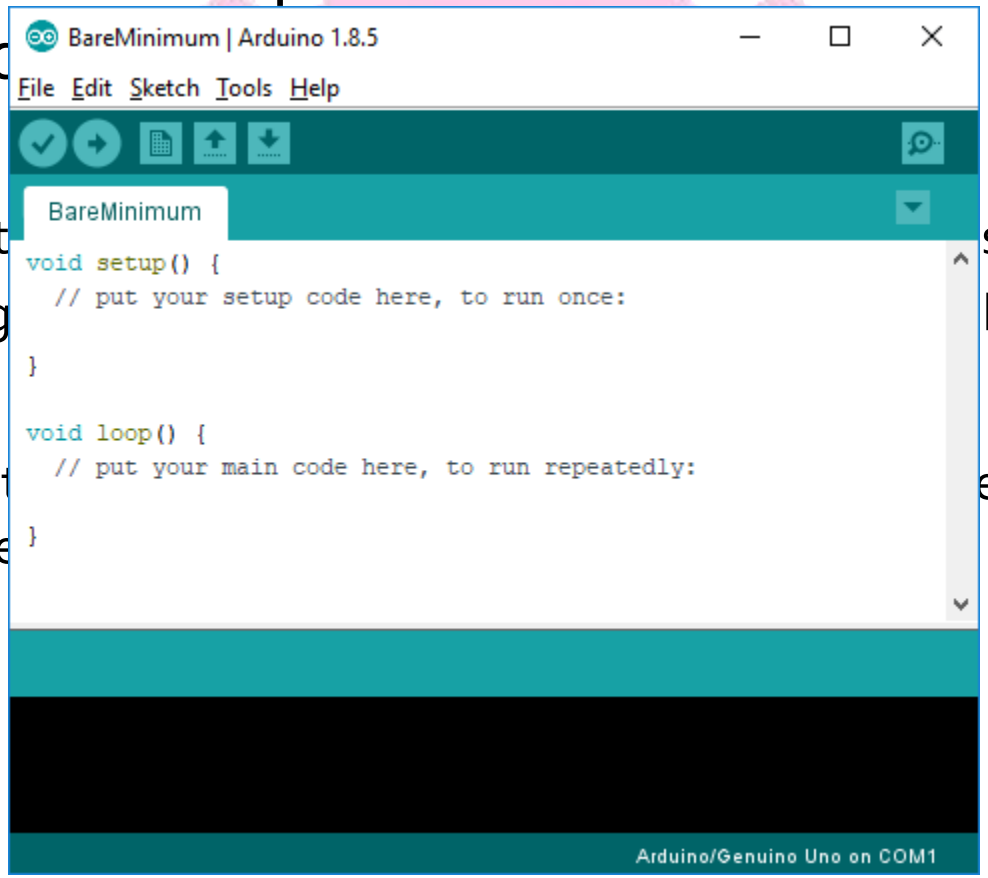
Bare Minimum

- There are two special functions that are part of every Arc
  - setup()
    - A funct
    - Setting
  - loop()
    - A funct

s

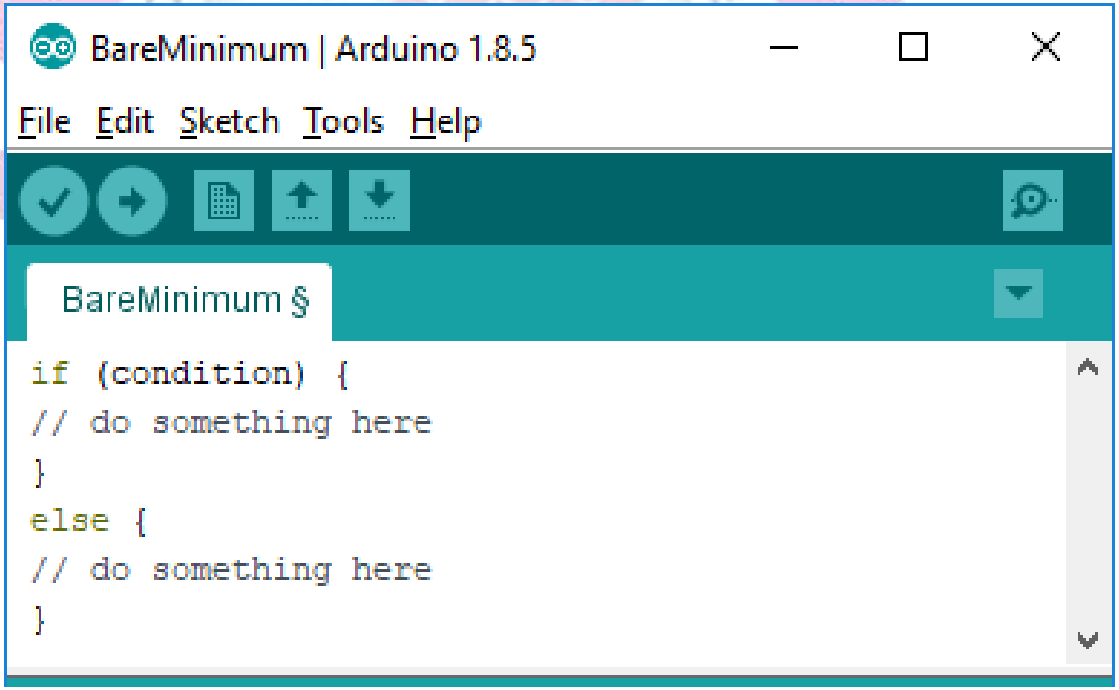
here

e heart of most sketche



Control Structures

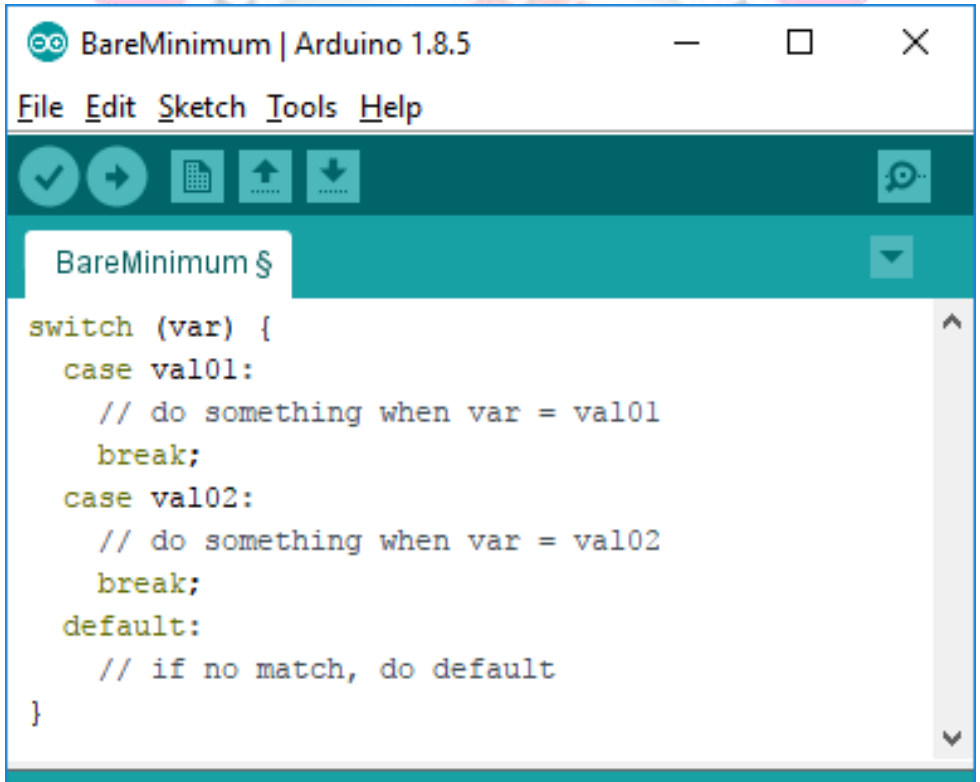
- If-Else Statement
  - It allows you to make something happen or not depending on whether a given condition is true or not.
  - Syntax:



Control Structures

Switch-Case Statement

- Depends on whether the value of a specified variable matches the values specified in case statements.
- Syntax:



Control Structures

Switch-Case Sta

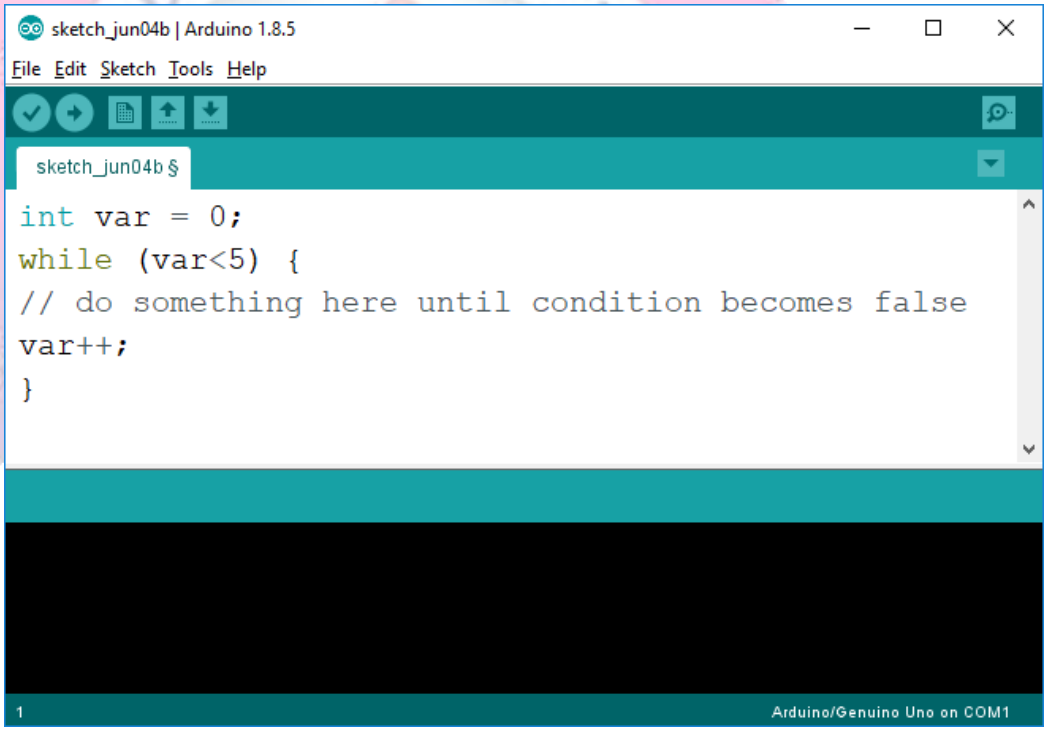
- Depends on v f a specified variable matches the va e statements.
- Syntax:

```
switch
  case
    //
    bre
  case
    //
    bre
  defau
    //
}
```

```
#include <stdio.h>
int main()
{
    int i=2;
    switch (i)
    {
        case 1:
            printf("Case1 ");
            break;
        case 2:
            printf("Case2 ");
            break;
        case 3:
            printf("Case3 ");
            break;
        case 4:
            printf("Case4 ");
            break;
        default:
            printf("Default ");
    }
    return 0;
}
```

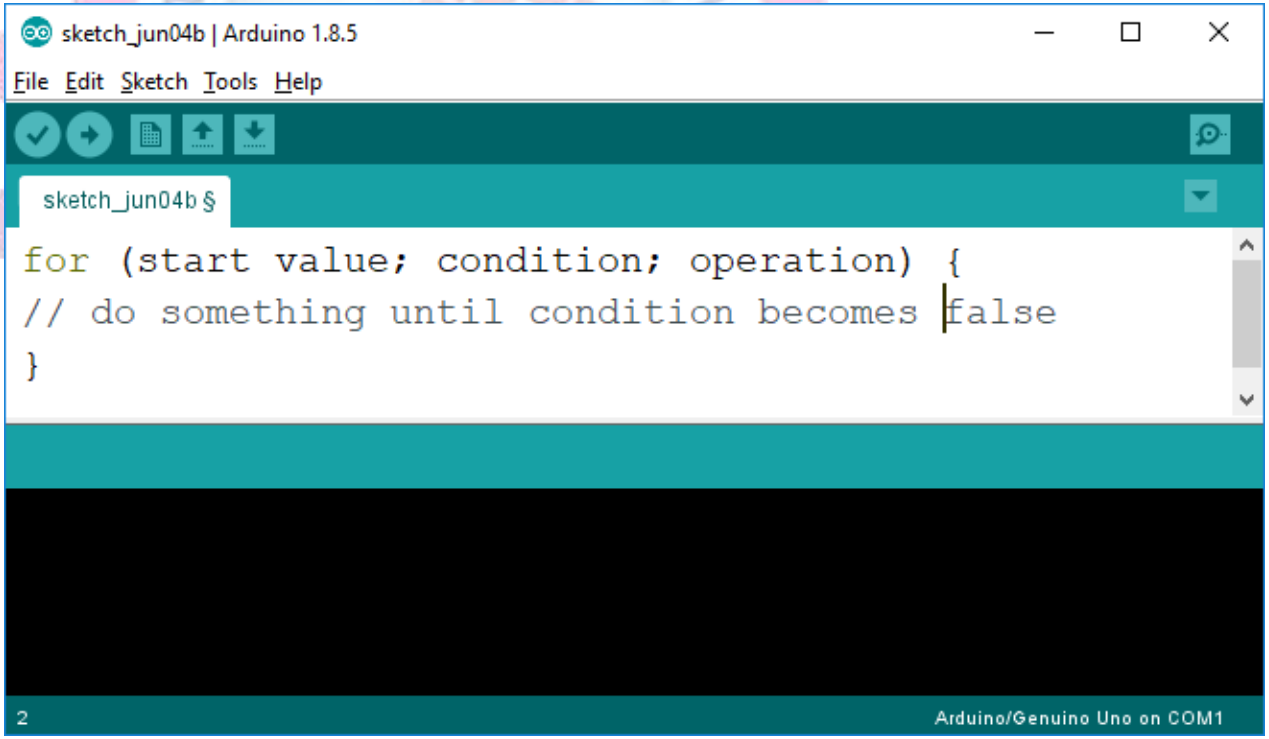
Control Structures

- While Loop
  - Continue program until a given condition is true
  - Syntax:



Control Structures

- For Loop
  - Distinguished by a increment/decrement counter for termination
  - Syntax:



Timing Controls

- delay()
  - Halts the sequence of execution during the time specified
  - Syntax:
    - delay(milliseconds);
    - milliseconds = the time value in milliseconds

Digital Pins

- pinMode()
  - Configures the specified pin to behave either as an input or an output
  - Syntax:
    - pinMode(pin, mode);
  - pin = the number label of the pin whose mode you wish to set
  - mode = either INPUT or OUTPUT

Digital Pins

- 14 configurable digital pins – can be specified to produce an OUTPUT or expect an INPUT
- Pins are configured as INPUT by default
- Pins labeled 0-13 can be used, note however those with multiple use:
  - 0 and 1 – Serial RX and TX (permanent)
  - 2 and 3 – Serial RX and TX (assigned)
  - 3, 5, 6, 9, 10, 11 – PWM/Analog Output
  - 13 – Internal resistor and LED

- Low-impedance state, able to provide a maximum of 40mA of current
- Watch out for short circuit connection and/or high current draw!
- Good practice to always connect an external resistor before interfacing
  - No need for this on Pin 13
  - Remove if maximum current draw is required

Digital Output

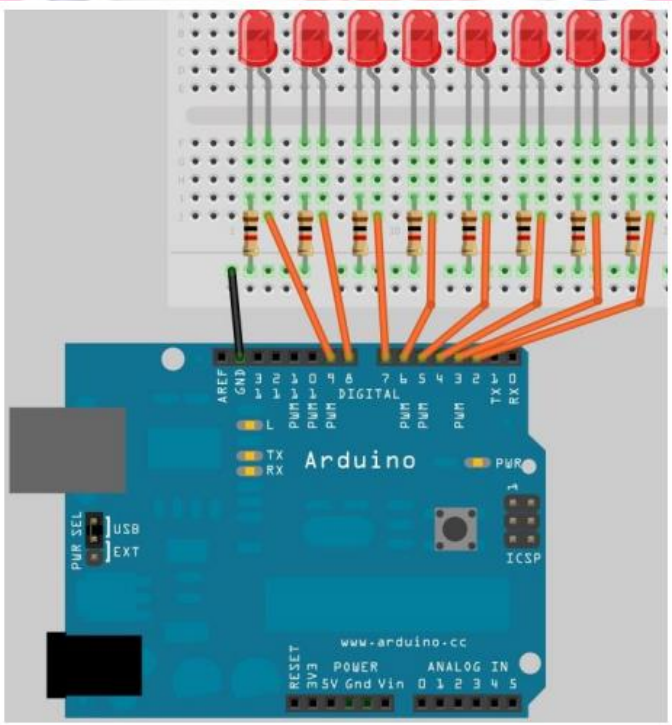
- digitalWrite()
  - Command an OUTPUT pin to produce either a HIGH or a LOW value
  - Syntax:
    - digitalWrite(pin, value);
  - pin = the number label of the pin to be used.
  - value = either HIGH or LOW.

Digital Values

- Logic '0'
  - LOW value
  - Represents GND or 0 volts
  - Equivalent to 'False'
- Logic '1'
  - HIGH value
  - Represents +5 volts
  - Equivalent to 'True'

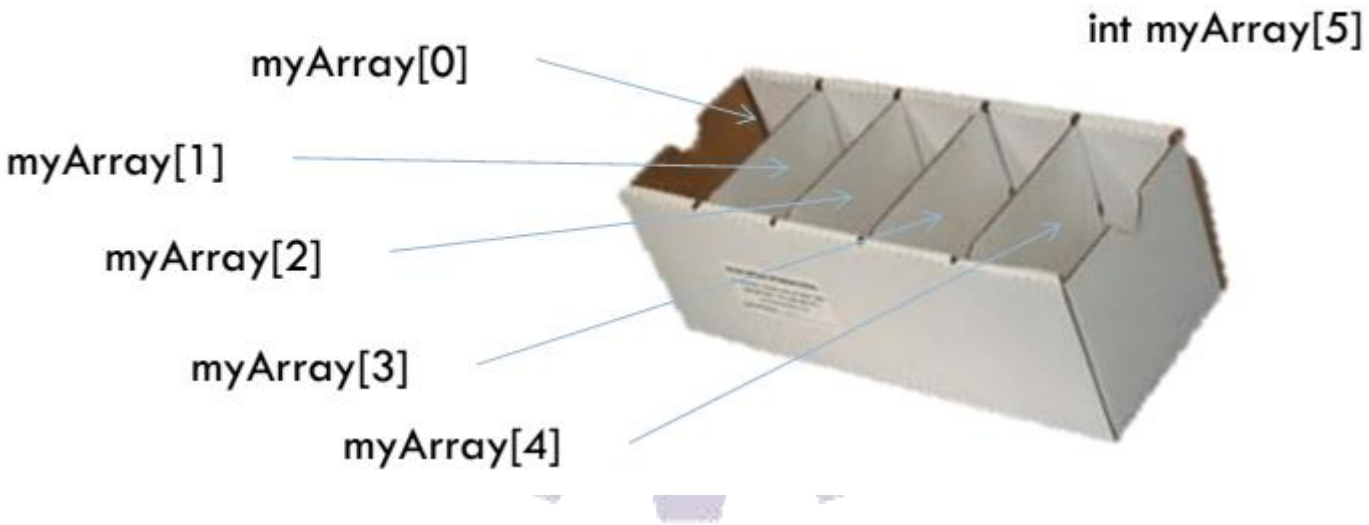
Digital Output

- Exercise:
  - Modify the Blink code to blink multiple LEDs on different pins at the same time



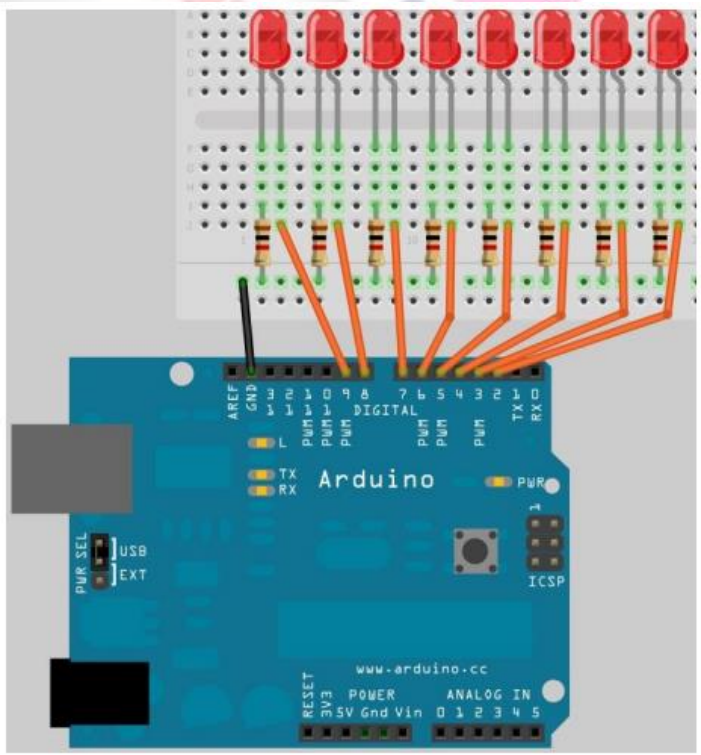
Arrays

- Array – a collection of variables of the same type accessed via a numerical index
- It is a series of variable instances placed adjacent in memory, and labeled sequentially with the index



Digital Output

- Exercise:
  - Further modifying the code and using our new technique, make the LEDs run! Light them one LED at a time from left to right then right to left on an endless loop.



Digital Input

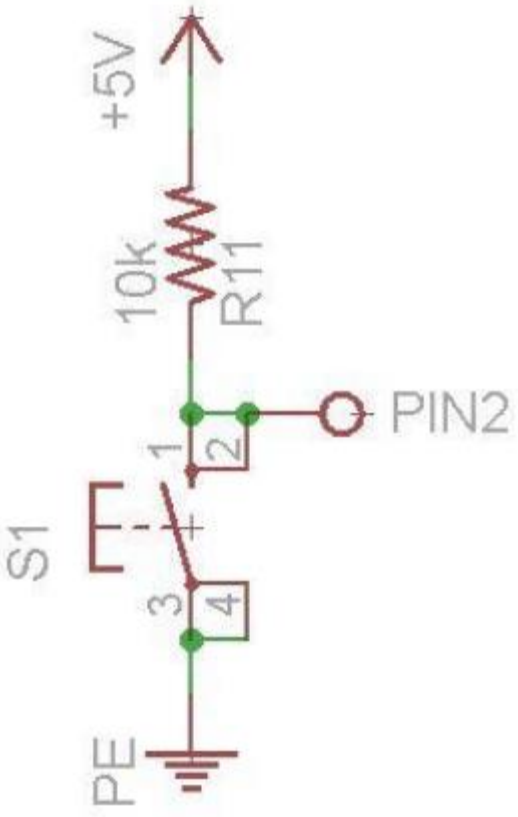
- High-impedance state, has no effect on external circuitry and needs only very little current to sense incoming stimuli.
- Able to sink up to 40mA of current. Avoid exposing pins to higher current values.
- Always configure with pinMode() despite having pins as INPUT by default to ensure proper operation.

Digital Input

- digitalRead()
  - Reads the value from a specified digital INPUT pin, returns either HIGH or LOW
  - Syntax:
    - variable = digitalRead(pin);
  - pin = the number label of the pin to be used
  - variable = variable assigned to hold the read value

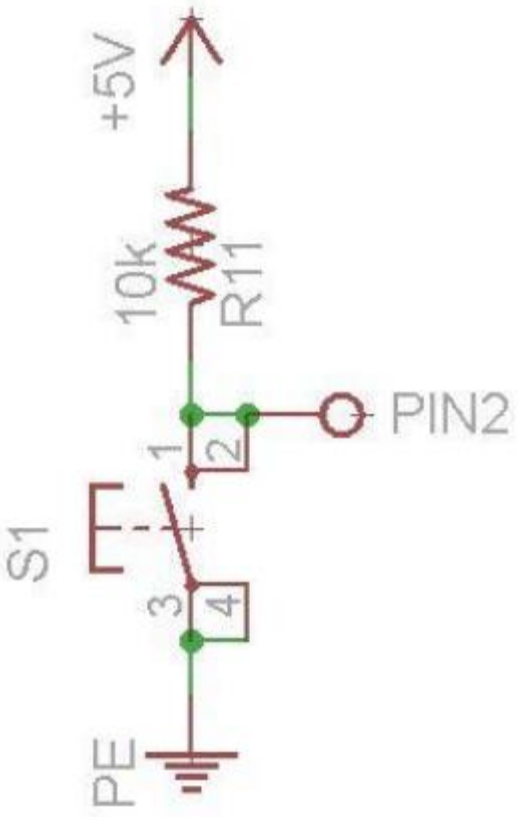
Digital Input

- Exercise:
  - Connect pin 2 to the schematic as shown.
  - Open the LED switch sketch and upload to your board.
  - Press and de-press the switch. What happens?



Digital Input

- Exercise:
  - Try to run the procedure in reverse! What do we do to change it?

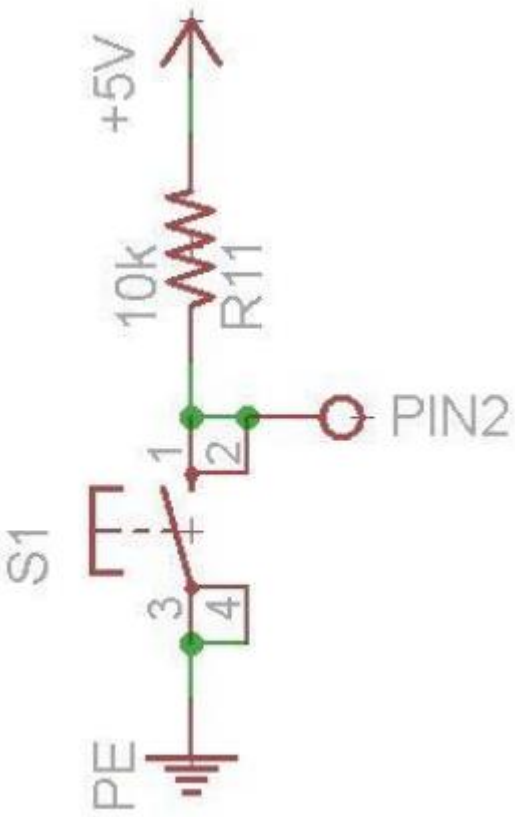


Logic

- Conditional AND
  - Symbolized by &&
  - Used when all conditions need to be satisfied.
- Conditional OR
  - Symbolized by ||
  - Used when either of the conditions need to be satisfied.
- Conditional NOT
  - Symbolized by **!**
  - Appends to other statements to invert its state

Digital Input

- Exercise:
  - Wire up another push button circuit. Modify the sketch so that the LED lights up only when both buttons are pushed.

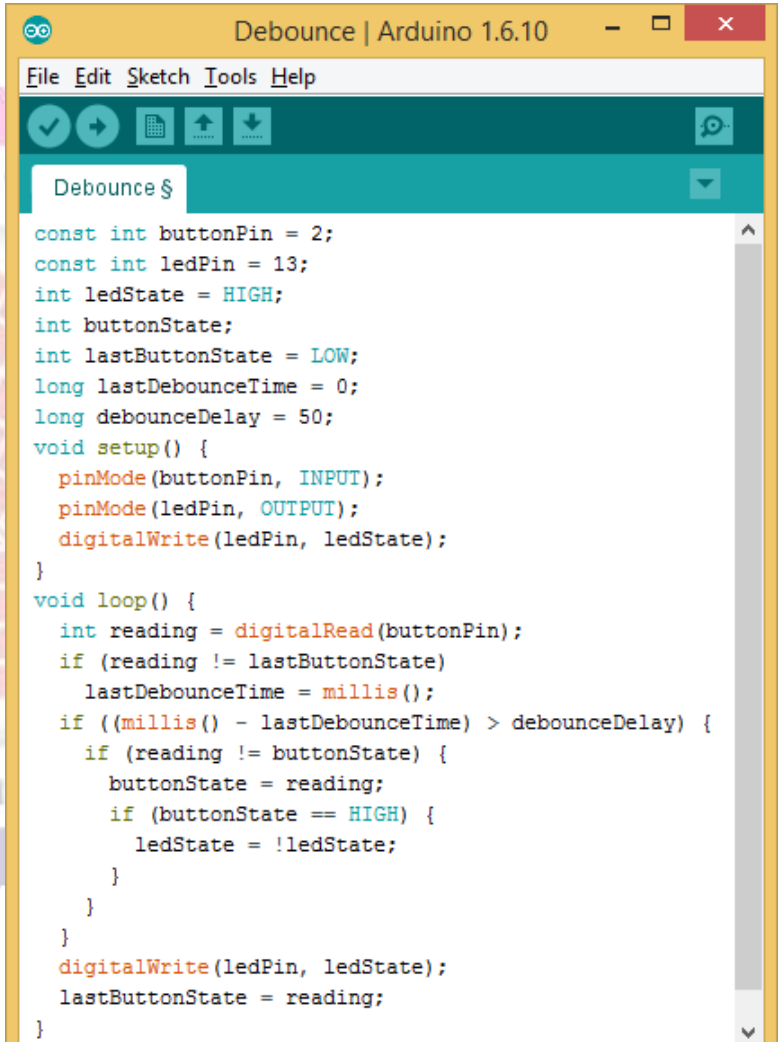


Latch and Debounce

- Push Button Debounce and Latching
  - Debounce – change state once the output is stable.
  - Latch – change state and retain it until the next change state.

Latch and Debounce

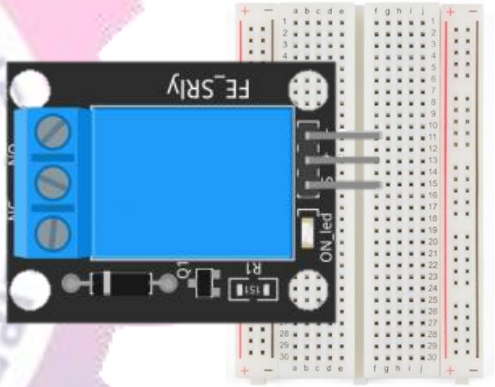
- Pushbutton Debounce and Latching
  - Debounce – change state once the output is stable.
  - Latch – change state and retain it until the next change state.




```
const int buttonPin = 2;
const int ledPin = 13;
int ledState = HIGH;
int buttonState;
int lastButtonState = LOW;
long lastDebounceTime = 0;
long debounceDelay = 50;
void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledState);
}
void loop() {
  int reading = digitalRead(buttonPin);
  if (reading != lastButtonState)
    lastDebounceTime = millis();
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }
  digitalWrite(ledPin, ledState);
  lastButtonState = reading;
}
```

5V Relay Module

- Arduino Relay Module is used to control AC circuits, the relay acts as a switch that responds to a signal received from the Arduino, it has an integrated LED that indicates if the signal is high or low.
- Commonly used in IoT projects to control lights and other electronic appliances.

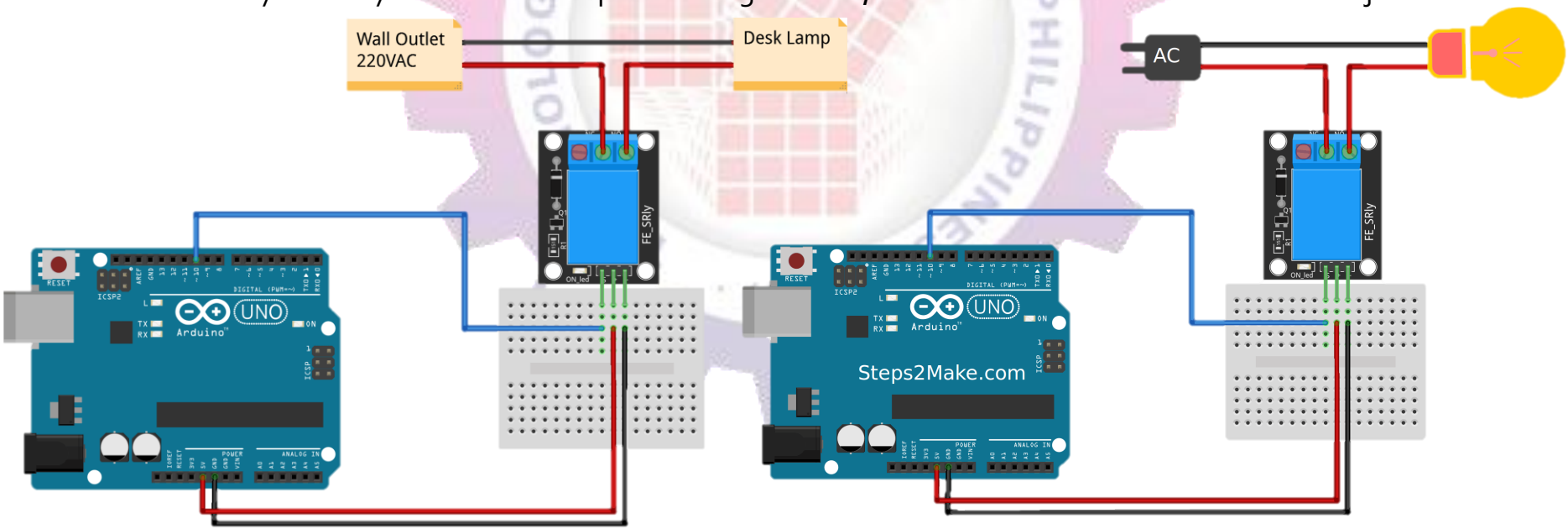


- Specifications
  - The consist of a 1MΩ resistor, a LED, a 1N4007 rectifier diode and a 5VDC relay capable of handling up to 250VAC and 10A.
  - On the DC side of the board there are 3 pins for signal, power and ground. On the AC side there are 3 contacts, NC (Normally Closed), Common and NO (Normally Open).

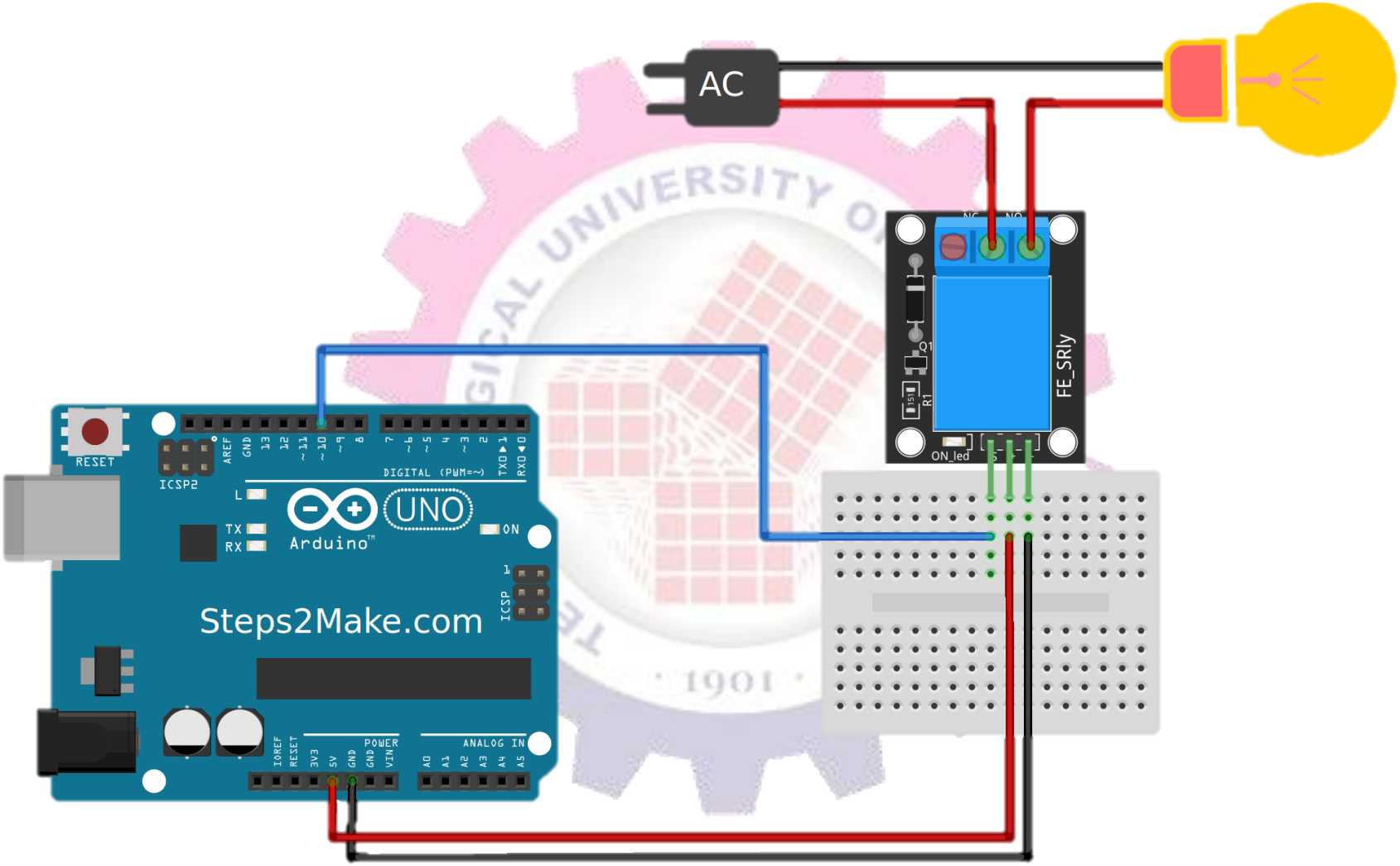
TTL Control Signal	5VDC to 12VDC (some boards may work with 3.3)
Maximum AC	10A 250VAC
Maximum DC	10A 30VDC
Contact Type	NC and NO
Dimensions	27mm x 34mm [1.063in x 1.338in]

5V Relay Module

- Connection Diagram
  - For the DC part of the circuit connect S (signal) to pin 10 on the Arduino, also connect the Power line (+) and ground (-) to +5 and GND respectively.
  - On the AC side connect your feed to Common (middle contact) and use NC or NO according to your needs.
  - NO (Normally Open) will get power when (S) is high, NC (Normally Closed) gets disconnected when (S) is high.
  - Always be very careful when experimenting with AC, electrical shock can result in serious injures.



5V Relay Module



5V Relay Module

- Example Code
  - The following Arduino sketch will turn the relay on/off every second. We'll connect a desk lamp to the relay using the NO (Normally Open) connection so the lamp is off when the relay is off. Running the sketch will cause the lamp to light up intermittently.

```
int relay = 10;  //Pin 10

void setup()
{
    pinMode(relay,OUTPUT);      // Define the port attribute as output
}

void loop()
{
    digitalWrite(relay,HIGH);   // turn the relay ON
                                // [NO] is connected to feed
                                // [NC] is not connected to feed
    delay(1000);

    digitalWrite(relay,LOW);    // turn the relay OFF
                                // [NO] is not connected to feed
                                // [NC] is connected to feed
    delay(1000);
}
```

Serial Communications

Initializing Serial

- Serial.begin()
  - Initialize serial communication
  - Commonly placed at the setup function of your sketch
- Syntax:
  - Serial.begin(baud)
    - baud: serial baud rate value to be used
    - e.g. 300, 1200, 2400, 4800, 9600, 14400, 28800, 38400, 57600, 115200)
    - Ex. Serial.begin(9600);

Printing Lines using Serial

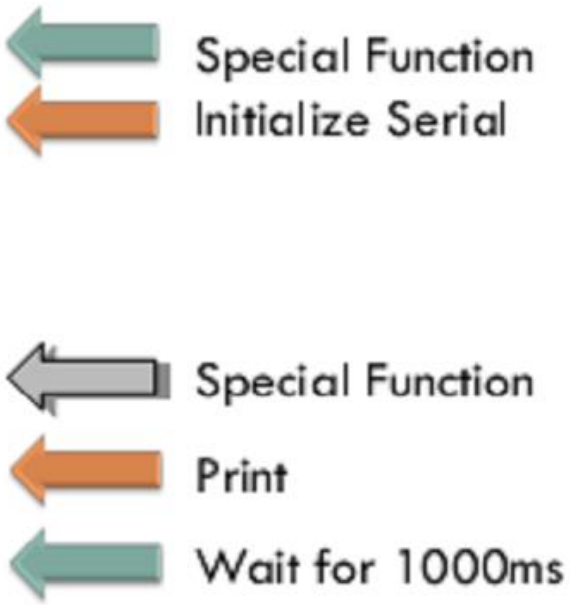
- Serial.println()
  - Prints data to the Serial port as ASCII text
- Syntax:
  - Serial.println(val)
    - val: the value to print, which can be any data type
    - Ex. Serial.println("Hello World!");
      - Displays Hello World! In Serial Monitor
  - Serial.println(val, format)
    - format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

Printing Lines using Serial


```
int analogValue = 0;    // variable to hold the analog value

void setup() {
  // open the serial port at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // read the analog input on pin 0:
  analogValue = analogRead(0);

  // print it out in many formats:
  Serial.println(analogValue);       // print as an ASCII-encoded decimal
  Serial.println(analogValue, DEC);  // print as an ASCII-encoded decimal
  Serial.println(analogValue, HEX);  // print as an ASCII-encoded hexadecimal
  Serial.println(analogValue, OCT);  // print as an ASCII-encoded octal
  Serial.println(analogValue, BIN);  // print as an ASCII-encoded binary

  // delay 10 milliseconds before the next reading:
  delay(10);
}
```

Sample Code

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

void loop() {
        Serial.println("Hello World!");
        delay(1000);
}
```



Available Data from Serial

- Serial.available()
  - Checks if there is data present on the line and returns the number of bytes to be read
  - Syntax:
    - Serial.available()
      - Ex. if (Serial.available() > 0) { …

Reading from Serial

- Serial.read()
  - Returns the first byte of incoming serial data
  - Syntax:
    - variable = Serial.read();
      - variable can be either an int or a char

Sample Code

```
void setup() {
        Serial.begin(9600);
}
void loop() {
        if (Serial.available() > 0) {
                ReceivedByte = Serial.read();
                Serial.print(ReceivedByte);
                delay(10);
        }
}
```

Analog Input

Analog Input

- Arduino can "recognize" analog voltage by converting it into a digital output.
- The Atmel chip has an onboard 6 channel, 10-bit analog-to-digital converter.
- Analog pins also have the same functionality of general purpose input/output (GPIO) pins.

Analog Input

- analogRead()
  - Function to perform ADC
  - Returns value from **0** to **1023** levels between **0V** to **5V**
- Syntax:
  - variable = analogRead(A5);
- Value 'channel' is the label of the analog pin used

Analog Input

- Connect a potentiometer to the Arduino as shown

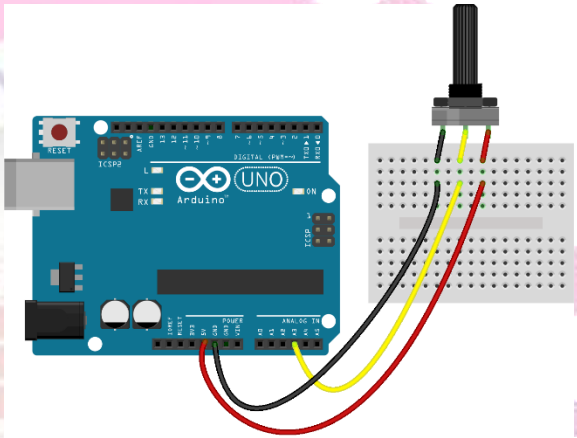
sketch_apr19a | Arduino 1.8.13

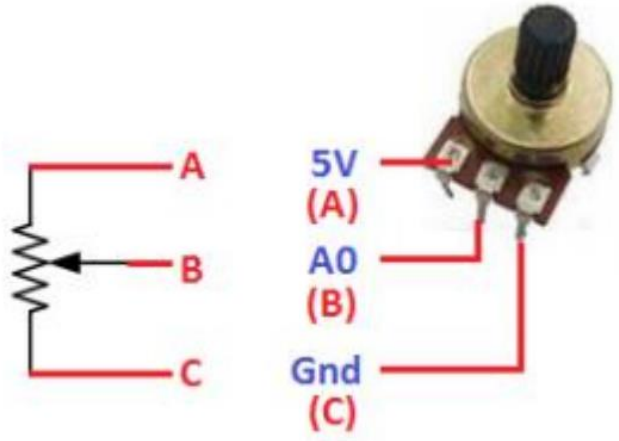
File Edit Sketch Tools Help

sketch_apr19a §

```
1 void setup() {
2   Serial.begin(9600);
3 }
4
5 void loop() {
6   float var = analogRead(A0);
7   Serial.println(var);
8   delay(50);
9 }
```

8 Arduino Uno on COM8





- Open ADC sketch. Open the Serial Monitor and turn the knob. What happens?

Analog Input

- Example
  - Create a sketch that will BLINK an LED faster or slower when turning the potentiometer.

Analog Input

- Example
  - Create a sketch that controls the intensity of an LED using a potentiometer.
  - Hint: The variable that will change is not delay()..

map()

- Re-maps a number from one range to another. That is, a value of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.
- The map() function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged.
- Syntax
  - **int var1 = map(var, 0, 1023, 0, 255);** **var*100/1023**
- Parameters
  - value: the number to map
  - fromLow: the lower bound of the value's current range
  - fromHigh: the upper bound of the value's current range
  - toLow: the lower bound of the value's target range
  - toHigh: the upper bound of the value's target range
- Returns
  - The mapped value.

Map and AnalogRead

Map and AnalogRead

The Arduino's ADCs can read 1024 levels between 0V and 5V, and so the value returned by the *analogRead* function is an integer in the range 0 through 1023. For many uses, this is fine. However, using map you can scale the value back to the range 0–5, which might be more helpful when specifically measuring voltages. For example:

```
int v2 = map(v1, 0, 1023, 0, 5);
```

Map accepts five arguments:

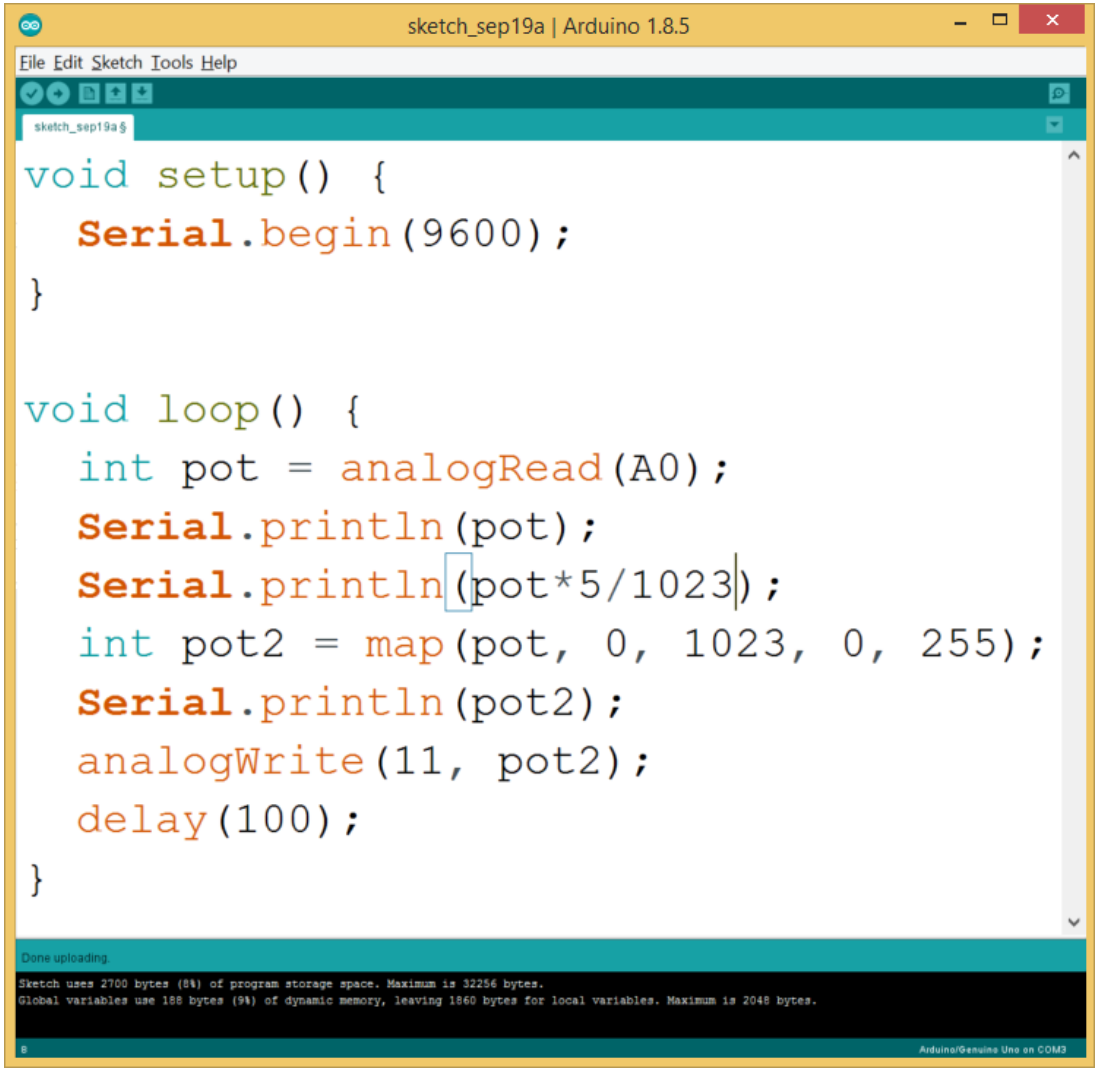
1. The value to be scaled (v1).
2. The start value of the source range (0).
3. The end value of the source range (1023).
4. The start value of the target range (0).
5. The end value of the target range (5).

map()

Example Code

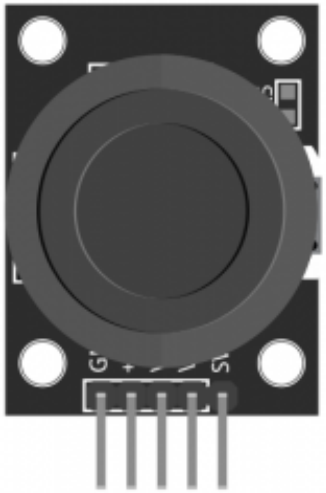
```
/* Map an analog value to 8 bits (0 to 255) */
void setup() {}

void loop()
{
  int val = analogRead(0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

map()


sketch_sep19a | Arduino 1.8.5
File Edit Sketch Tools Help
sketch_sep19a §
void setup() {
Serial.begin(9600);
}
void loop() {
int pot = analogRead(A0);
Serial.println(pot);
Serial.println(pot*5/1023);
int pot2 = map(pot, 0, 1023, 0, 255);
Serial.println(pot2);
analogWrite(11, pot2);
delay(100);
}
Done uploading.
Sketch uses 2700 bytes (8%) of program storage space. Maximum is 32256 bytes.
Global variables use 188 bytes (9%) of dynamic memory, leaving 1860 bytes for local variables. Maximum is 2048 bytes.
Arduino/Genuino Uno on COM3

Dual Axis Joystick Module

- Arduino joystick module, it uses a biaxial potentiometer to control the X and Y axis. When pushed down, it activates a switch. Based on the PS2 controller's joystick, it's used to control a wide range of projects from RC vehicles to color LEDs.

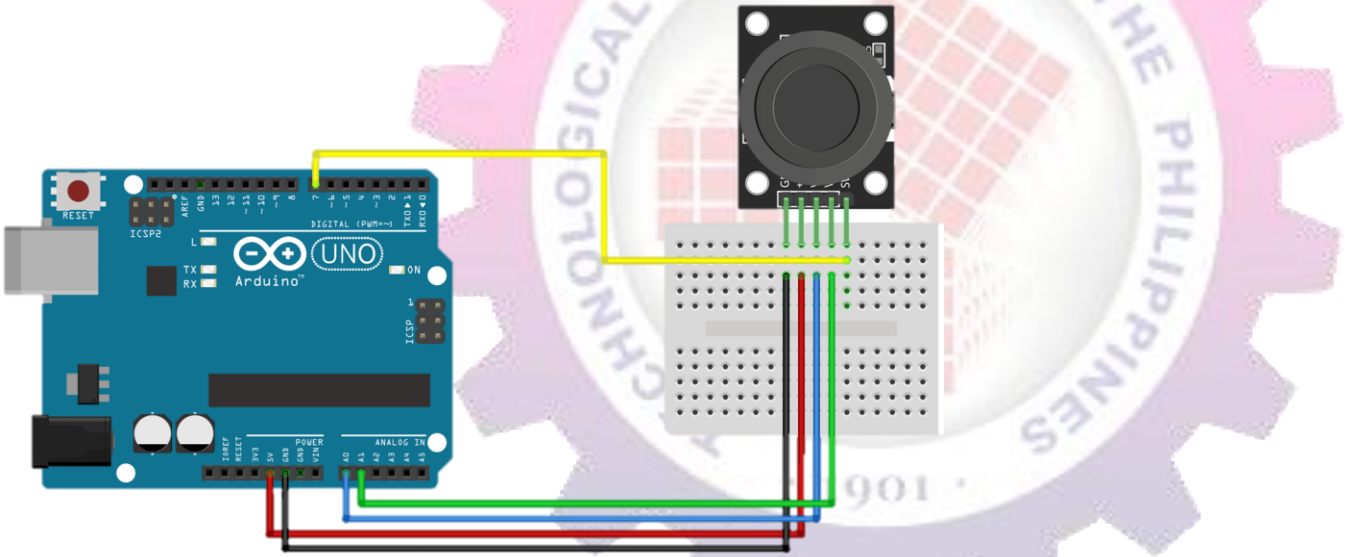


- Specifications
  - The Joystick module consists of two 10k potentiometers perpendicularly positioned to control the X and Y axis by changing the resistance when the joystick is moved. A push button is activated when the joystick is pushed down the Z axis.

Operating Voltage	3.3V to 5V
Board Dimensions	2.6cm x 3.4cm [1.02in x 1.22in]

Dual Axis Joystick Module

- Arduino joystick module, it uses a biaxial potentiometer to control the X and Y axis. When pushed down, it activates a switch. Based on the PS2 controller's joystick, it's used to control a wide range of projects from RC vehicles to color LEDs.



Dual Axis Joystick Module

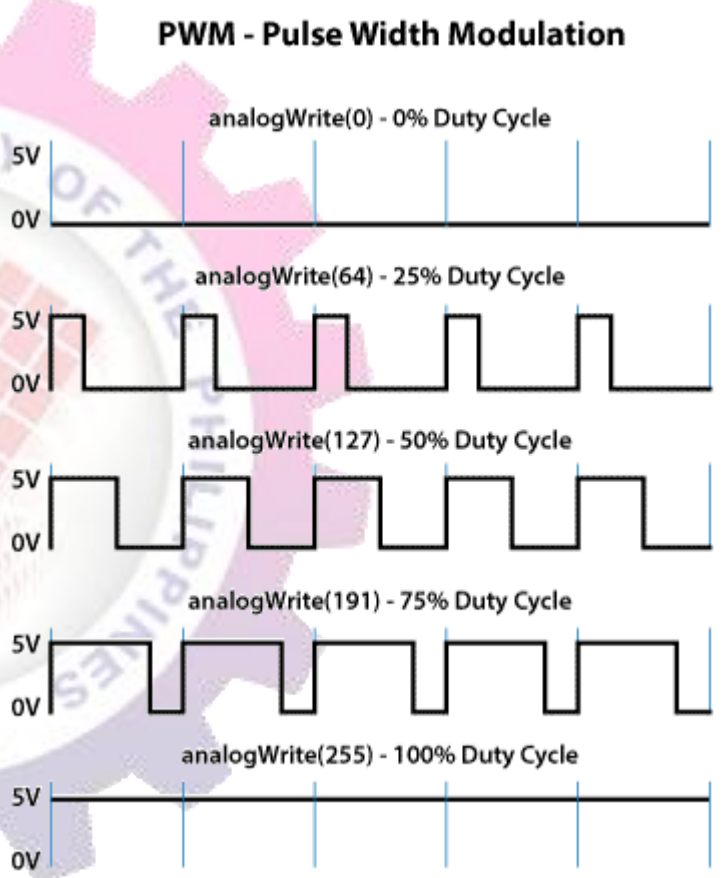
- **Example Code**
  - The following Arduino sketch will continually read values from the potentiometers and button on the module. Moving the joystick up/down will increase/decrease the values of X and moving the joystick left/right will increase / decrease for values of Y. Push the joystick down to activate the button.

```
int value = 0;

void setup() {
    //pinMode(A0, INPUT);
    //pinMode(A1, INPUT);
    pinMode(7, INPUT);
    Serial.begin(9600);
}

void loop() {
  value = analogRead(A0);   // read X axis value [0..1023]
  Serial.print("X:");
  Serial.print(value, DEC);

  value = analogRead(A1);   // read Y axis value [0..1023]
  Serial.print(" | Y:");
  Serial.print(value, DEC);

  value = digitalRead(7);   // read Button state [0,1]
  Serial.print(" | Button:");
  Serial.println(value, DEC);

  delay(100);
}
```

Analog Output

Analog Output

- Reverse method of analog read: from a specific value from a range, produce the equivalent analog voltage.
- Range: 0 – 255
- Pin is turned on and off very fast – achieves an "averaged" effect on the output side



analogWrite()

- analogWrite()
  - Supported pins: 3, 5, 6, 9, 10, 11
- Syntax:
  - analogWrite(pin, var1)
- Value 'pin' is the label of the digital pin used
- Value 'range' is a value between 0-255

Active Buzzer Module

- Active Buzzer Module Arduino module, it produces a single-tone sound when signal is high. To produce different tones use the Passive Buzzer module.

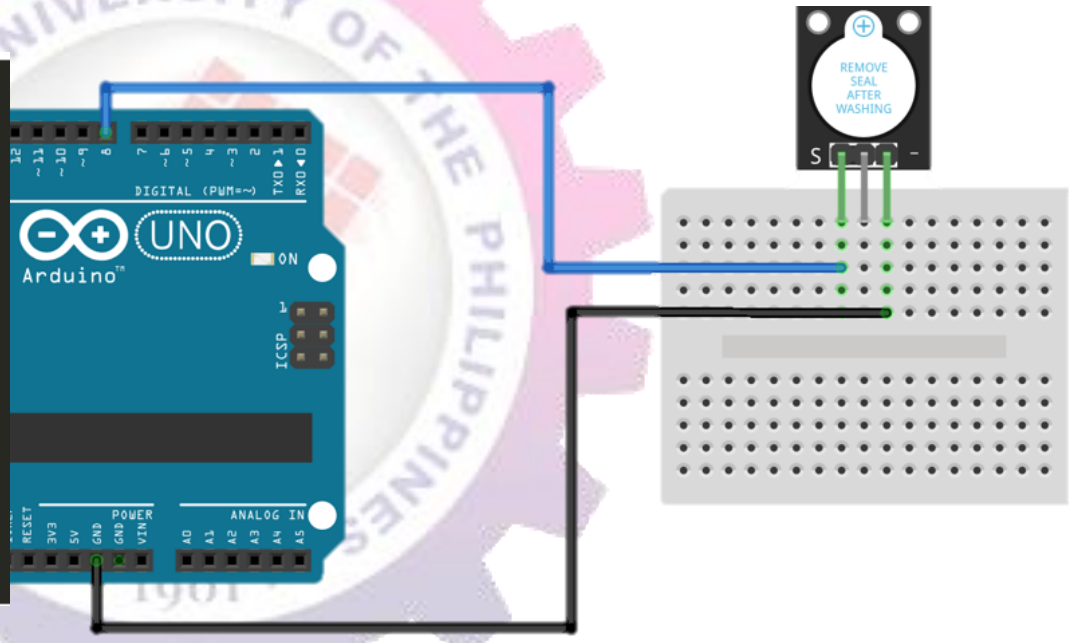
- Specifications
  - The Active Buzzer module consists of an active piezoelectric buzzer, it generates a sound of approximately 2.5kHz when signal is high.

Operating Voltage	3.5V ~ 5.5V
Maximum Current	30mA / 5VDC
Resonance Frequency	2500Hz ± 300Hz
Minimum Sound Output	85Db @ 10cm
Working Temperature	-20°C ~ 70°C [-4°F ~ 158°F]
Storage Temperature	-30°C ~ 105°C [-22°F ~ 221°F]
Dimensions	18.5mm x 15mm [0.728in x 0.591in]

Active Buzzer Module

- Connection Diagram
  - Connect signal (S) to pin 8 on the Arduino and Ground (-) to GND. Be aware that some boards are wrongly labeled, try inverting the cables if you can't hear any sound when running the sketch.

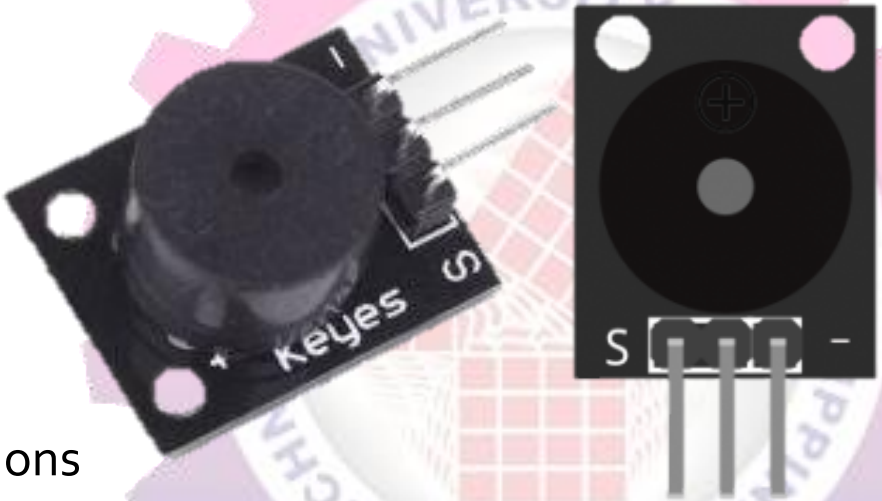
```
1  int buzzerPin = 8;
2
3  void setup ()
4  {
5    pinMode (buzzerPin, OUTPUT);
6  }
7
8  void loop ()
9  {
10   digitalWrite (buzzerPin, HIGH);
11   delay (500);
12   digitalWrite (buzzerPin, LOW);
13   delay (500);
14 }
```



- Example Code
  - The following Arduino Sketch will continually turn the buzzer on and off generating a series of short high-pitched beeps.

Passive Buzzer Module

- Arduino Passive Piezoelectric Buzzer Module can produce a range of sound tones depending on the input frequency.



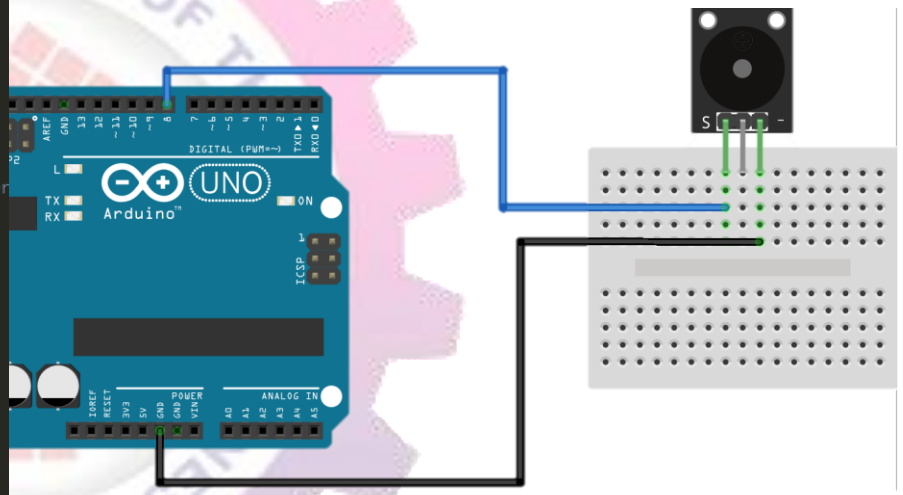
- Specifications
  - The Passive Buzzer Module consists of a passive piezoelectric buzzer, it can generate tones between 1.5 to 2.5 kHz by switching it on and off at different frequencies either using delays or PWM.

Operating Voltage	1.5 ~ 15V DC
Tone Generation Range	1.5 ~ 2.5kHz
Dimensions	18.5mm x 15mm [0.728in x 0.591in]

Passive Buzzer Module

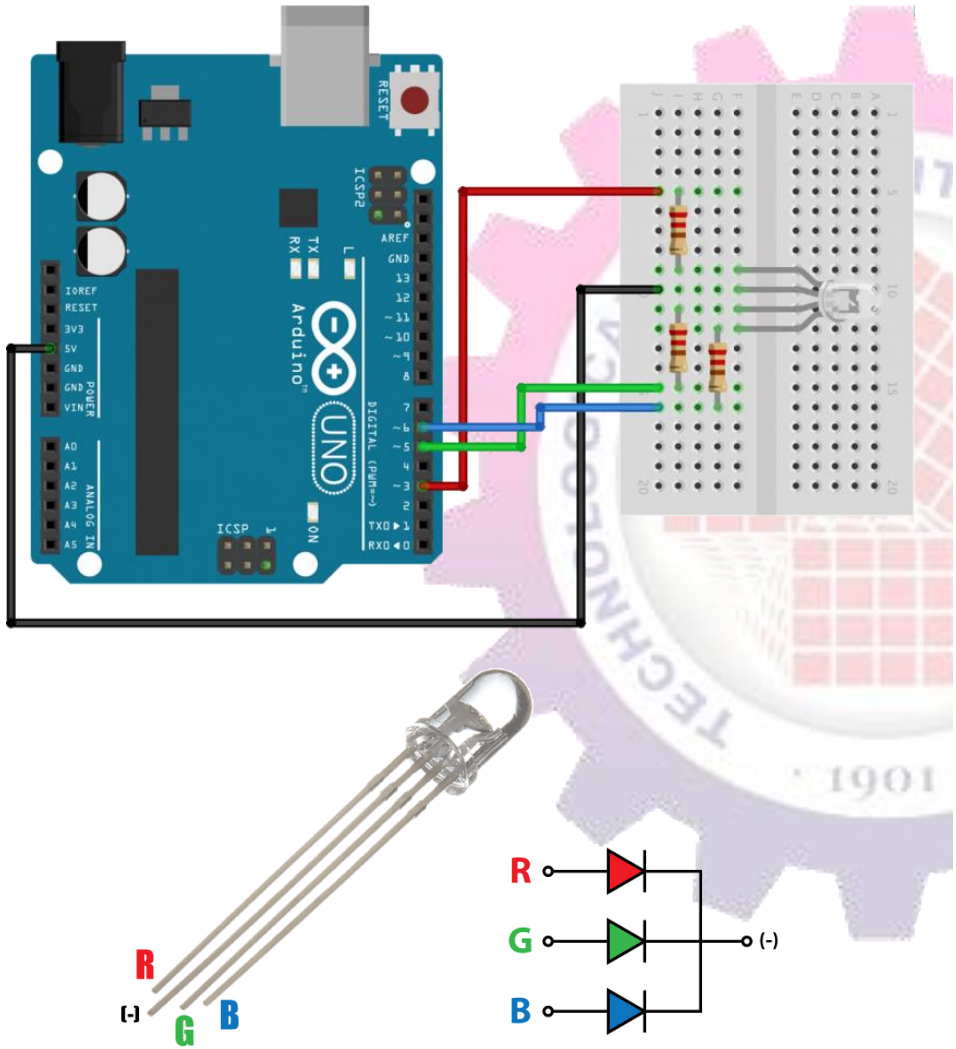
- Connection Diagram
  - Connect signal (S) to pin 8 on the Arduino and ground (-) to GND. The middle pin is not used.

```
int buzzer = 8; // set the buzzer control digital IO pin

void setup() {
    pinMode(buzzer, OUTPUT); // set pin 8 as output
}

void loop() {
    for (int i = 0; i < 80; i++) {  // make a sound
        digitalWrite(buzzer, HIGH); // send high signal to buzzer
        delay(1); // delay 1ms
        digitalWrite(buzzer, LOW); // send low signal to buzzer
        delay(1);
    }
    delay(50);
    for (int j = 0; j < 100; j++) { //make another sound
        digitalWrite(buzzer, HIGH);
        delay(2); // delay 2ms
        digitalWrite(buzzer, LOW);
        delay(2);
    }
    delay(100);
}
```



- Example Code
  - The following Arduino sketch will generate two different tones by turning on and off the passive buzzer at different frequencies using a delay.

RGB Full color (common cathode)




```arduino
int redPin= 7;
int greenPin = 6;
int bluePin = 5;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  setColor(255, 0, 0); // Red Color
  delay(1000);
  setColor(0, 255, 0); // Green Color
  delay(1000);
  setColor(0, 0, 255); // Blue Color
  delay(1000);
  setColor(255, 255, 255); // White Color
  delay(1000);
  setColor(170, 0, 255); // Purple Color
  delay(1000);
}

void setColor(int redValue, int greenValue, int blueValue) {
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```

RGB LED (common Anode)

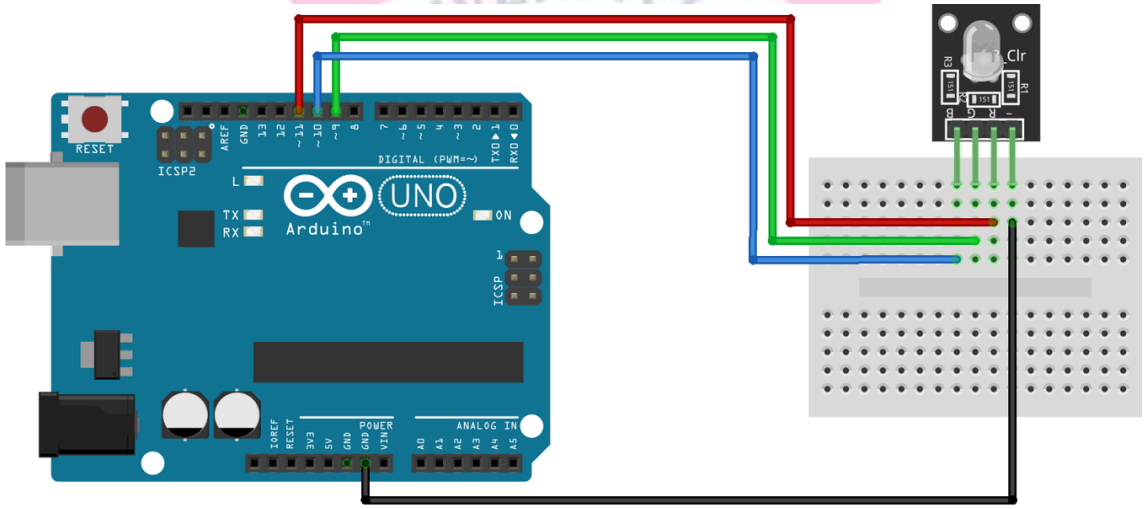
- Exercise:
  - Connect a 4-pin RGB LED to your Arduino. Longest pin goes to +5V, as for the other pins follow the configurations below
  - Open and upload the RGB_LED sketch, what happens?
  - CHALLENGE – Make a rainbow!
    - Sweep values using PWM and produce the colors of the rainbow in sequence

RGB LED (common Anode)

- Exercise:
  - Connect a 4-pin RGB LED to your Arduino. Longest pin goes to +5V, as for the other pins follow the configurations below
  - Open and upload the RGB_LED sketch, what happens?
  - CHALLENGE – Make a rainbow!
    - Sweep values using PWM and produce the colors of the rainbow in sequence

RGB Full color LED module

Arduino full color 5mm RGB LED, different colors can be obtained b
mixing the three primary colors.



Specifications

This module consists of a 5mm RGB LED and three 150Ω limiting resistors to prevent burnout. Adjusting the PWM signal on each color pin will result on different colors.

Operating Voltage	5V
LED drive mode	Common cathode driver
LED diameter	5 mm

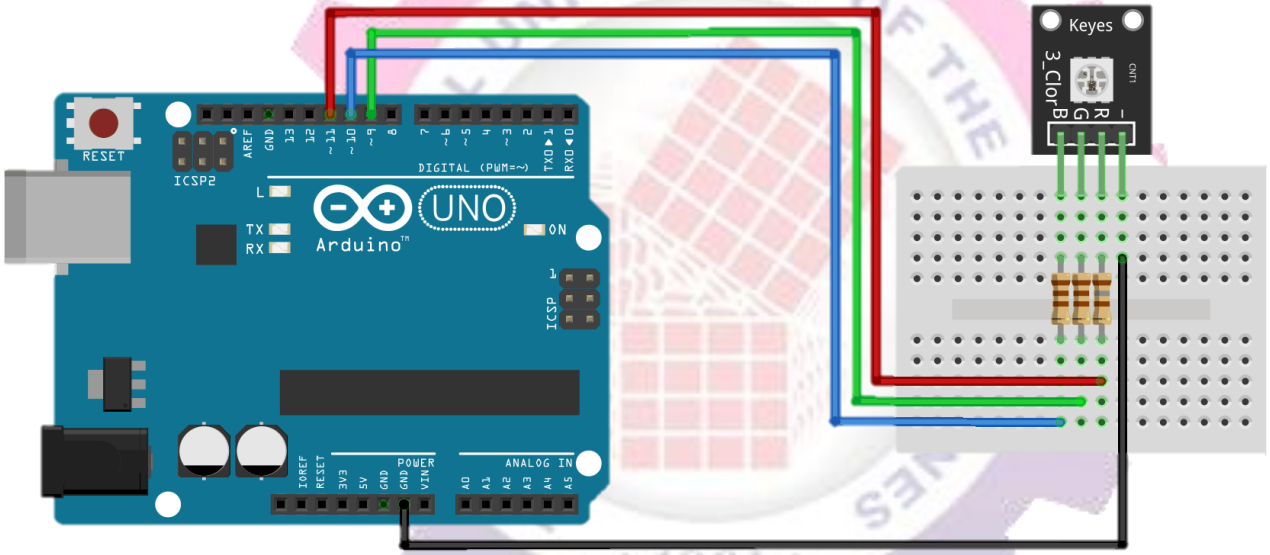
Example Code

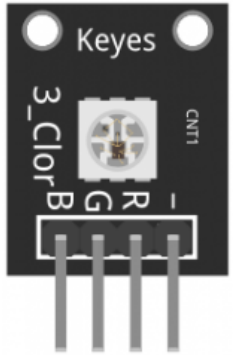
- The following Arduino sketch will gradually increase/decrease the PWM values on the red, green and blue pins causing the LED to cycle through various colors.

```
int redpin = 11; // select the pin for the red LED
int bluepin =10; // select the pin for the  blue LED
int greenpin =9; // select the pin for the green LED

int val;

void setup() {
  pinMode(redpin, OUTPUT);
  pinMode(bluepin, OUTPUT);
  pinMode(greenpin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  for(val = 255; val > 0; val--)
  {
    analogWrite(11, val);
    analogWrite(10, 255 - val);
    analogWrite(9, 128 - val);

    Serial.println(val, DEC);
    delay(5);
  }
  for(val = 0; val < 255; val++)
  {
    analogWrite(11, val);
    analogWrite(10, 255 - val);
    analogWrite(9, 128 - val);

    Serial.println(val, DEC);
    delay(5);
  }
}
```

RGB Full color LED SMD Module

RGB full color LED Module KY-009 for Arduino, emits a range of colors by mixing red, green and blue. The amount of each primary color is adjusted using PWM.





Specifications

The RGB Full Color LED SMD Module consists of a 5050 SMD LED, use with limiting resistors to prevent burnout. Compatible with popular electronics platforms like Arduino, Raspberry Pi and ESP8266.

Example Code

- The following Arduino sketch will cycle through various colors by changing the PWM value on each of the three primary colors.

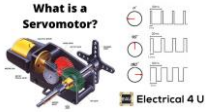
```
int redpin = 11; //select the pin for the red LED
int bluepin =10; // select the pin for the  blue LED
int greenpin = 9;// select the pin for the green LED

int val;

void setup() {
  pinMode(redpin, OUTPUT);
  pinMode(bluepin, OUTPUT);
  pinMode(greenpin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  for(val = 255; val > 0; val--)
  {
    analogWrite(redpin, val);  //set PWM value for red
    analogWrite(bluepin, 255 - val); //set PWM value for blue
    analogWrite(greenpin, 128 - val); //set PWM value for green
    Serial.println(val); //print current value
    delay(1);
  }
  for(val = 0; val < 255; val++)
  {
    analogWrite(redpin, val);
    analogWrite(bluepin, 255 - val);
    analogWrite(greenpin, 128 - val);
    Serial.println(val);
    delay(1);
  }
}

```

Motor

Types of Motors

- AC/DC Motors – simple to connect and use (just two leads), driven via voltage, not very precise.

- Servo Motors – relatively simple to connect and use (three leads), driven to a particular position or at a particular speed via one or two control commands.



- Stepper Motors – complex to connect and use (four or more leads), driven to a particular position via special sequencing of control commands, can move very precise distances at very precise speeds

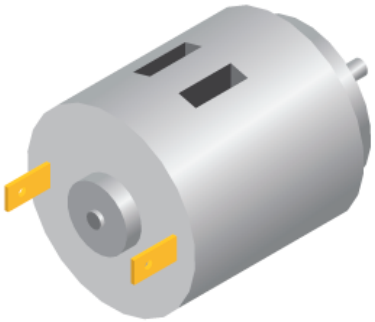
Types of Motors

DC motors

- A DC motor can run freely in both directions, but is very difficult to control when it comes to speed or position. It's not easy to make it stop with exact precision. It comes with two cables: power and ground. Note that generally, an Arduino digital pin cannot power a DC motor.

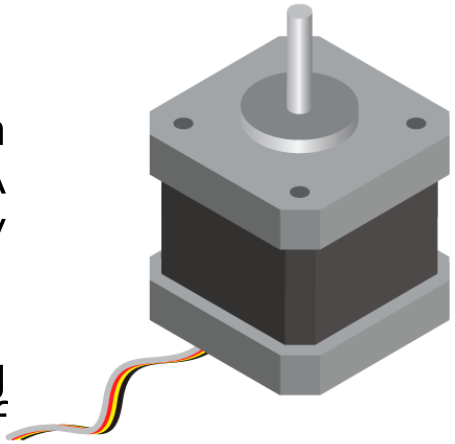
- The cables can be connected to ground or to a digital pin. Set the digital pin to HIGH to make it move and to LOW to make it stop. To rotate counterclockwise, reverse the wire connections.

- It is possible to control the speed of a DC motor using a transistor and a technique called pulse width modulation (PWM). With several transistors arranged on an H-bridge, you can even control the direction without physically changing the connections.



Types of Motors

Stepper motors



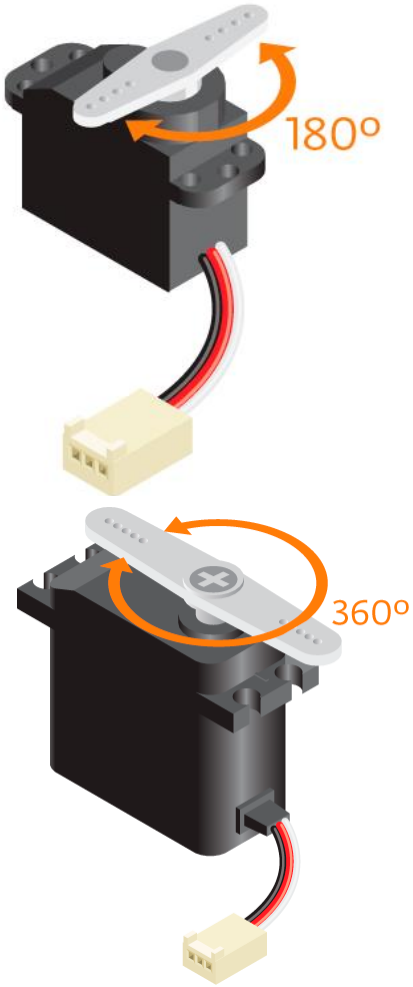
- Stepper motors can be found in electronics where high precision is important such as scanners and printers. A stepper motor, in contrast to the DC motor, can be very precise with both position and speed.
- A stepper motors full rotation is divided into equally big steps and you can control the motor to stop at each of these steps. The steps are measured in degrees, usually 1.8, 3.6 or 7.2. The smaller the steps, the more precise. This makes it very useful when repeated positioning is needed.
- Stepper motors will never be very fast compared to a DC motor. It generally has 4 or more wires and usually needs more than 5 volts to work. This means it cannot be powered by an Arduino, but we can use an external power supply.

Types of Motors

Servo motors

- Servo motors are often used in robotics and toys. These are the type of motors that you are going to use for your project because they are simple to connect and to control from an Arduino. They have three wires: one for power, one for ground, and one for the control signal.

- There are two types: a standard rotation servo and a continuous rotation servo. The standard rotation servo can rotate 180 degrees with precision like the stepper motors. The continuous rotation servo is similar to a DC motor and can spin in both directions, but not as fast. You can control both the speed and the direction without the use of transistors.



DC Motor

- DC Motors are tricky to connect to microcontrollers!
  - Needs a lot of power to run
  - Produces lots of electrical noise that may interfere with microcontroller pin outputs
  - Usual and safest practice to provide separate power source for the DC Motor
  - Ensure all ground points (GND) are connected (common)

DC Motor

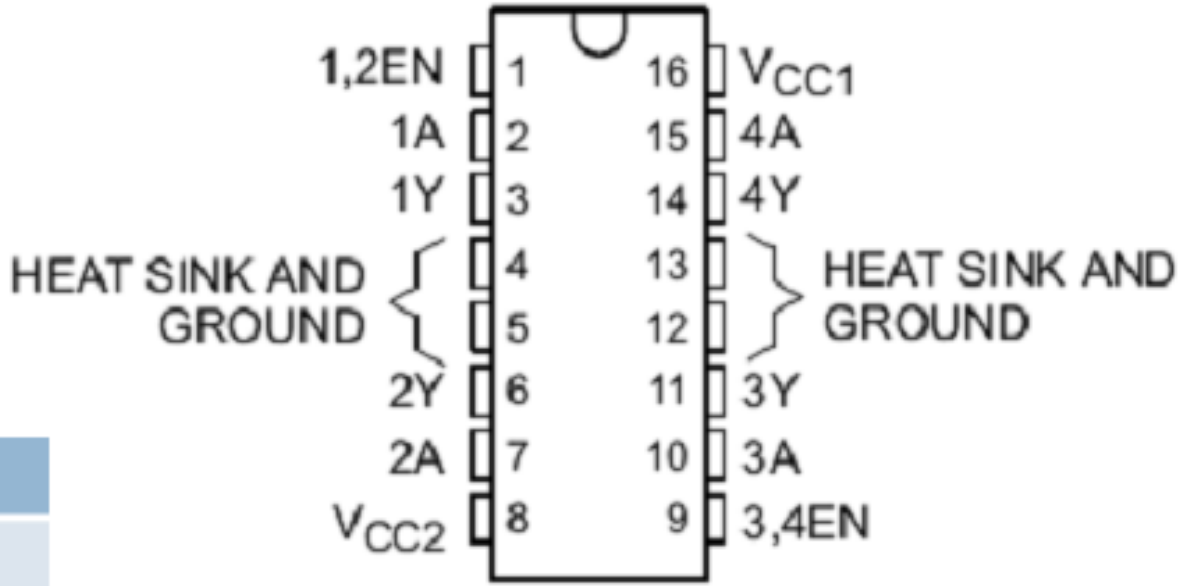
- SPEED
  - Pulse Width Modulation (PWM)
- DIRECTION
  - Motor Direction follows Source Current Direction
  - Needs external circuitry that will reverse the direction of the current coming from the motor's power supply
  - Direction is binary logic!

DC Motor

- Numerous ways to implement this control scheme!
  - Collective term – DC Motor Driver
  - H-Bridge configuration is the most commonly used and simplest to build
- L293D – Quadruple Half H-Bridge Drivers
  - Single-chip solution for bi-directional control of up to 2 DC Motors
  - Internal protection circuitry and wide supply voltage range allows easy and direct interfacing with most microcontroller systems

DC Motor

L293D	MOTOR
1Y	Motor Pin
2Y	Motor Pin

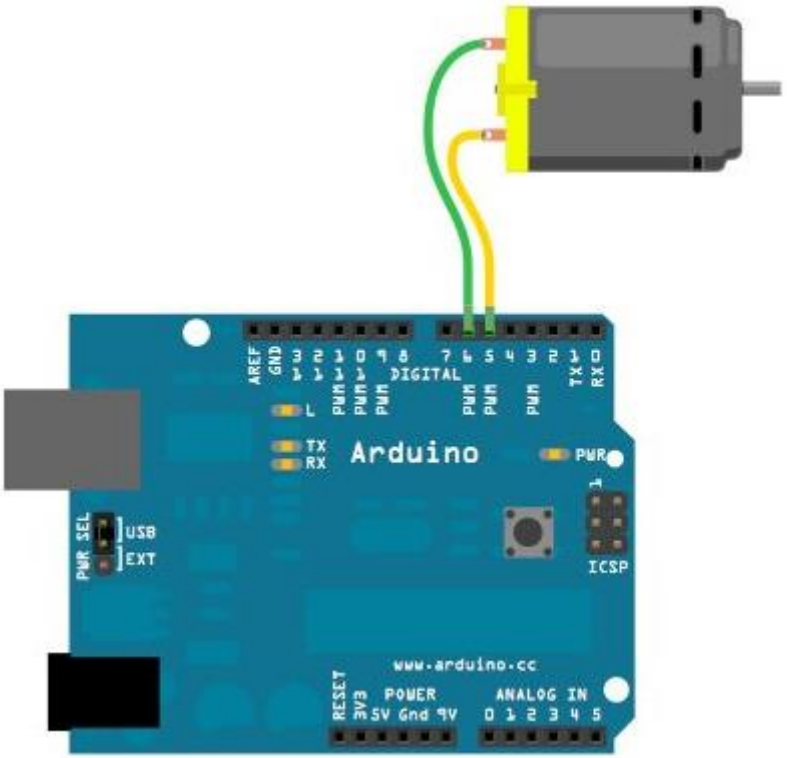


L293D	GIZDUINO
1,2EN	Pin 5
2A	Pin 4
VCC 1	5V
Heat Sink and Ground	GND

L293D	BATTERY
VCC 2	Red Wire
Heat Sink and Ground	Black Wire

DC Motor

- DC Motor without driver.



dc_motor | Arduino 1.8.5

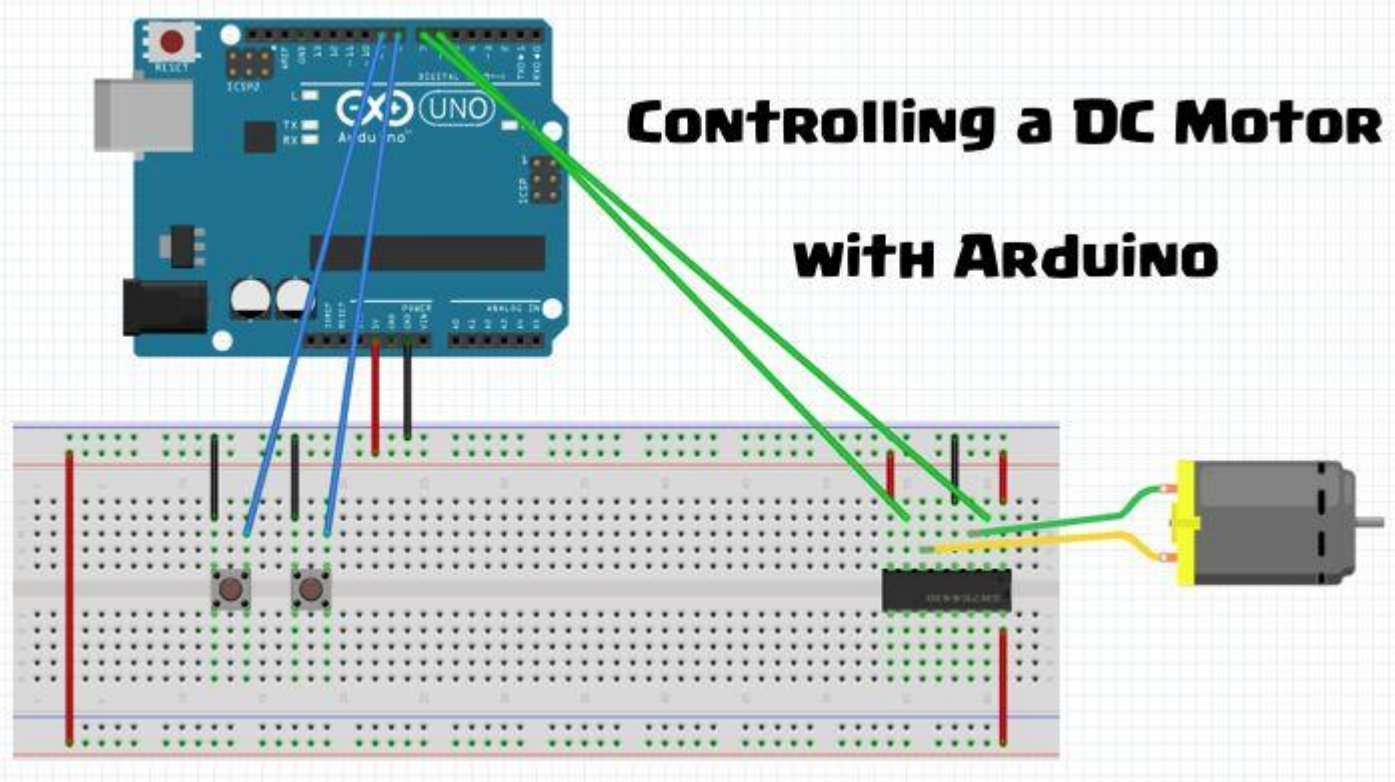
File Edit Sketch Tools Help

dc_motor §

```
int PWMPin = 5;
int DirPin = 6;
int switchPin = 2;

void setup() {
  pinMode(DirPin, OUTPUT);
  pinMode(switchPin, INPUT);
  analogWrite(PWMPin, 255);
}

void loop() {
  int switch_state = digitalRead(switchPin);
  if (switch_state == 0) {
   digitalWrite(DirPin, LOW);
  }
  else {
    digitalWrite(DirPin, HIGH);
    }
}
```

DC Motor with L293D

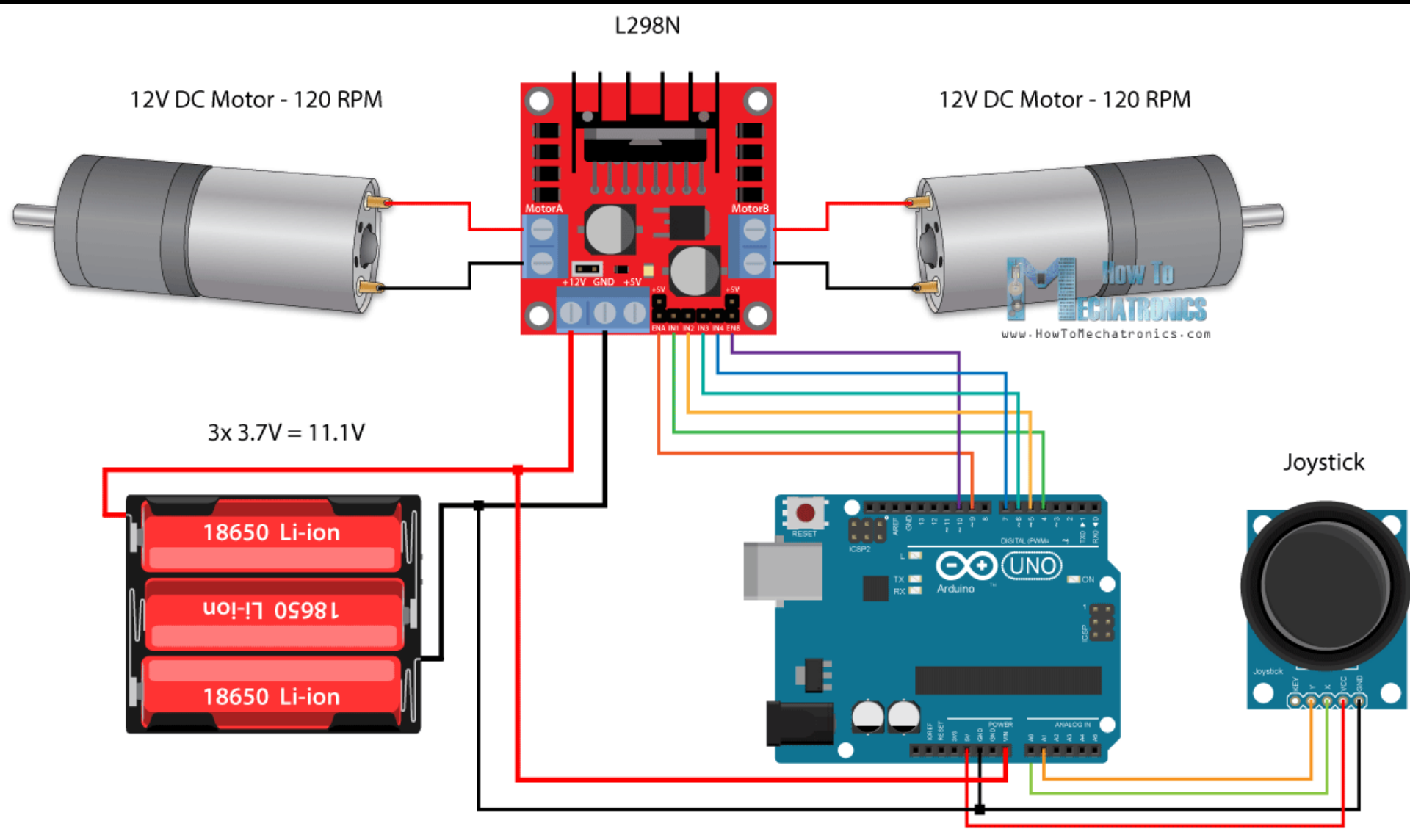
- DC Motor with L293D driver.



DC Motor with L293D


```
int inlpin = 6;
int in2pin = 7; // h bridge pins
int leftButton = 8;
int rightButton = 9; // buttons
void setup() {
  pinMode(inlpin, OUTPUT);
  pinMode(in2pin, OUTPUT); // outputs
  pinMode(leftButton, INPUT_PULLUP);
  pinMode(rightButton, INPUT_PULLUP); // inputs w internal pullup resistors
}
void loop() {
  int leftPinState = digitalRead(leftButton);
  int rightPinState = digitalRead(rightButton); // set value names for read data
  if (leftPinState == LOW) { // if left button is pressed ...
    digitalWrite(inlpin, HIGH); // make motor go one way
    digitalWrite(in2pin, LOW);
  }
  else if (rightPinState == LOW) { // if right button is pressed ...
    digitalWrite(inlpin, LOW);
    digitalWrite(in2pin, HIGH); // make motor go other way
  }
  else { // if neither button is pressed ...
    digitalWrite(inlpin, LOW); // nothing happens
    digitalWrite(in2pin, LOW);
  }
}
```

DC Motor with Driver



Library Concept

Library Concept

- Additional sketch files composed of a series of instructions and routines tailored specifically to interface with a certain device.
- Condenses a long and complicated sequence of instructions into a single line command.
- Different libraries exist for different devices and modules
- Not all libraries are made to be compatible with similar modules due to varying hardware designs and specifications.
- Libraries may conflict with other libraries due to the use of the same interfaces

Program Library

- What are Libraries?
  - Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc.
- How to Install a Library
  - Using the Library Manager
  - Importing a .zip Library
  - Manual installation

Servo Motor

- Composed of an electric motor mechanically linked to a potentiometer
  - PWM signals are used as position commands
  - 3 wire interface – Power, Ground, Control

Servo	Arduino
Red	5V
Brown	GND
Yellow	PIN 9

Servo Motor

- Arduino has a standard library for servos:
  - Sketch - > Import Library -> Servo
- You need to create first a Servo object using the following command:
  - Servo myservo;
  - myservo is the servo object

Servo Motor

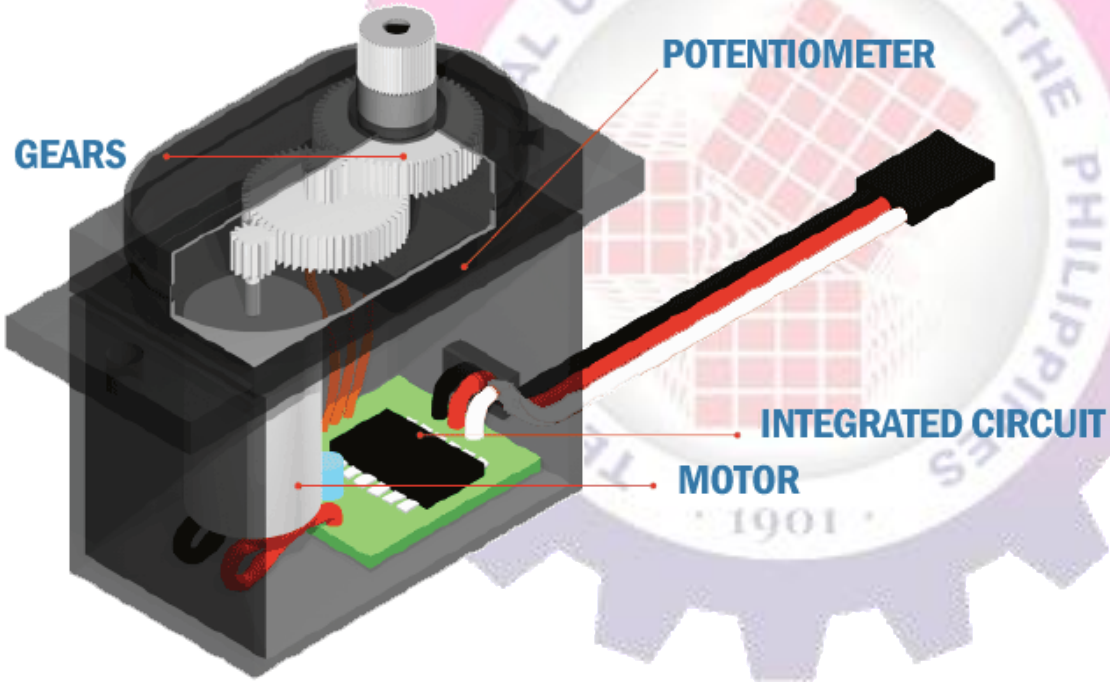
- Attach the servo to a pin
- Syntax:
  - attach(pin)
    - Pin – the pin number that the servo is attached to
- Example:
  - myservo.attach(3);

Servo Motor

- write()
  - Set the angle of the shaft to the corresponding angle.
- Syntax:
  - write(angle);
    - angle – the value to write to the servo, from 0 to 180
    - ex: **myservo.write(180);**

Servo Motor

- A servomotor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration.



Servo Motor

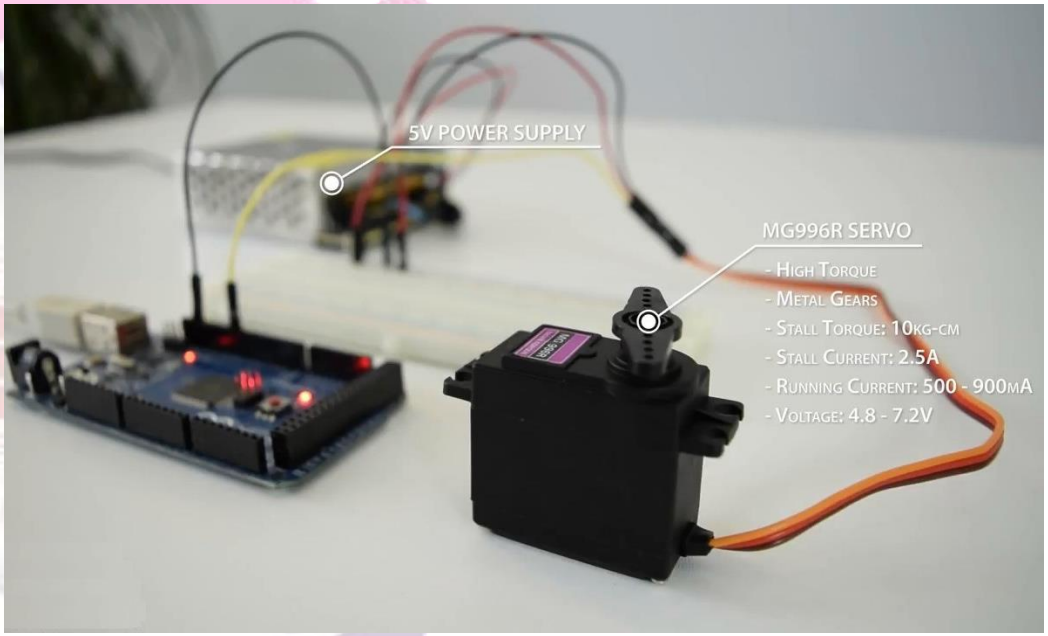
Knob | Arduino 1.8.5

File Edit Sketch Tools Help

Knob §

```
#include <Servo.h>
Servo myservo;
int potpin = 0;
int val;
void setup() {
  myservo.attach(9);
}
void loop() {
  val = analogRead(potpin);
  val = map(val, 0, 1023, 0, 180);
  myservo.write(val);
  delay(15);
}
```

13 Arduino/Genuino Uno on COM1

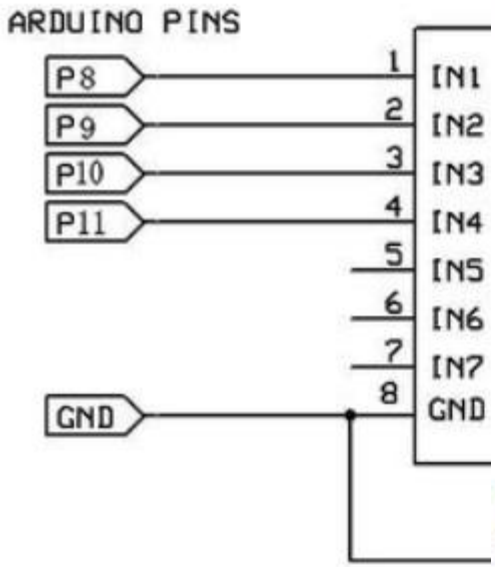


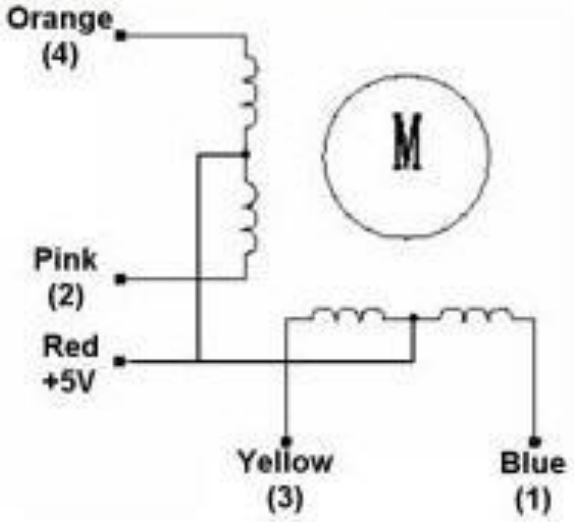
Stepper Motor

- What is so special about steppers
  - A **stepper motor** can move in accurate, fixed angle increments known as steps. For practical purposes, a stepper motor is a bit like a servo: you can tell it to move to a pre-defined position and can count on getting fairly consistent results with multiple repetitions.
  - **Servos** though, are usually **limited to a 0-180** degree range, while a stepper motor can rotate continuously, similar to a regular DC motor. The advantage of steppers over DC motors is that you can achieve much higher precision and control over the movement. The downside of using steppers is that they are a bit more complex to control than servos and DC motors.

Stepper Motor 28BYJ-48

Stepper Motor

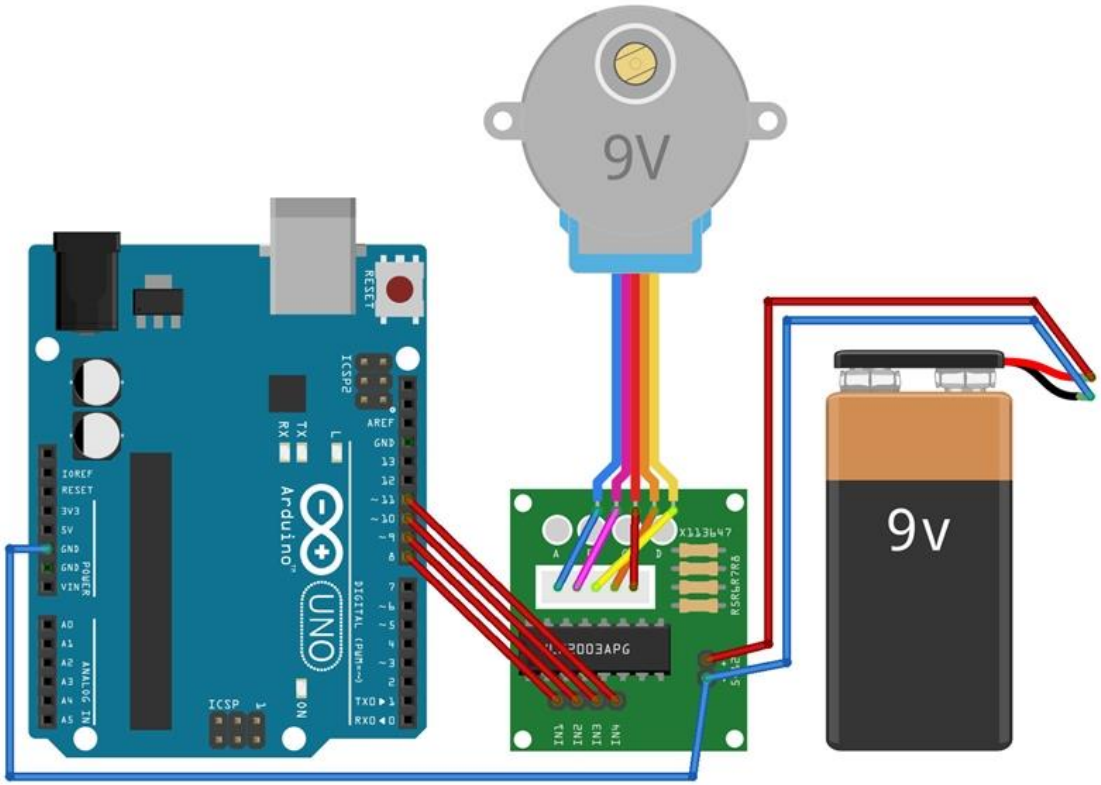




Half-Step Switching Sequence

Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 Orange	-	-						-
3 Yellow		-	-	-				
2 Pink				-	-	-		
1 Blue						-	-	-

Stepper Motor




```
int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;
int delayTime = 10;
void setup() {
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
}
void loop() {
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  delay(delayTime);
}
```

AccelStepper Library


```
#include <AccelStepper.h>
#define HALFSTEP 8

// Motor pin definitions
#define motorPin1  3     // IN1 on the ULN2003 driver 1
#define motorPin2  4     // IN2 on the ULN2003 driver 1
#define motorPin3  5     // IN3 on the ULN2003 driver 1
#define motorPin4  6     // IN4 on the ULN2003 driver 1

// Initialize with pin sequence IN1-IN3-IN2-IN4 for using the AccelStepper with 28BYJ-48
AccelStepper stepper1(HALFSTEP, motorPin1, motorPin3, motorPin2, motorPin4);

void setup() {
  stepper1.setMaxSpeed(1000.0);
  stepper1.setAcceleration(100.0);
  stepper1.setSpeed(200);
  stepper1.moveTo(20000);

}//--(end setup )---

void loop() {

  //Change direction when the stepper reaches the target position
  if (stepper1.distanceToGo() == 0) {
    stepper1.moveTo(-stepper1.currentPosition());
  }
  stepper1.run();
}
```

LCD Monitor

LCD Monitor

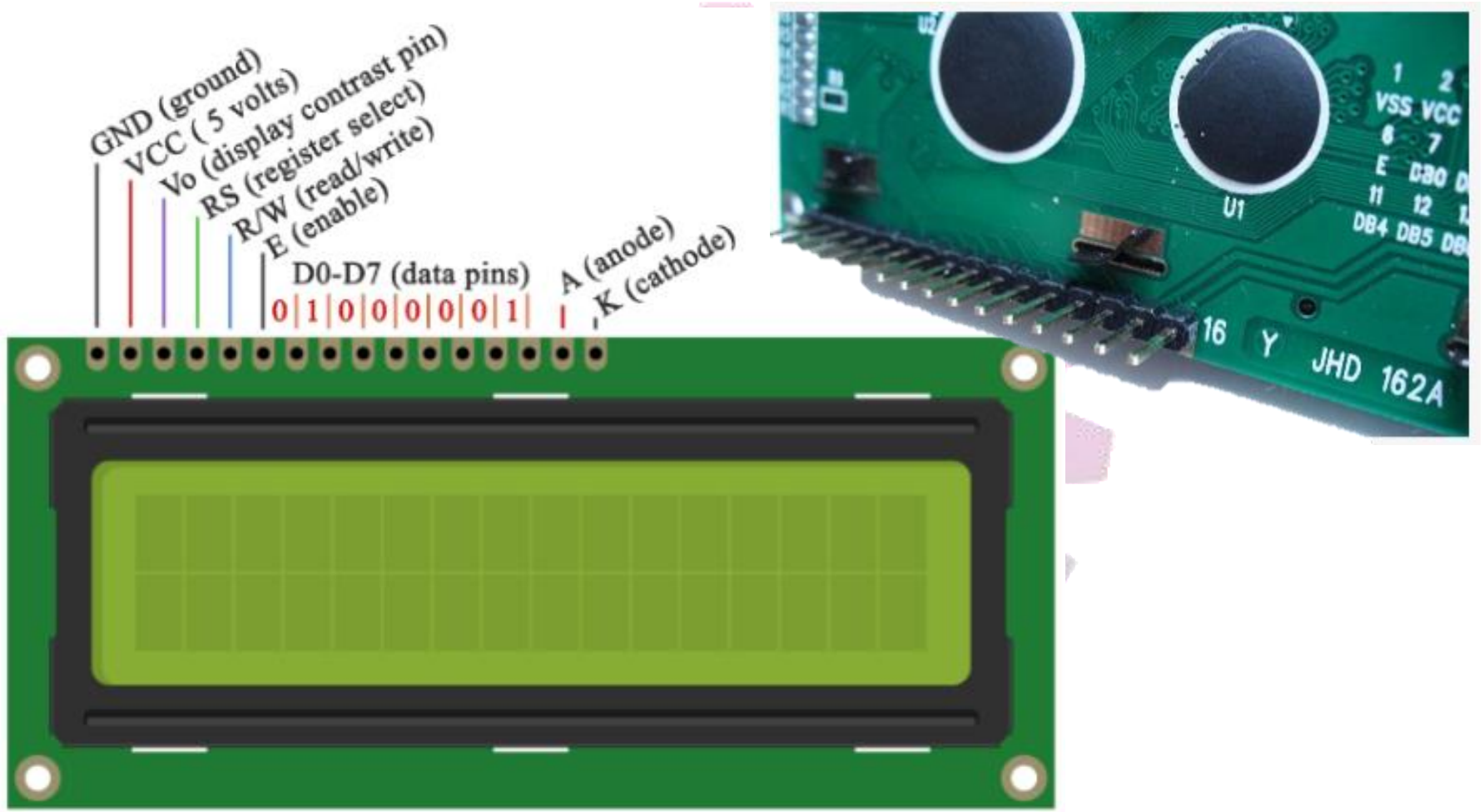
- A thin, flat, electronic visual display that uses the light modulating properties of liquid crystals
- Used in a wide range of applications, including computers, monitors, televisions, instrument panels, etc.
- For interfacing with Arduino system, we will use the HD44780 Character LCD

LCD Monitor

- LCD or Liquid Crystal Display have parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display.

- Interface pins:
  - 1. Register select pin(RS) pin – Controls where the LCD's memory you are writing data to
  - 2. Read/Write (R/W) pin – Select reading mode or writing mode
  - 3. Enable pin - Enables writing to the registers.
  - 4. Data pins (Do to D7) – consist of 8 data pins. The states of these pins (high/low) are the bits that you are writing to a register when you write, or the value s you are reading when you read.
  - 5. Contrast pin (Vo), - control the display contrast
  - 6. Power supply pins (Vcc/GND) – Power the LCD
  - 7. LED Backlight (Bklt+ and BKlt-) – Turns On/Off the LED backlight

LCD Monitor


GND (ground)
VCC (5 volts)
Vo (display contrast pin)
RS (register select)
R/W (read/write)
E (enable)
D0-D7 (data pins)
0 1 0 0 0 0 0 1
A (anode)
K (cathode)
U1
VSS VCC
DB4 DB5
16 Y JHD 162A

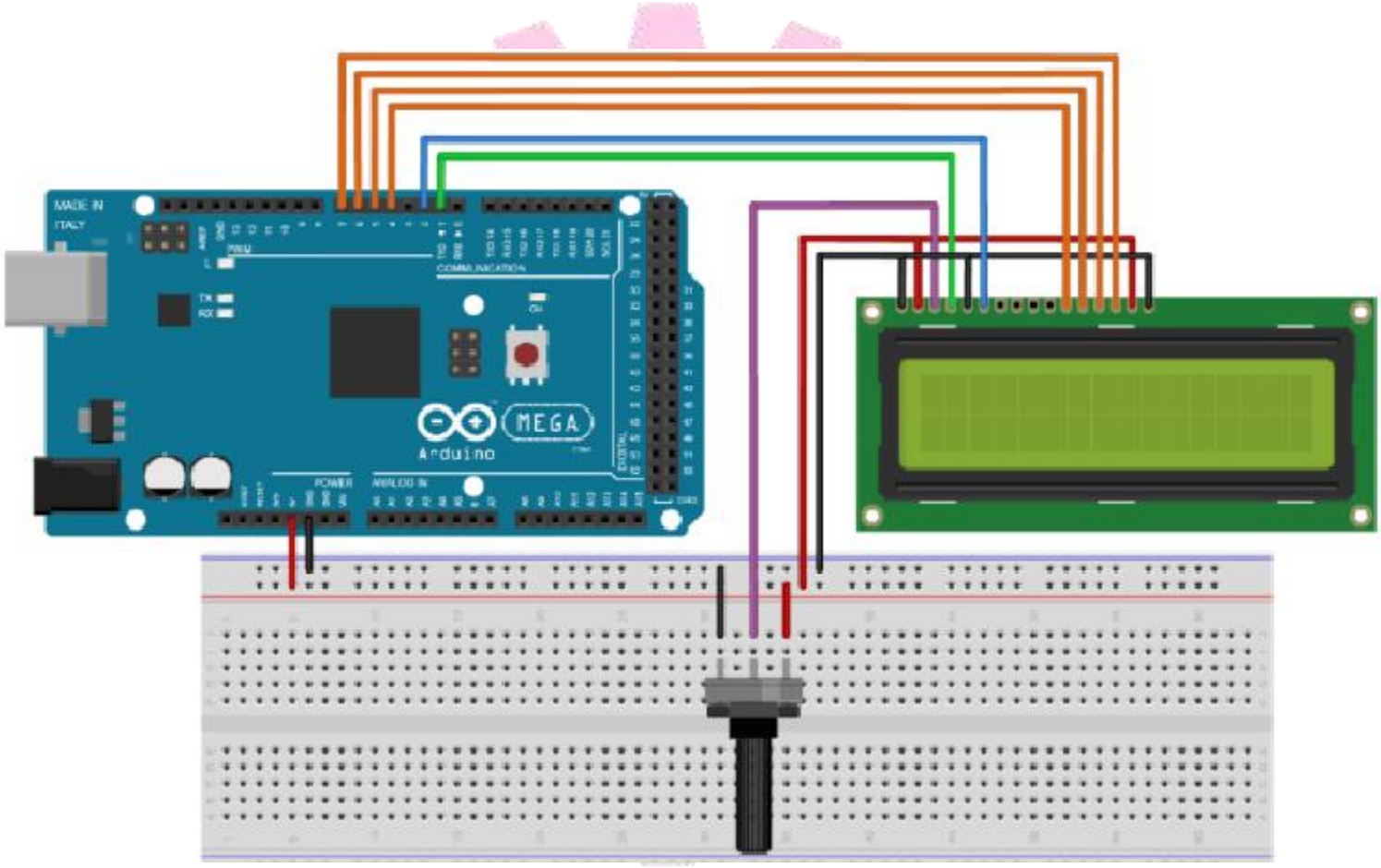
LCD Monitor

- The process of controlling the LCD display involves putting the data from the variable/image that you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystalDisplay Library simplifies the process so it won't be complicated to the user.

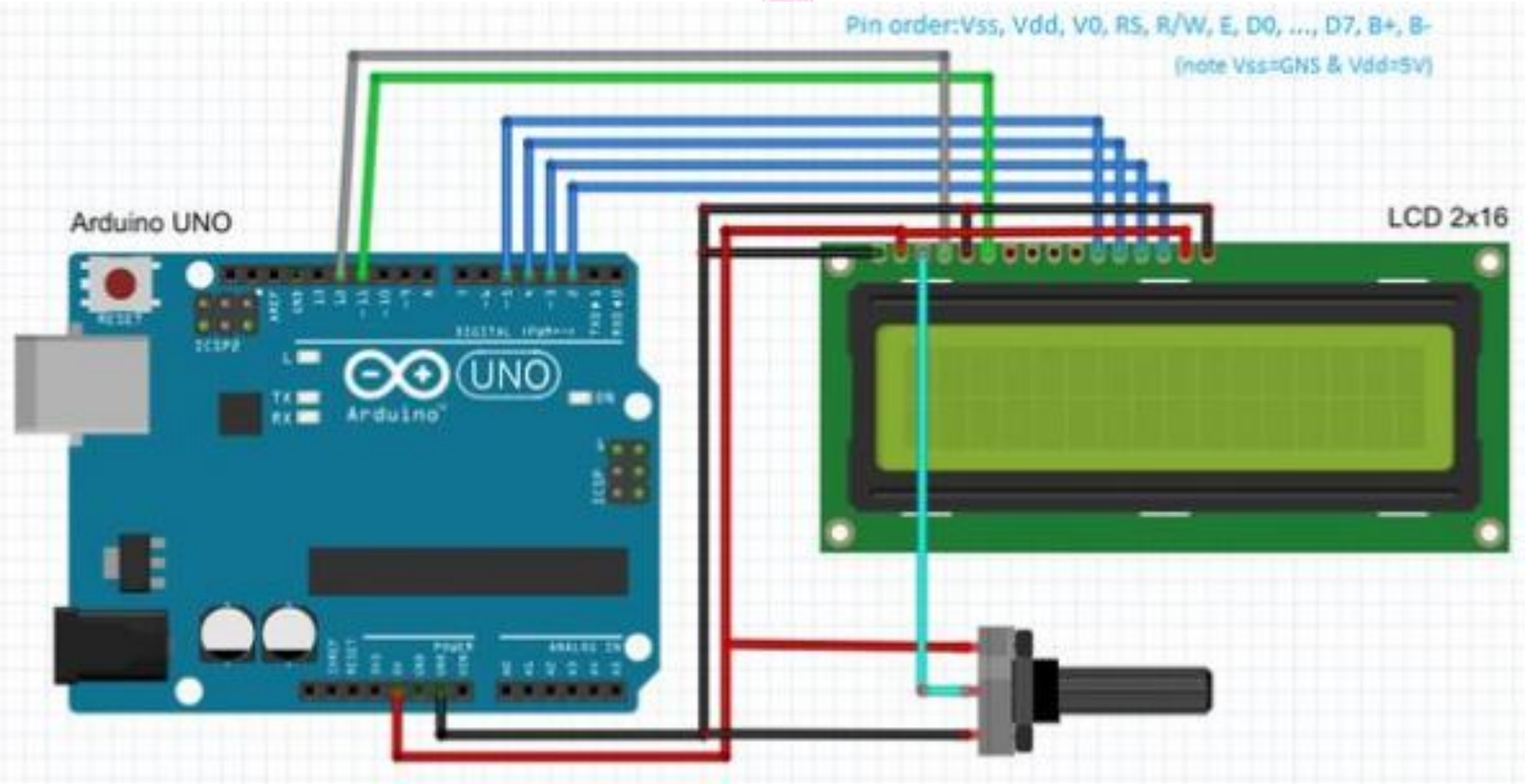
- The Hitachi-compatible LCD's can be controlled in two modes:
  - 4-bit mode requires seven(7) I/O pins from the Arduino
  - 8-bit mode requires eleven (11) I/O pins from the Arduino

LCD Monitor

-



LCD Monitor


Pin order:Vss, Vdd, V0, RS, R/W, E, D0, ..., D7, B+, B-
(note Vss=GNS & Vdd=5V)
Arduino UNO
LCD 2x16
UNO
Arduino

LCD Monitor Functions

- lcd.begin(cols,row) command set up the LCD number of columns and rows. It specifies the dimensions (width and height of the display).
  - Ex. 16x2 LCD Display or 20x4 LCD Display
  - lcd.begin(16,2) or lcd.begin(20,4)

- lcd.print("insert your message here") command print a string that must have a maximum length to lcd column number.
  - Ex. 16 column max length is equal with 16 and 20 column display max length is equal with 20.

- lcd.setCursor(column,line) command that set the position of the cursor's line and column.
  - Column – 0 to 15
  - Line – 0 to 1 (16x2) or 0 to 3 (20x4)
  - lcd.setCursor(0,1); set cursor to second line column 0

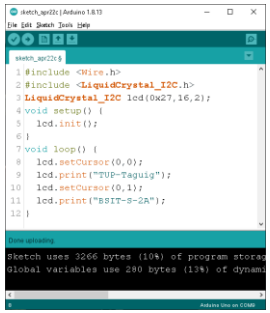
LCD Monitor Functions

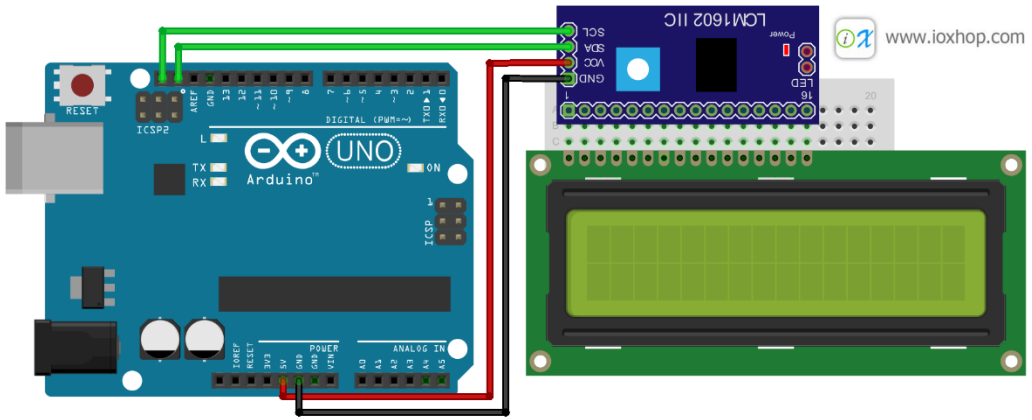
- Has many more specified instructions
  - clear() – clears the whole display
  - home() – places the cursor at the upper left corner (0,0)
  - cursor(), blink(), display() – control device behavior during standby
  - scrollDisplayLeft() and scrollDisplayRight() – moves the display one space to the left/right
  - autoscroll() – automatically scrolls incoming text based on text direction
  - leftToRight() and rightToLeft() – control text direction

LCD Monitor & Arduino UNO

Most common I2C addresses.

- *PCF8574 = 0x20,*
- *PCF8574A = 0x38,*
- *PCF8574AT = 0x3F*
- *PCF8574T = 0x27*





LCD Monitor & Arduino UNO

LCD_i2c_3F_16x2_Analog | Arduino 1.8.5

File Edit Sketch Tools Help

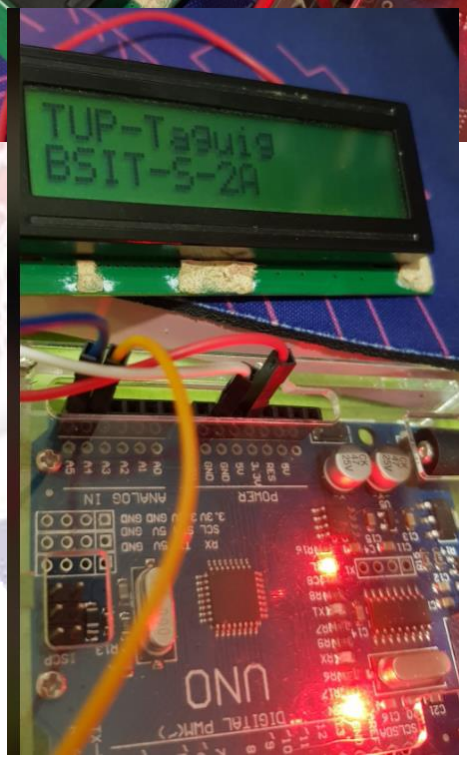
LCD_i2c_3F_16x2_Analog §

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F,16,2);  // set the LCD a

void setup()
{
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Analog Devices");
  lcd.setCursor(5,1);
  lcd.print("BSECE");
}
void loop(){
}
```

Done Saving.

```
The sketch name had to be modified. Sketch names can
of ASCII characters and numbers and be less than 64
```

4 Arduino/Genuino Uno on COM4



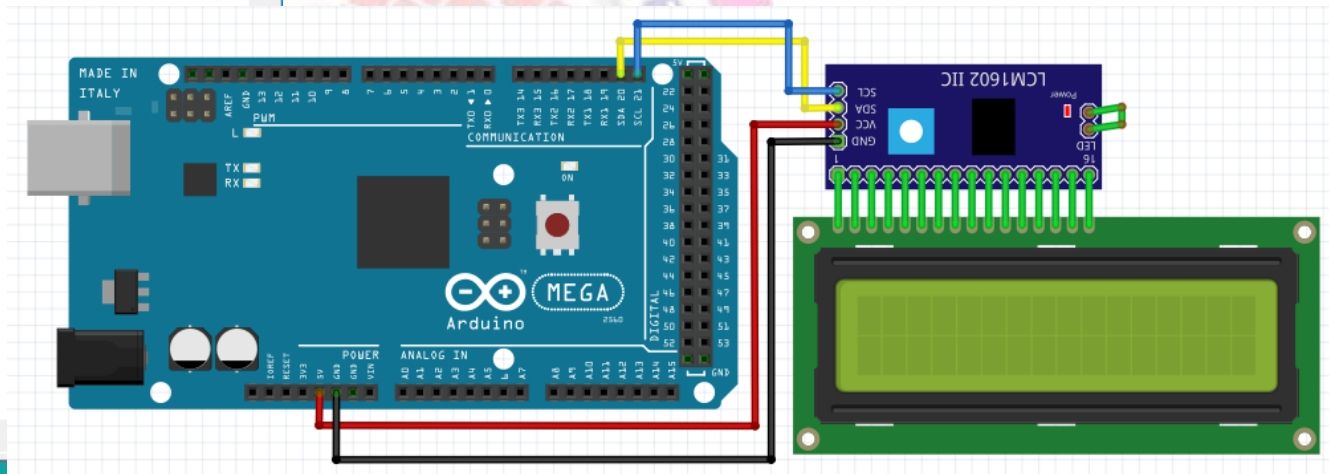
LCD Monitor & Arduino MEGA

LCD_i2c_3F_16x2_Analog | Arduino 1.8.5

File Edit Sketch Tools Help

LCD_i2c_3F_16x2_Analog §

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F,16,2);  // set the LCD a

void setup()
{
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Analog Devices");
  lcd.setCursor(5,1);
  lcd.print("BSECE");
}
void loop(){
}
```



Done Saving.

```
The sketch name had to be modified. Sketch names can
of ASCII characters and numbers and be less than 64
```

4 Arduino/Genuino Uno on COM4

LCD Monitor DEMO

- LCD without using I2C
  - (DF Robot with Keypad)

- LCD using I2C communication
  - I2C scanner
  - 0x3F or 0x27
  - Using A4 & A5 and SDA SCL

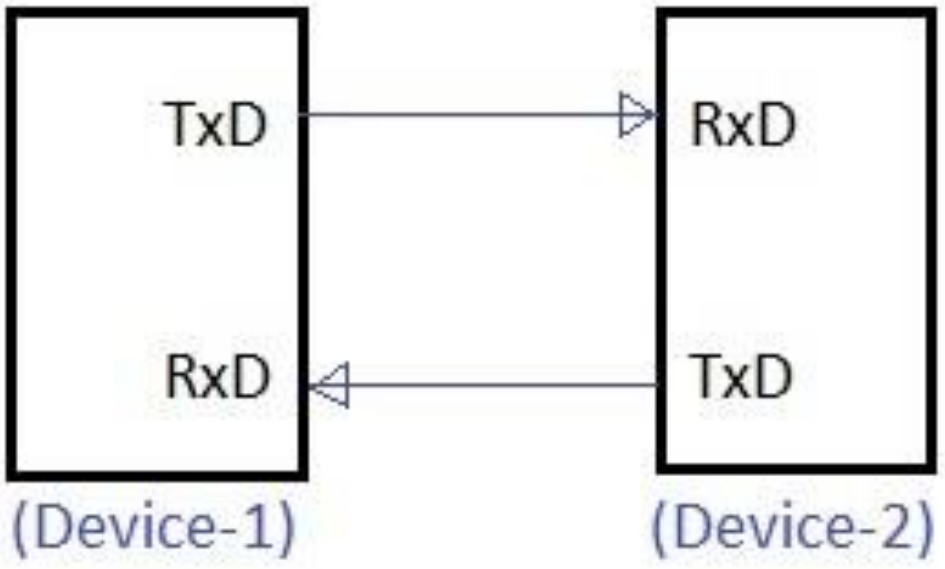
Serial (UART)
I²C (Inter-Integrated Circuit)
SPI (Serial Peripheral Interface)

Serial (UART)

- UART – Universal Synchronous/Asynchronous Receiver/Transmitter
- Used for communication between the Arduino board and a computer or other devices.
- All Arduino boards have at least one serial port (also known as a UART or USART) Serial.
- It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.
- Serial communication on pins TX/RX uses TTL logic levels (5V or 3.3V depending on the board). Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.

UART (Universal Asynchronous Receiver/Transmitter)

- Data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.
- Type of communication
  - Asynchronous
- Distance:
  - Lower about 50 feet
- Pins
  - TxD: Transmit Data
  - RxD: Receive Data



UART Interface Diagram

UART (Universal Asynchronous Receiver/Transmitter)

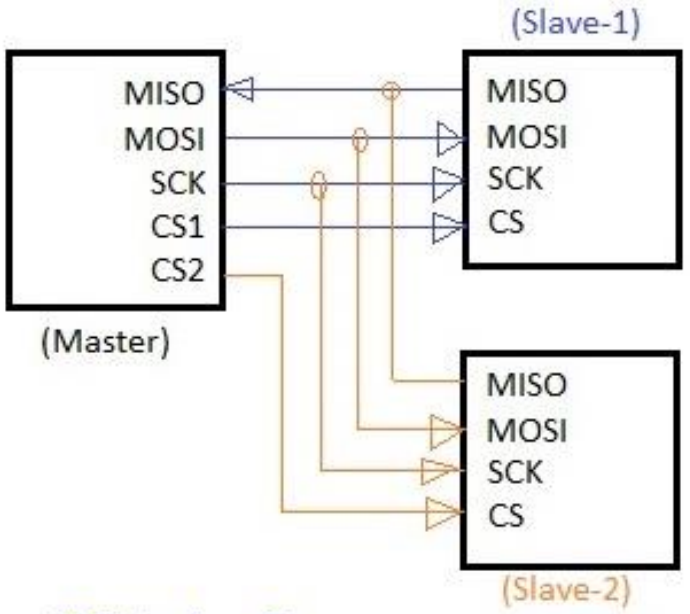
Advantage	Disadvantage
• It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface.	• They are suitable for communication between only two devices. • It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled.

Software Serial

- To be able to upload code and access the Serial Monitor, the UART lines (pins 0 and 1) must be dedicated to the Arduino ONLY!
- SoftwareSerial Library enables any digital pin to become a UART line
- ISSUES
  - If running multiple lines, only ONE can receive data at a time
  - Additional burden in memory!

Serial Peripheral Interface (SPI)

- Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps.
- Type of communication
  - Synchronous
- Distance:
  - Highest
- Pins
  - SCLK: Serial Clock
    MOSI: Master Output, Slave Input
    MISO: Master Input, Slave Output
    CS/SS: Slave Select



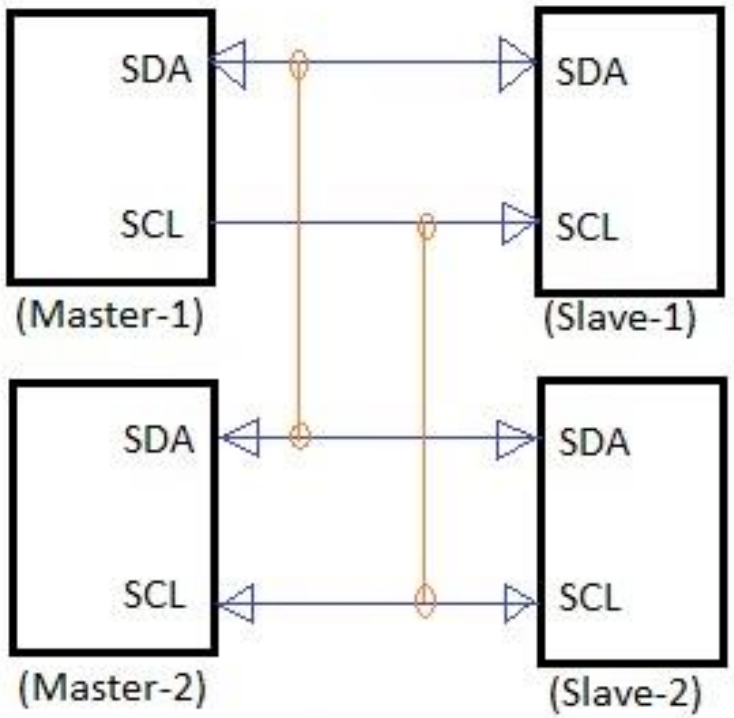
SPI Interface Diagram

Serial Peripheral Interface (SPI)

Advantage	Disadvantage
• It is simple protocol and hence so not require processing overheads. • Supports full duplex communication. • Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. • SPI uses push-pull and hence higher data rates and longer ranges are possible. • SPI uses less power compare to I2C	• As number of slave increases, number of CS lines increases, this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS line and changes in software for particular device addressing is concerned. • Master and slave relationship can not be changed as usually done in I2C interface. • No flow control available in SPI.

Inter-Integrated Circuit (I²C)

- I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also supports 10 Kbps and 1 Mbps.
- Type of communication
  - Synchronous
- Distance:
  - Higher
- Pins
  - SDA: Serial Data
    SCL: Serial Clock



I2C Interface Diagram

Inter-Integrated Circuit (I²C)

Advantage	Disadvantage
• Due to open collector design, limited slew rates can be achieved. • More than one masters can be used in the electronic circuit design. • Needs fewer i.e. only 2 wires for communication. • I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. • It uses open collector bus concept. Hence there is bus voltage flexibility on the interface bus. • Uses flow control.	• Increases complexity of the circuit when number of slaves and masters increases. • I2C interface is half duplex. • Requires software stack to control the protocol and hence it needs some processing overheads on microcontroller/microprocessor.

Sensors

What Are Sensors?

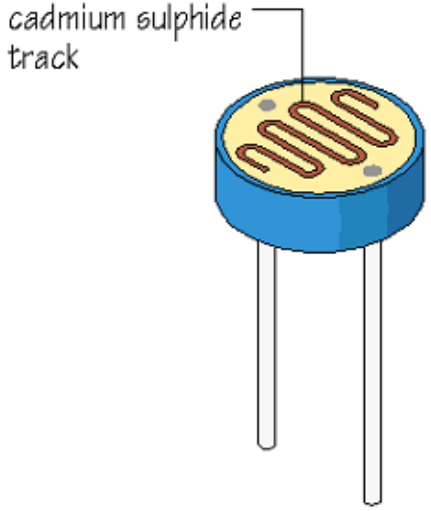
- Electronic device that responds to an external stimulus and gives off electronic signals with respect to said stimulus.
- Can be grouped into 2 categories
  - Go or No-Go Sensors - produce a binary (digital) output, like an on or off switch.
  - Analog Sensors - produce an output signal that is proportional to its input stimulus.

Sensor Concepts

- Conditioning Circuit
  - additional circuitry added between the sensor and processor to modify the sensor signal.
- Interfacing Circuit
  - additional circuitry needed between the sensor and processor to access the sensor signal.

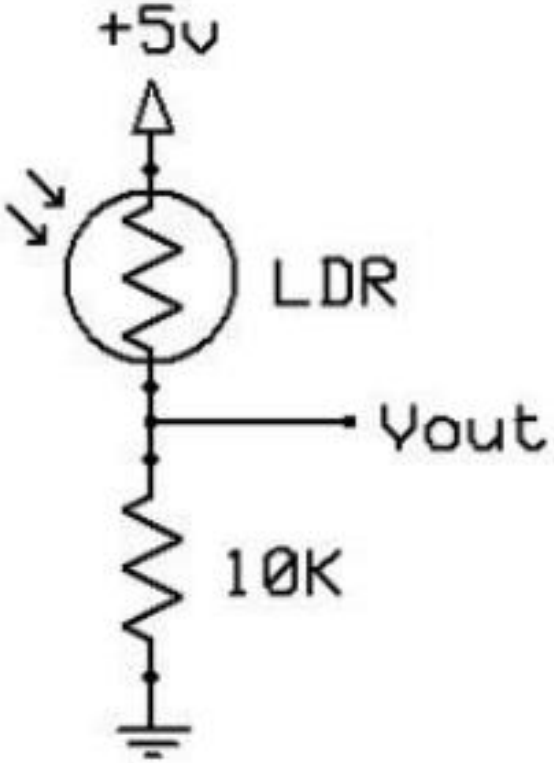
LDR

- Light Dependent Resistor (LDR)
  - A type of variable resistor whose resistance value decreases with increasing light intensity.
- Semiconductor material reacts with light, allowing the device to conduct electricity and effectively reduce its resistance.
- Use it as a resistance element, the varying resistance allows it to output different voltage values.



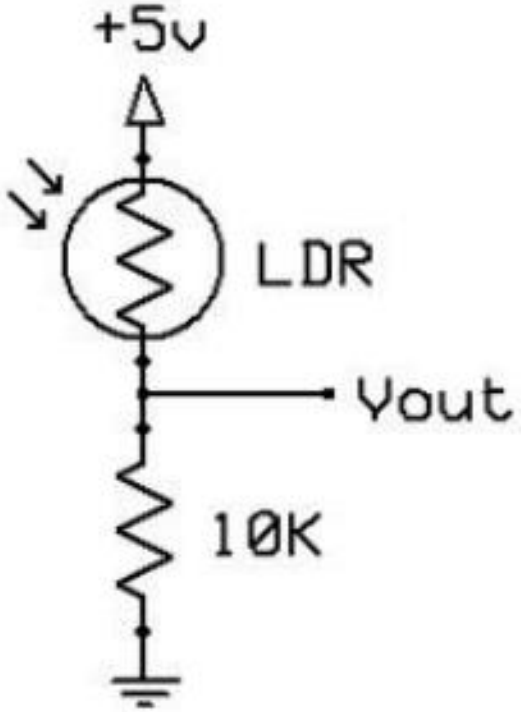
LDR

- Follow the wiring diagram on your right, then upload ADC sketch on your Arduino.
- The circuit used is a simple voltage divider circuit, the principles of which are similar with a potentiometer.



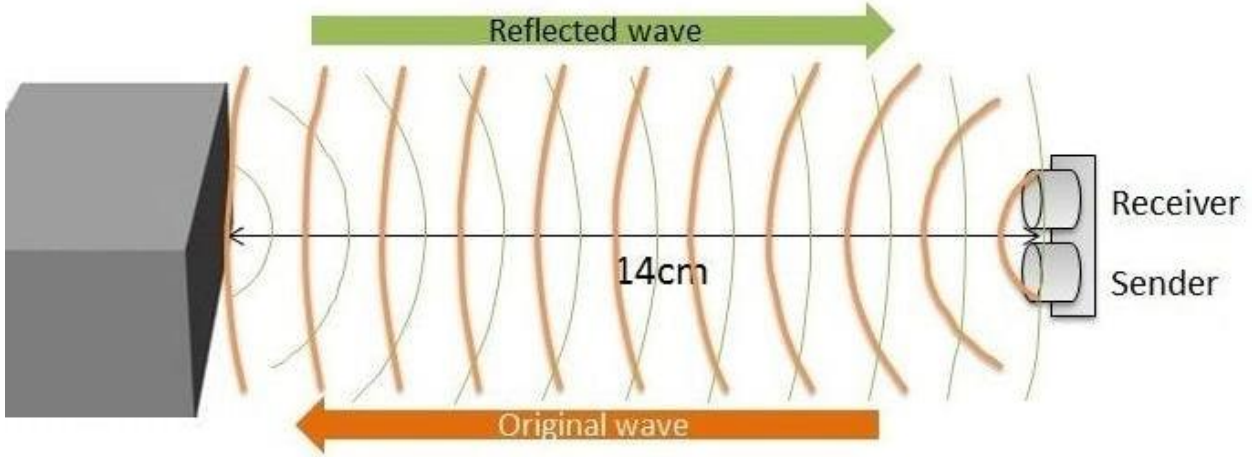
LDR control Servo

- With the LDR input, make an LED light up when the LDR senses darkness, and turn off LED when LDR senses light.
- Next, make the LED adjust linearly with the LDR input, dim the LED with respect to the light intensity the LDR senses.



Ultrasonic Sensor

- The use of sound/vibrations as the active material in sensing.
- Ultrasonic refers to the frequency of sound that is above the human audible range (40kHz)
- PROS: non-intrusive, no need for physical contact, able to detect objects not applicable for vision/optics based sensors.
- CONS: very sensitive to temperature and position of operation (angles).

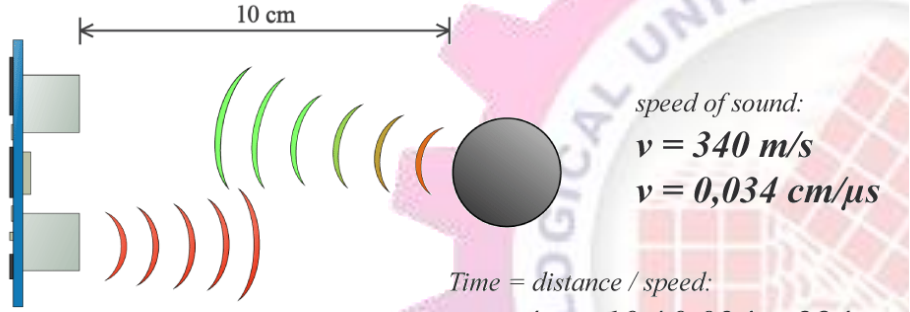


Ultrasonic Sonar

- HC-SR04 is a commonly used module for non-contact distance measurement for distances from 2cm to 400cm.
- It uses sonar (like bats and dolphins) to measure distance with high accuracy and stable readings.
- It consists of an ultrasonic transmitter, receiver and control circuit.
- The transmitter transmits short bursts which gets reflected by target and are picked up by the receiver. The time difference between transmission and reception of ultrasonic signals is calculated.
- Using the speed of sound and '**Speed = Distance/Time**' equation, the distance between the source and target can be easily calculated.

Ultrasonic Sensor

- VCC connection of the sensor attached to +5V
- GND connection of the sensor attached to ground
- TRIG connection of the sensor attached to digital pin 2
- ECHO connection of the sensor attached to digital pin 4



speed of sound:

$v = 340\ m/s$

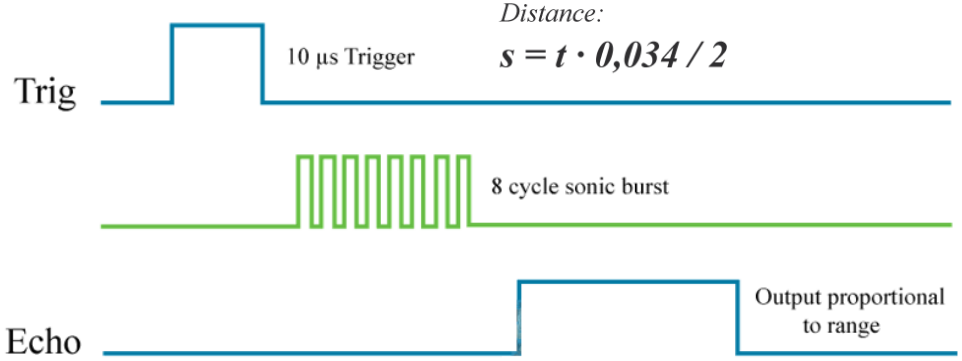
$v = 0{,}034\ cm/\mu s$

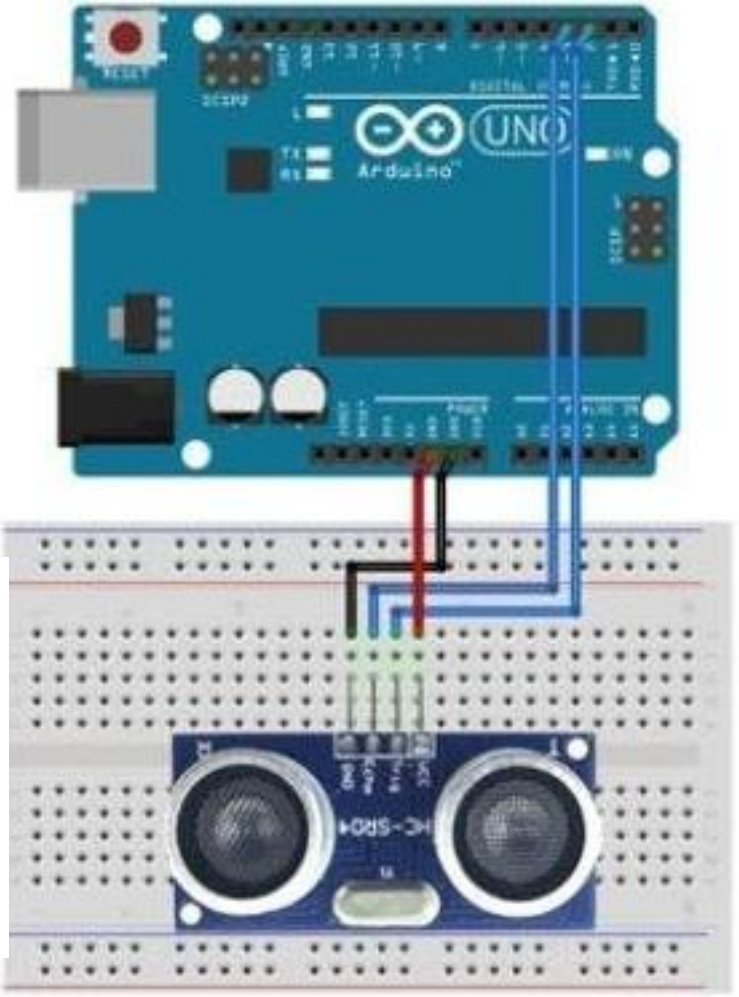
Time = distance / speed:

$t = s / v = 10 / 0{,}034 = 294\ \mu s$

Distance:

$s = t \cdot 0{,}034 / 2$





Connection and Code

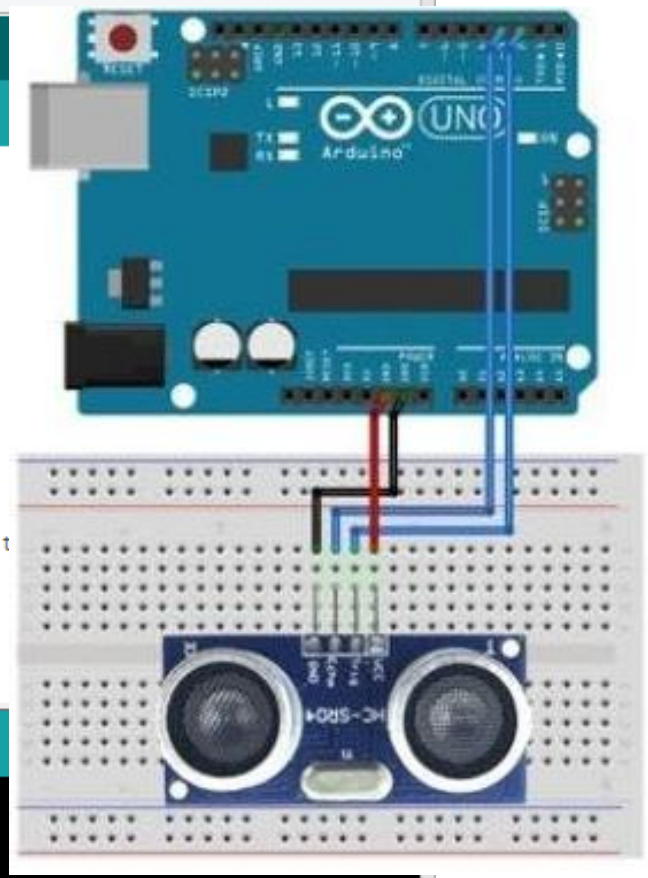
sketch_sep25a | Arduino 1.8.5

File Edit Sketch Tools Help

sketch_sep25a §

```
void setup() {
  pinMode(9, OUTPUT); // Sets the trigPin as an Output
  pinMode(10, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}
void loop() {
  digitalWrite(9, LOW); // Clears the trigPin for 2 microseconds
  delayMicroseconds(2);
  digitalWrite(9, HIGH); // Sets the trigPin on HIGH state for 10 microseconds
  delayMicroseconds(10);
  digitalWrite(9, LOW);
  long duration = pulseIn(10, HIGH); // Reads the echoPin, returns the sound wave t
  int distance= duration*0.034/2; // Calculating the distance
  Serial.print("Distance: ");
  Serial.println(distance);
}
```

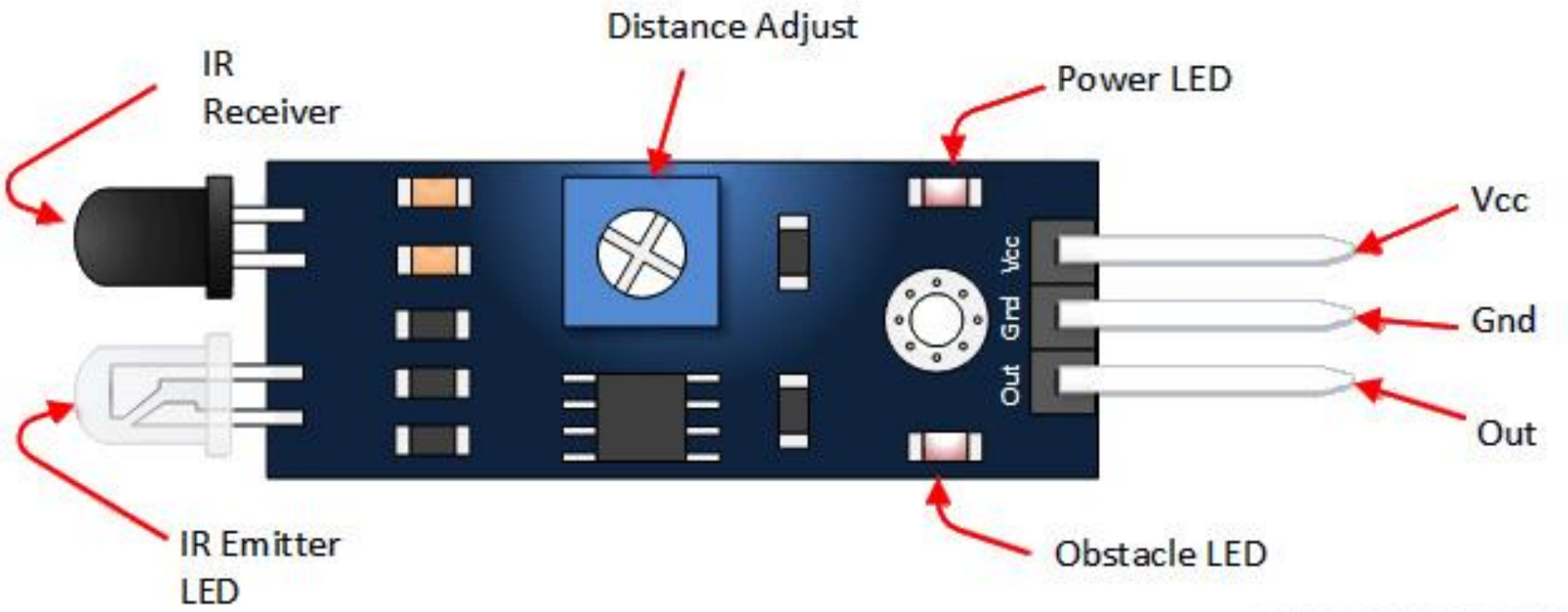
1 Arduino/Genuino Uno on COM1



Ultrasonic w/ LCD (parallel)


```
#include <LiquidCrystal.h> // includes t
LiquidCrystal lcd(1, 2, 4, 5, 6, 7); //
void setup() {
  lcd.begin(16,2); // Initializes the in
  pinMode(9, OUTPUT);
  pinMode(10, INPUT);
}
void loop() {
  digitalWrite(9, LOW);
  delayMicroseconds(2);
  digitalWrite(9, HIGH);
  delayMicroseconds(10);
  digitalWrite(9, LOW);
  long duration = pulseIn(10, HIGH);
  int distanceCm= duration*0.034/2;
  int distanceInch = duration*0.0133/2;
  lcd.setCursor(0,0); // Sets the locati
  lcd.print("Distance: "); // Prints str
  lcd.print(distanceCm); // Prints the d
  lcd.print(" cm");
  delay(10);
  lcd.setCursor(0,1);
  lcd.print("Distance: ");
  lcd.print(distanceInch);
  lcd.print(" inch");
  delay(10);
}
```

Proximity Sensing



MQ Series

MQ-2
Smoke Gas

MQ-3
Alcohol

MQ-4
Methane gas

MQ-5
Methane Natural Gas

MQ-6
LPG Gas

MQ-7
Carbon monoxide gas

MQ-8
Hydrogen

MQ-9
Combustible gas

MQ-135
Air Quality

Gas Sensor Comparison

Sensor	Detects	Heater Voltage
MQ-2	Methane, Butane, LPG, smoke	5V
MQ-3	Alcohol, Ethanol, smoke	5V
MQ-4	Methane, CNG Gas	5V
MQ-5	Natural gas, LPG	5V
MQ-6	LPG, butane gas	5V
MQ-7	Carbon Monoxide	Alternating 5V and 1.4V
MQ-8	Hydrogen Gas	5V
MQ-9	Carbon Monoxide, flammable gasses.	Alternating 5V and 1.4V
MQ131	Ozone	6V
MQ135	Air Quality (Benzene, Alcohol, smoke)	5V
MQ136	Hydrogen Sulfide gas	5V
MQ137	Ammonia	5V
MQ138	Benzene, Toluene, Alcohol, Acetone, Propane, Formaldehyde gas, Hydrogen	5V
MQ214	Methane, Natural gas	6V
MQ216	Natural gas, Coal gas	5V
MQ303A	Alcohol, Ethanol, smoke	0.9V
MQ306A	LPG, butane gas	0.9V
MQ307A	Carbon Monoxide	Alternating 0.2V and 0.9V
MQ309A	Carbon Monoxide, flammable gasses	Alternating 0.2V and 0.9V
MG811	Carbon Dioxide (CO2)	6V
AQ-104	Air quality	-

MQ-XX gas sensor Breakout board

Optical Dust Sensor

- Sharp's GP2Y1010AU0F is an optical air quality sensor, designed to sense dust particles. An infrared emitting diode and a phototransistor are diagonally arranged into this device, to allow it to detect the reflected light of dust in air. It is especially effective in detecting very fine particles like cigarette smoke, and is commonly used in air purifier systems.

- The sensor has a very low current consumption (20mA max, 11mA typical), and can be powered with up to 7VDC. The output of the sensor is an analog voltage proportional to the measured dust density, with a sensitivity of 0.5V/0.1mg/m3.

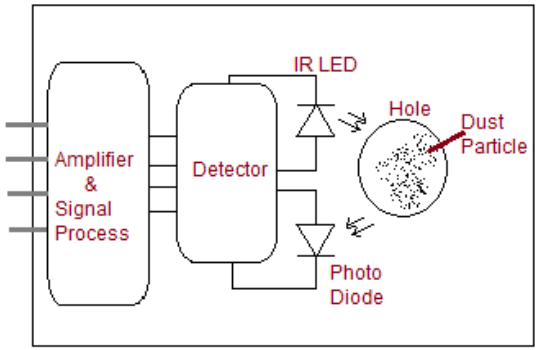
Optical Dust Sensor
(GP2Y1010AU0F)

- This dust sensor is compact in size and detects dust in the air and also smoke particles. It consumes low current to detect dust and uses photometry method to detect dust level in the Air.

Optical Dust Sensor - GP2Y1010AU0F

1 --> V-LED
2 --> LED-GND
3 --> LED
4 --> S-GND
5 --> Vo
6 --> Vcc

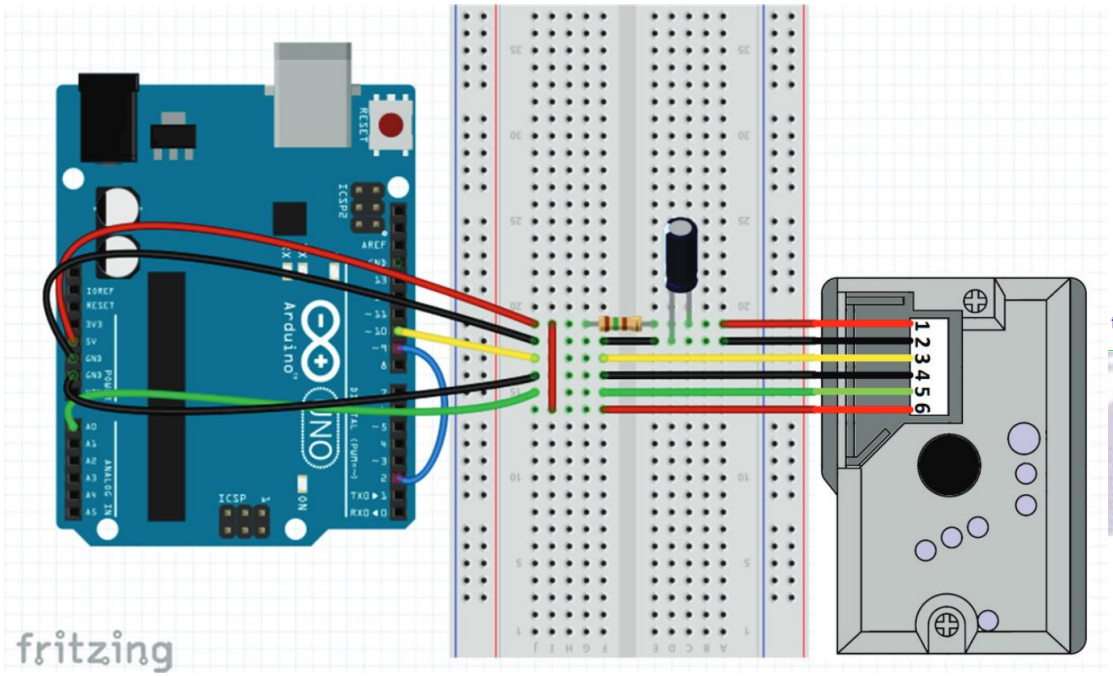
How Dust Sensor Works

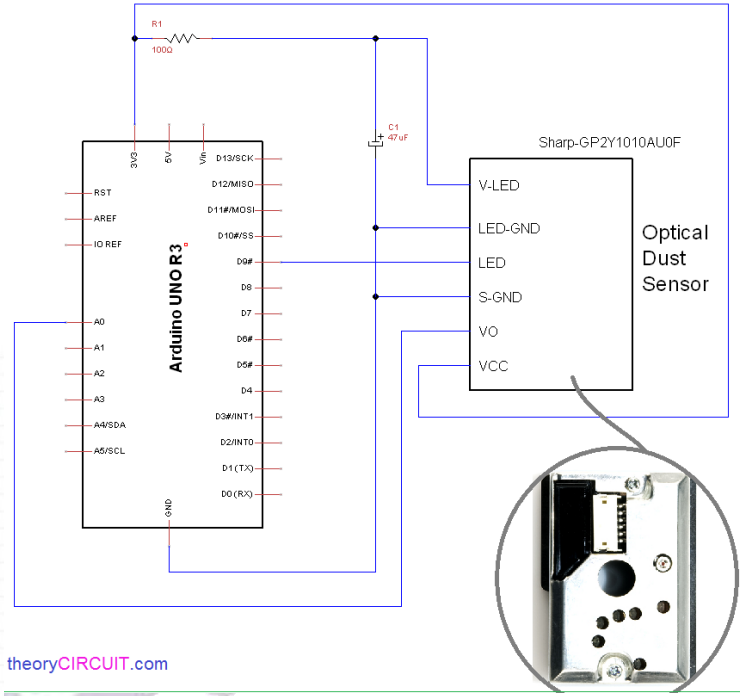



```
int sensePin = A0;
int ledPin = 9;
int Tsampling = 280;
int Tdelta = 40;
int Tsleep = 9680;
float outVo = 0;
float sigVolt = 0;
float dustLevel = 0;
void setup(){
  Serial.begin(9600);
  pinMode(ledPin,OUTPUT);
}
void loop(){
  digitalWrite(ledPin,LOW);
  delayMicroseconds(Tsampling);
  outVo = analogRead(sensePin);
  delayMicroseconds(Tdelta);
  digitalWrite(ledPin,HIGH);
  delayMicroseconds(Tsleep);
  sigVolt = outVo * (3.3 / 1024);
  dustLevel = 0.17 * sigVolt - 0.1;
  Serial.print("Measured Signal Value (0-1023): ");
  Serial.print(outVo);
  Serial.print("   Voltage: ");
  Serial.print(sigVolt);
  Serial.print("   Dust Density Level: ");
  Serial.println(dustLevel);
  delay(1000);
}
```

Optical Dust Sensor

Sharp pin 1 (V-LED) => 5V (connected to 150ohm resister)
Sharp pin 2 (LED-GND) => Arduino GND pin
Sharp pin 3 (LED) => Arduino pin 2
Sharp pin 4 (S-GND) => Arduino GND pin
Sharp pin 5 (Vo) => Arduino Ao pin
Sharp pin 6 (Vcc) => 5V





RFID (Radio Frequency IDentification)

- RFID stands for Radio Frequency IDentification and it's a non-contact technology that's broadly used in many industries for tasks such as personnel tracking, access control, supply chain management, books tracking in libraries, tollgate systems and so on.

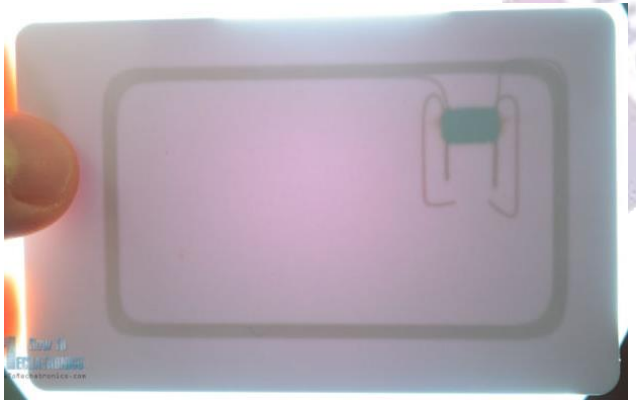
- How RFID Works
  - An RFID system consists of two main components, a transponder or a tag which is located on the object that we want to be identified, and a transceiver or a reader.

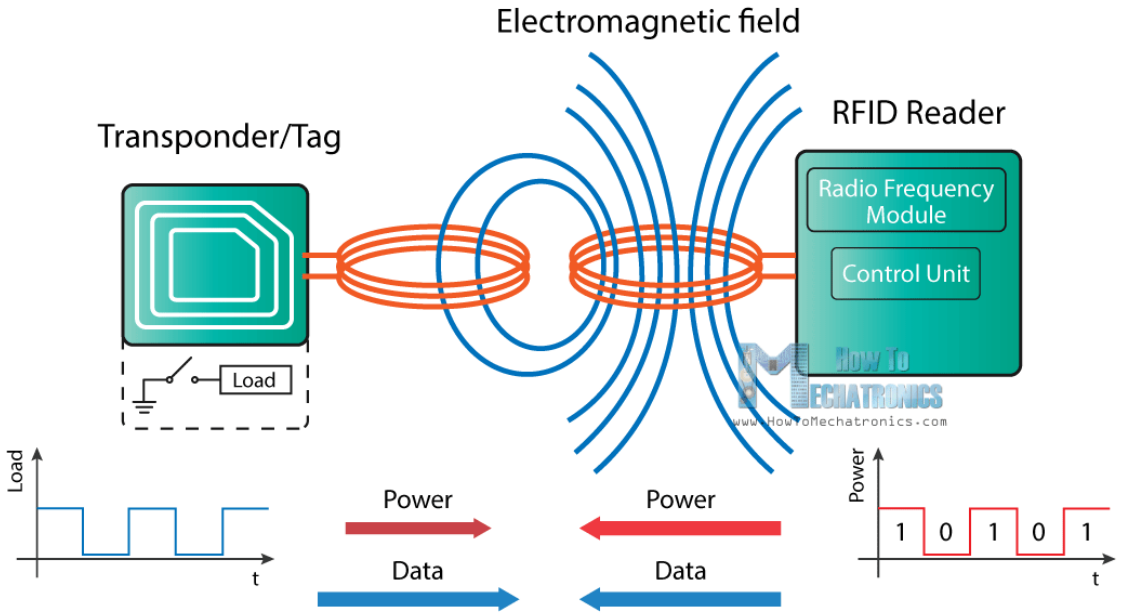
RFID Reader

- The RFID reader consist of a radio frequency module, a control unit and an antenna coil which generates high frequency electromagnetic field.
- The tag is usually a passive component, which consist of just an antenna and an electronic microchip, when it gets near the electromagnetic field of the transceiver, due to induction, a voltage is generated in its antenna coil and this voltage serves as power for the microchip.

Type of reading:

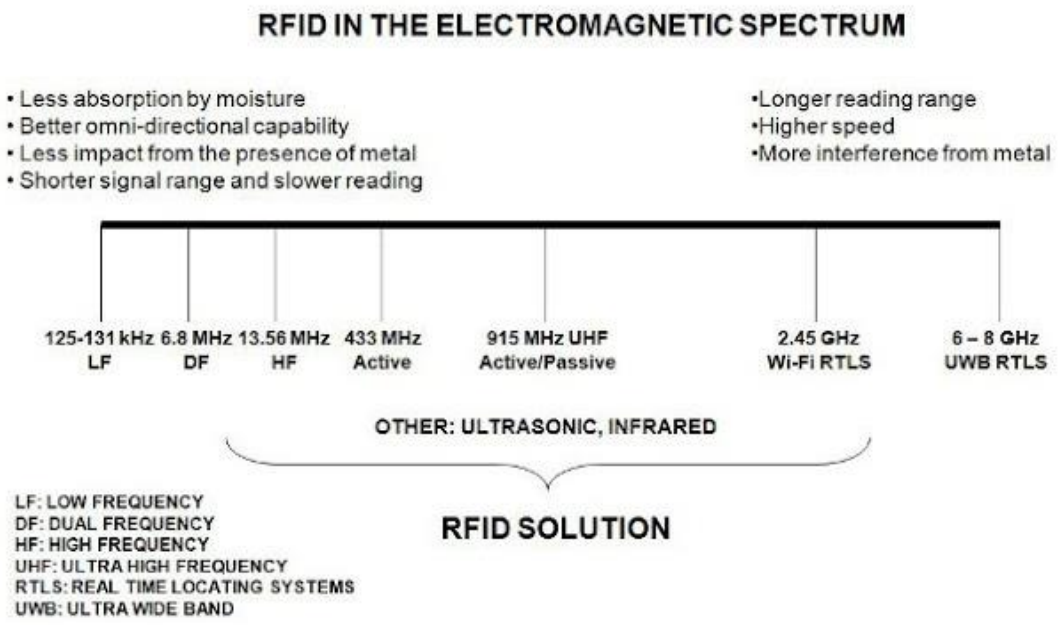
1. Load manipulation
2. Backscattered coupling





RFID (LF, HF, and UHF)

- RFID tags and readers have to be tuned to the same frequency in order to communicate.
- There are several different frequencies an RFID system can use.
- Generally, the most common are
  - Low frequency, or LF, (125 – 134 kHz)
  - High frequency, or HF, (13.56 MHz)
  - Ultra-high frequency, or UHF, (433, and 860-960 MHz)



Specification and Pin Configuration

- Specifications
  - Input voltage: 3.3V
  - Frequency: 13.56MHz

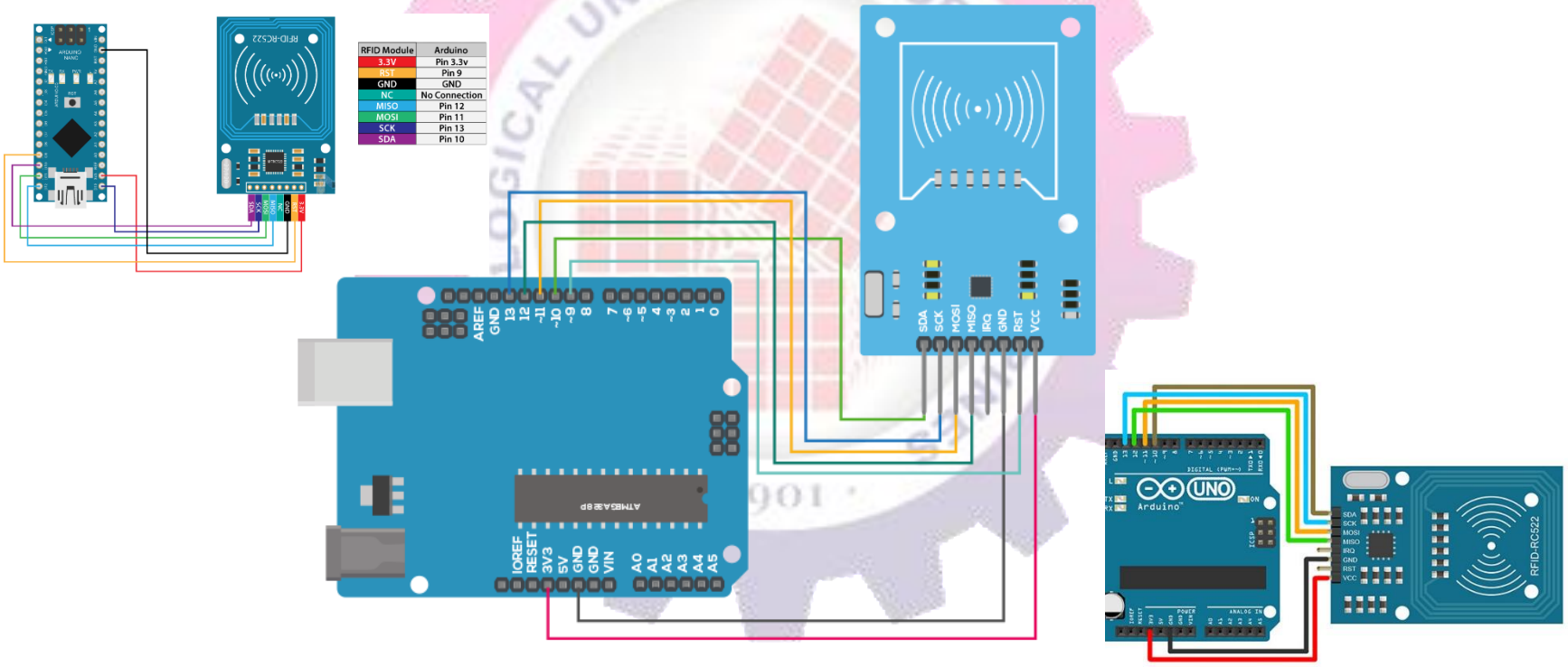
Pin	Wiring to Arduino Uno
SDA	Digital 10
SCK	Digital 13
MOSI	Digital 11
MISO	Digital 12
IRQ	unconnected
GND	GND
RST	Digital 9
3.3V	3.3V

Library download

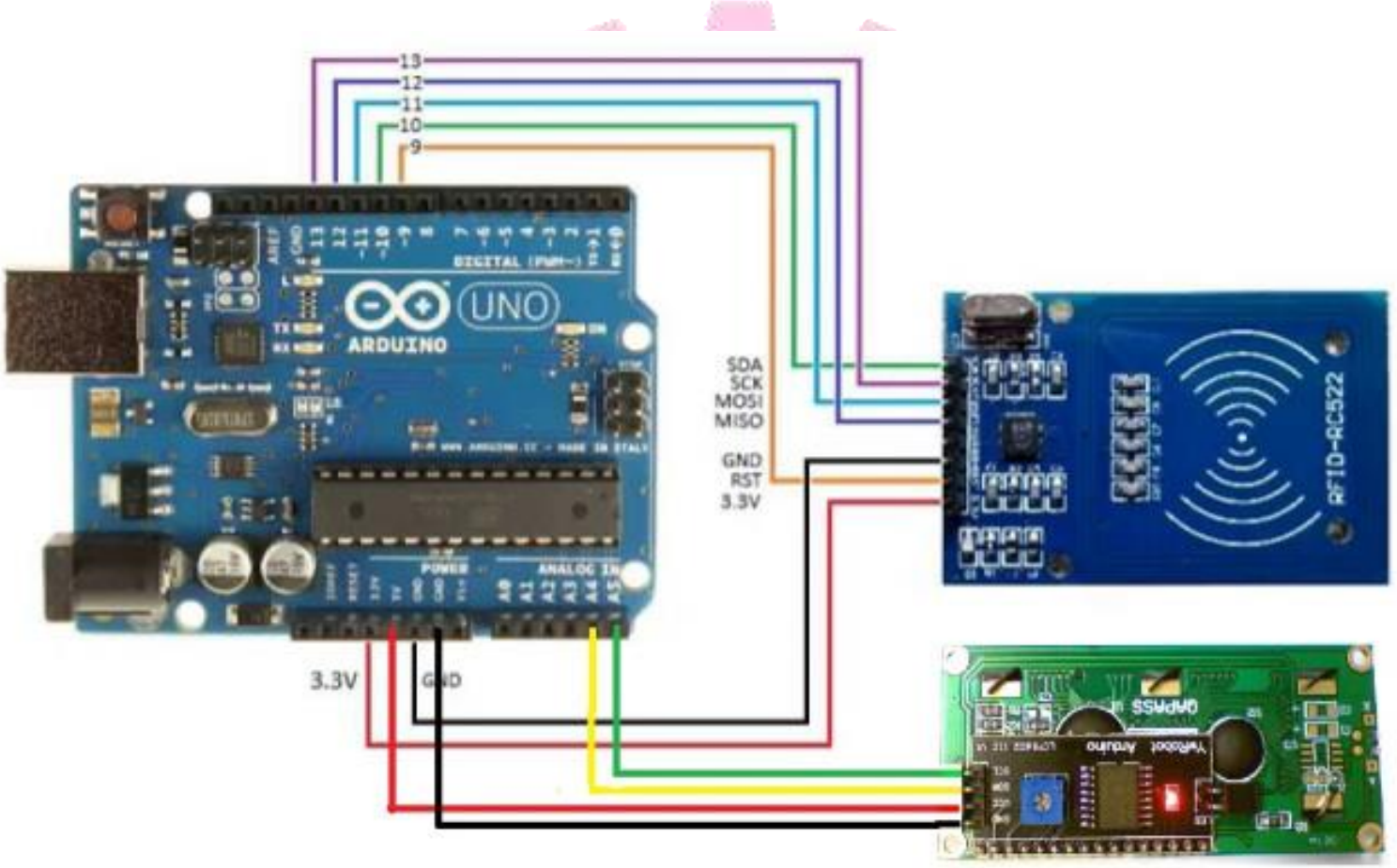
- Download the RFID library
- Unzip the RFID library
- Install the RFID library in your Arduino IDE
- Restart your Arduino IDE

RFID Reader/Card/Tag

- **IMPORTANT: Don't forget that the module is powered with 3.3v from Arduino you should use a level shifter (5v to 3.3v) to avoid damaging the component over time.**



RFID and LCD


13
12
11
10
9
SDA
SCK
MOSI
MISO
GND
RST
3.3V
3.3V
GND
RFID-RC522
ARDUINO
UNO

Internet of Things (IoT)

IoT Overview

- IoT systems allow users to achieve deeper automation, analysis, and integration within a system. They improve the reach of these areas and their accuracy. IoT utilizes existing and emerging technology for sensing, networking, and robotics.

- IoT exploits recent advances in software, falling hardware prices, and modern attitudes towards technology. Its new and advanced elements bring major changes in the delivery of products, goods, and services; and the social, economic, and political impact of those changes.

IoT – Key Features

- The most important features of IoT include artificial intelligence, connectivity, sensors, active engagement, and small device use. A brief review of these features is given below:

  - AI
  - Connectivity
  - Sensors
  - Active Engagement
  - Small Devices

IoT – Key Features

- AI – IoT essentially makes virtually anything "smart", meaning it enhances every aspect of life with the power of data collection, artificial intelligence algorithms, and networks. This can mean something as simple as enhancing your refrigerator and cabinets to detect when milk and your favorite cereal run low, and to then place an order with your preferred grocer.

- Connectivity – New enabling technologies for networking, and specifically IoT networking, mean networks are no longer exclusively tied to major providers. Networks can exist on a much smaller and cheaper scale while still being practical. IoT creates these small networks between its system devices.

IoT – Key Features

- Sensors – IoT loses its distinction without sensors. They act as defining instruments which transform IoT from a standard passive network of devices into an active system capable of real-world integration.
- Active Engagement – Much of today's interaction with connected technology happens through passive engagement. IoT introduces a new paradigm for active content, product, or service engagement.
- Small Devices – Devices, as predicted, have become smaller, cheaper, and more powerful over time. IoT exploits purpose-built small devices to deliver its precision, scalability, and versatility.

IoT – Advantages

- The advantages of IoT span across every area of lifestyle and business. Here is a list of some of the advantages that IoT has to offer:
  - Improved Customer Engagement
  - Technology Optimization
  - Reduced Waste
  - Enhanced Data Collection

IoT – Disadvantages

- Though IoT delivers an impressive set of benefits, it also presents a significant set of challenges. Here is a list of some its major issues:
  - Security
  - Privacy
  - Complexity
  - Flexibility
  - Compliance

IoT – Disadvantages

- **Security** – IoT creates an ecosystem of constantly connected devices communicating over networks. The system offers little control despite any security measures. This leaves users exposed to various kinds of attackers.
- **Privacy** – The sophistication of IoT provides substantial personal data in extreme detail without the user's active participation.
- **Complexity** – IoT systems are complicated in terms of design, deployment, and maintenance given their use of multiple technologies and a large set of new enabling technologies.
- **Flexibility** – Many are concerned about the flexibility of an IoT system to integrate easily with another. They worry about finding themselves with several conflicting or locked systems.
- **Compliance** – IoT, like any other technology in the realm of business, must comply with regulations. Its complexity makes the issue of compliance seem incredibly challenging when many consider standard software compliance a battle.

IoT – Hardware

- IoT – Sensors
  - The sensing module manages sensing through assorted active and passive measurement devices. Here is a list of some of the measurement devices used in IoT:

Devices	
accelerometers	temperature sensors
magnetometers	proximity sensors
gyroscopes	image sensors
acoustic sensors	light sensors
pressure sensors	gas sensors
humidity sensors	RFID sensors

Wearable Electronics

- Wearable electronic devices are small devices worn on the head, neck, arms, torso, and feet.

- Current smart wearable devices include:
  - Head – Helmets, glasses
  - Neck – Jewelry, collars
  - Arm – Watches, wristbands, rings
  - Torso – Clothing, backpacks
  - Feet – Socks, shoes

Wearable Electronics

- Smart watches not only help us stay connected, but as a part of an IoT system, they allow access needed for improved productivity.

Wearable Electronics

- Smart glasses help us enjoy more of the media and services we value, and when part of an IoT system, they allow a new approach to productivity.

Standard Devices

- The desktop, tablet, and cellphone remain integral parts of IoT as the command center and remotes.
  - The desktop provides the user with the highest level of control over the system and its settings.
  - The tablet provides access to the key features of the system in a way resembling the desktop, and also acts as a remote.
  - The cellphone allows some essential settings modification and also provides remote functionality.
- Other key connected devices include standard network devices like routers and switches.

IoT – Software

- IoT software addresses its key areas of networking and action through platforms, embedded systems, partner systems, and middleware. These individual and master applications are responsible for data collection, device integration, real-time analytics, and application and process extension within the IoT network. They exploit integration with critical business systems
  - (e.g., ordering systems, robotics, scheduling, and more) in the execution of related tasks.

  - Data Collection
  - Device Integration
  - Real-Time Analytics
  - Application and Process Extension

IoT – Technology and Protocols

- NFC and RFID
  - **RFID** (radio-frequency identification) and **NFC** (near-field communication) provide simple, low energy, and versatile options for identity and access tokens, connection bootstrapping, and payments.
    - **RFID** technology employs 2-way radio transmitter-receivers to identify and track tags associated with objects.
    - **NFC** consists of communication protocols for electronic devices, typically a mobile device and a standard device.

IoT – Technology and Protocols

- **Low-Energy Bluetooth**
  - This technology supports the low-power, long-use need of IoT function while exploiting a standard technology with native support across systems.
- **Low-Energy Wireless**
  - This technology replaces the most power hungry aspect of an IoT system. Though sensors and other elements can power down over long periods, communication links (i.e., wireless) must remain in listening mode. Low-energy wireless not only reduces consumption, but also extends the life of the device through less use.
- **Radio Protocols**
  - ZigBee, Z-Wave, and Thread are radio protocols for creating low-rate private area networks. These technologies are low-power, but offer high throughput unlike many similar options. This increases the power of small local device networks without the typical costs.

IoT – Technology and Protocols

- LTE-A
  - LTE-A, or LTE Advanced, delivers an important upgrade to LTE technology by increasing not only its coverage, but also reducing its latency and raising its throughput. It gives IoT a tremendous power through expanding its range, with its most significant applications being vehicle, UAV, and similar communication.
- WiFi-Direct
  - WiFi-Direct eliminates the need for an access point. It allows P2P (peer-to-peer) connections with the speed of WiFi, but with lower latency. WiFi-Direct eliminates an element of a network that often bogs it down, and it does not compromise on speed or throughput.

IoT – Common Uses

- Engineering, Industry, and Infrastructure
- Government and Safety
- Home and Office
- Health and Medicine

IoT – Common Uses

- Engineering, Industry, and Infrastructure
- Applications of IoT in these areas include improving production, marketing, service delivery, and safety. IoT provides a strong means of monitoring various processes; and real transparency creates greater visibility for improvement opportunities.
- The deep level of control afforded by IoT allows rapid and more action on those opportunities, which include events like obvious customer needs, nonconforming product, malfunctions in equipment, problems in the distribution network, and more.

IoT – Common Uses

- Government and Safety

- IoT applied to government and safety allows improved law enforcement, defense, city planning, and economic management. The technology fills in the current gaps, corrects many current flaws, and expands the reach of these efforts. For example, IoT can help city planners have a clearer view of the impact of their design, and governments have a better idea of the local economy.

IoT – Common Uses

- Home and Office

- In our daily lives, IoT provides a personalized experience from the home to the office to the organizations we frequently do business with. This improves our overall satisfaction, enhances productivity, and improves our health and safety. For example, IoT can help us customize our office space to optimize our work.

IoT – Common Uses

- Health and Medicine

- IoT pushes us towards our imagined future of medicine which exploits a highly integrated network of sophisticated medical devices. Today, IoT can dramatically enhance medical research, devices, care, and emergency care. The integration of all elements provides more accuracy, more attention to detail, faster reactions to events, and constant improvement while reducing the typical overhead of medical research and organizations.

Bluetooth / GSM / WiFi

- **Bluetooth** is a wireless technology standard for exchanging data between fixed and mobile devices over short distances using short-wavelength UHF radio waves in the industrial, scientific and medical radio bands, from **2.400 to 2.485 GHz**, and building personal area networks.
- **Dr. Jaap Haartsen**, who invented Bluetooth while working at Ericsson in the 1990's.
- Frequency: **2.45 GHz**
- Physical range: Typically less than **10 m (33 ft)**, up to **100 m (330 ft)**; Bluetooth 5.0: **40–400 m (100–1,000 ft)**
- Compatible hardware:
  - Personal computers
  - Smartphones
  - Gaming consoles
  - Audio devices

Bluetooth / GSM / WiFi

- **HC-05 Bluetooth Module** has 6 pins- Vcc, GND, TX, RX, Key, and LED. It comes pre-programmed as a slave, so there is no need to connect the Key pin, unless you need it change it to Master Mode.

- The major difference between Master and Slave modes is that, in Slave mode the Bluetooth module **cannot initiate a connection, it can however accept incoming connections**. After the connection is established the Bluetooth module can transmit and receive data regardless of the mode it is running in. If you are using a phone to connect to the Bluetooth module, you can simply use it in the Slave mode. The default data transmission rate is 9600kbps.

- The range for Bluetooth communication is usually **30m or less**. The module has a factory set pin of "1234" which is used while pairing the module to a phone.

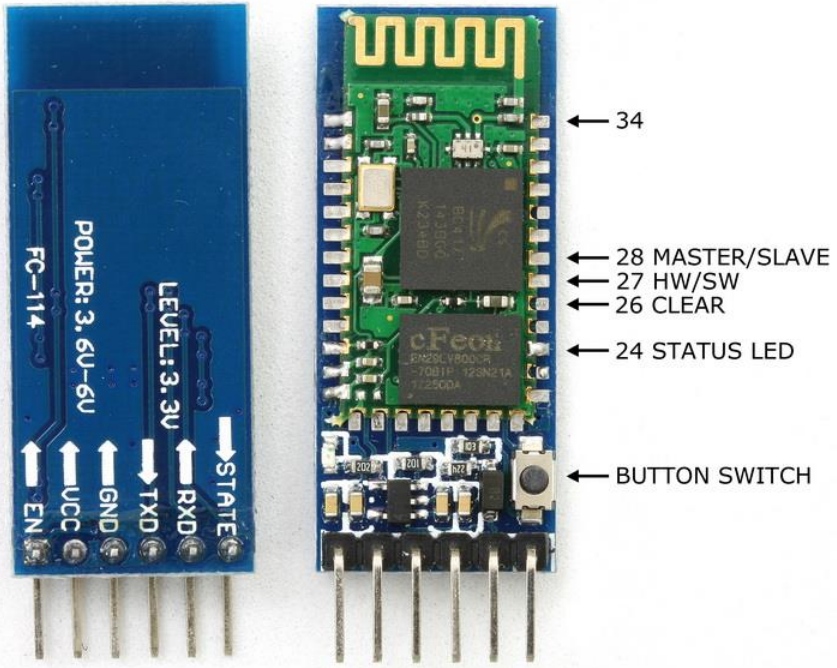
Bluetooth / GSM / WiFi

Pin #	Pin Name	Description
1	Enable / Key	This pin is used to toggle between Data Mode (set low) and AT command mode (set high). By default it is in Data mode
2	Vcc	Powers the module. Connect to +5V Supply voltage
3	Ground	Ground pin of module, connect to system ground.
4	TX – Transmitter	Transmits Serial Data. Everything received via Bluetooth will be given out by this pin as serial data.
5	RX – Receiver	Receive Serial Data. Every serial data given to this pin will be broadcasted via Bluetooth
6	State	The state pin is connected to on board LED, it can be used as a feedback to check if Bluetooth is working properly.
7	LED	Indicates the status of Module •Blink once in 2 sec: Module has entered Command Mode •Repeated Blinking: Waiting for connection in Data Mode •Blink twice in 1 sec: Connection successful in Data Mode
8	Button	Used to control the Key/Enable pin to toggle between Data and command Mode

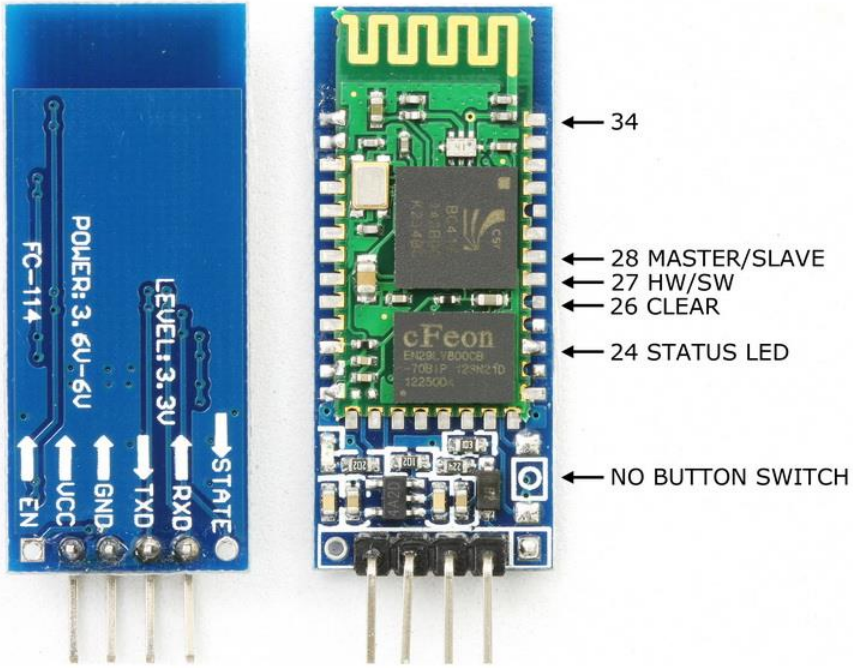
- AT command mode allows you to interrogate the Bluetooth module and to change some of the settings; like the name, the baud rate, whether or not it operates in slave mode or master mode. When used as a master device AT commands allow you to connect to other Bluetooth slave devices.

Bluetooth / GSM / WiFi

HC-05 FC-114

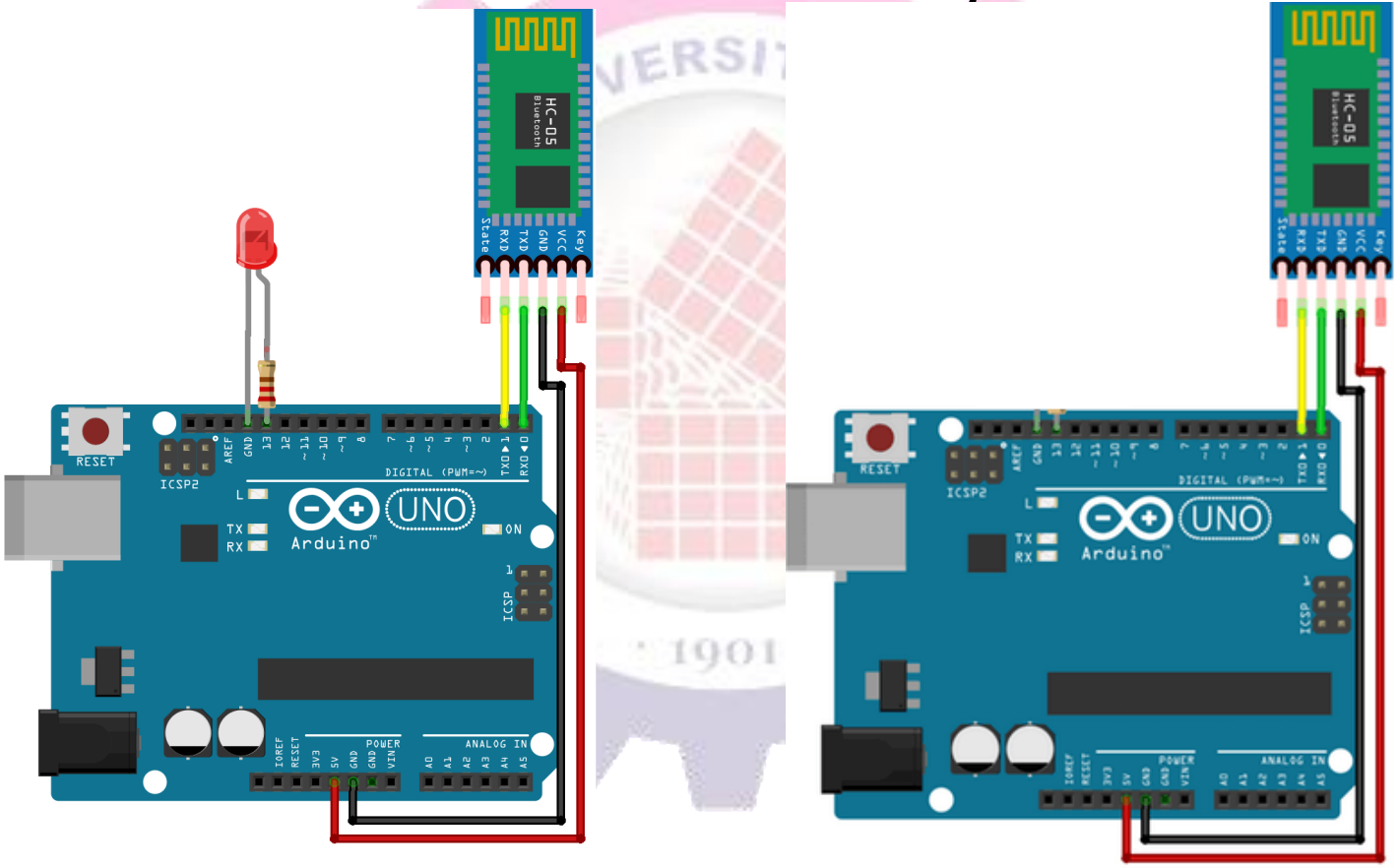


HC-06 FC-114



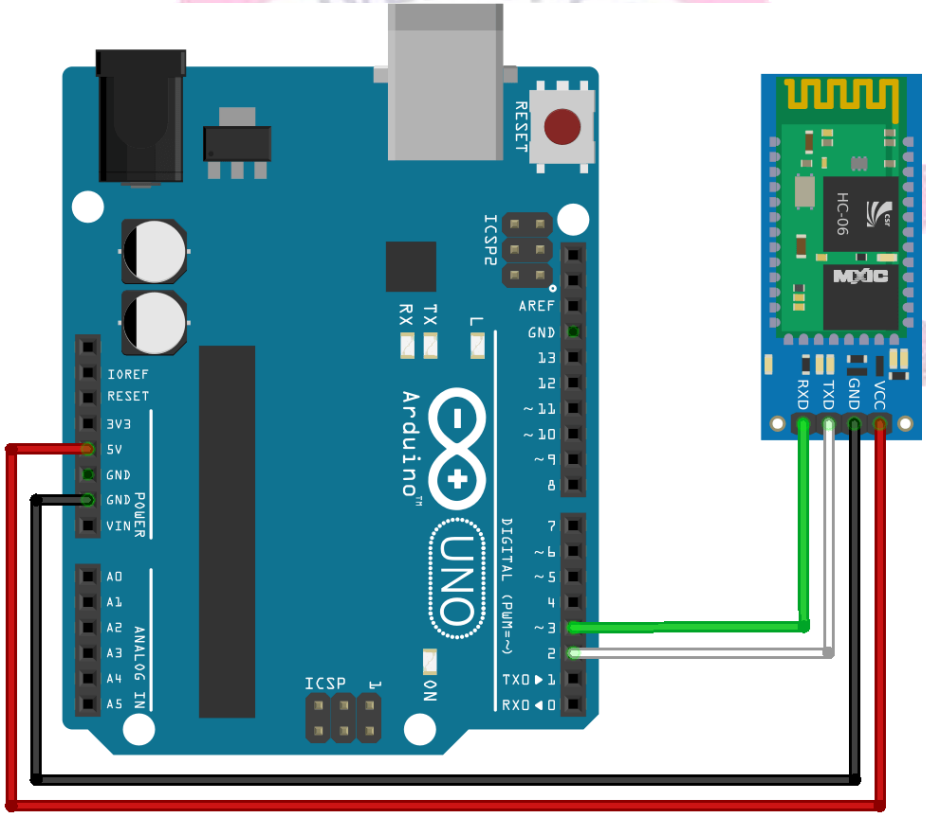
Bluetooth / GSM / WiFi

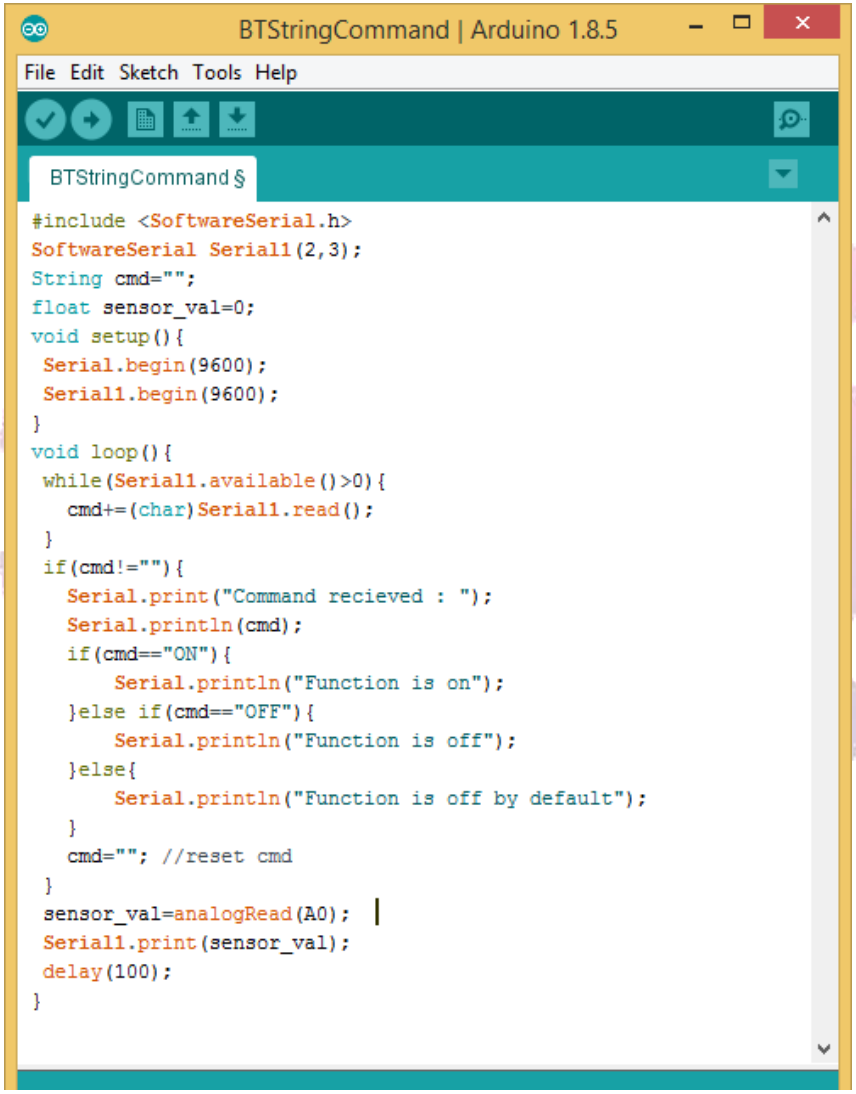
- Bluetooth and Arduino wiring connection



Bluetooth / GSM / WiFi

- Bluetooth and Arduino wiring connection




BTStringCommand | Arduino 1.8.5
File Edit Sketch Tools Help
BTStringCommand §
#include <SoftwareSerial.h>
SoftwareSerial Serial1(2,3);
String cmd="";
float sensor_val=0;
void setup(){
Serial.begin(9600);
Serial1.begin(9600);
}
void loop(){
while(Serial1.available()>0){
cmd+=(char)Serial1.read();
}
if(cmd!=""){
Serial.print("Command recieved : ");
Serial.println(cmd);
if(cmd=="ON"){
Serial.println("Function is on");
}else if(cmd=="OFF"){
Serial.println("Function is off");
}else{
Serial.println("Function is off by default");
}
cmd=""; //reset cmd
}
sensor_val=analogRead(A0);
Serial1.print(sensor_val);
delay(100);
}

Zeigbee

- ZigBee is an open global standard for wireless technology designed to use low-power digital radio signals for personal area networks.

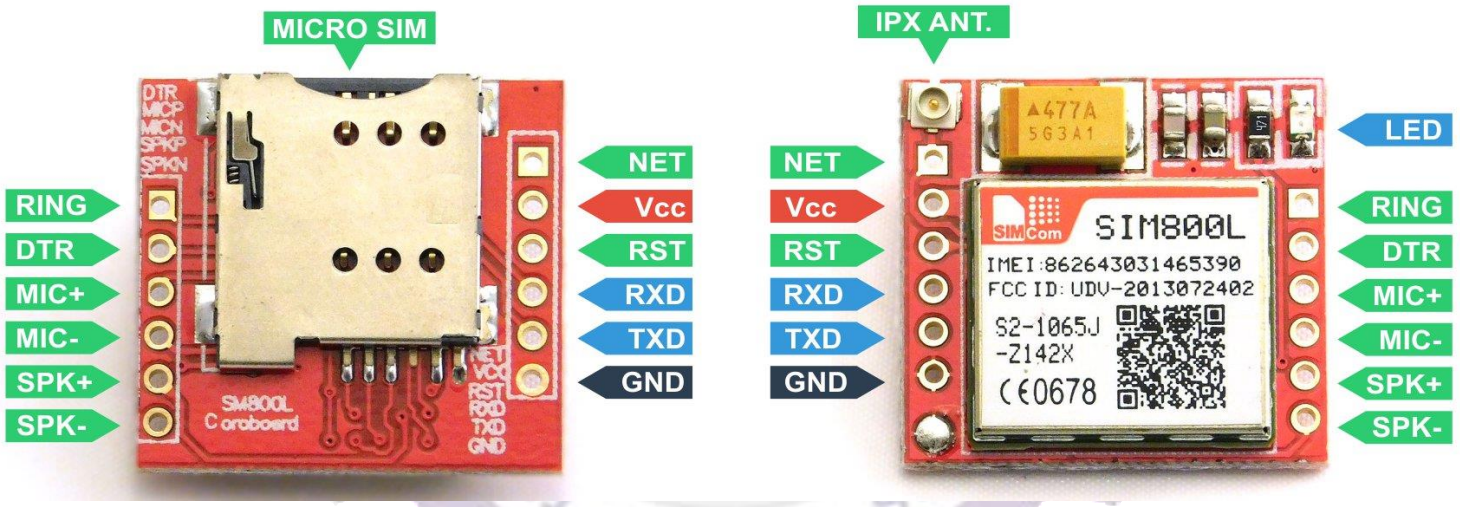
Arduino – GSM Module

- Global System for Mobile Communications (GSM) is a standard developed by the European Telecommunications Standards Institute (ETSI) to describe the protocols for second-generation (2G) digital cellular networks used by mobile devices such as mobile phones and tablets.

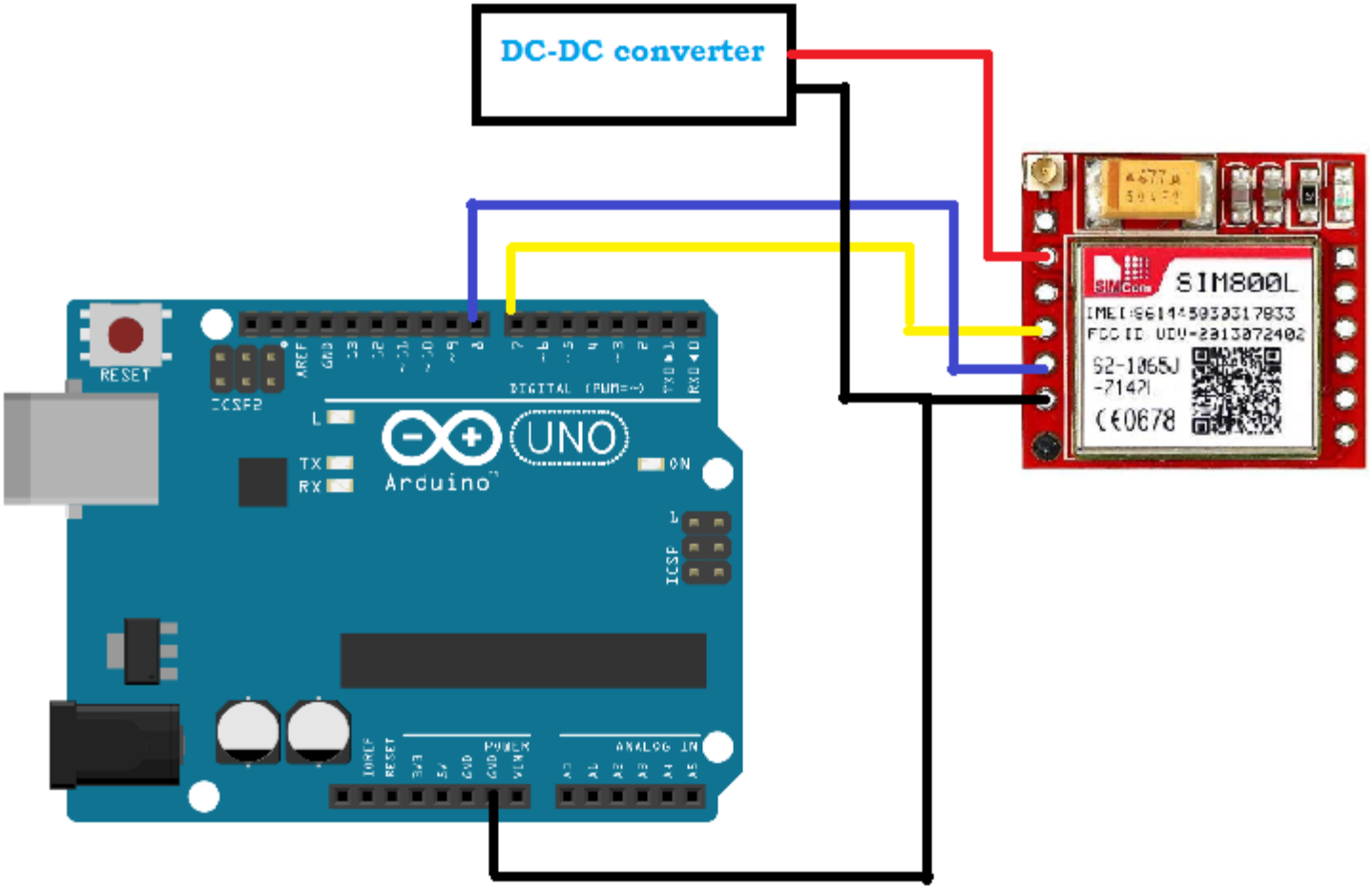
- General Packet Radio Service (GPRS) is a packet oriented mobile data standard on the 2G and 3G cellular communication network's global system for mobile communications (GSM).

- SIM800L GSM/GPRS module is a miniature GSM modem

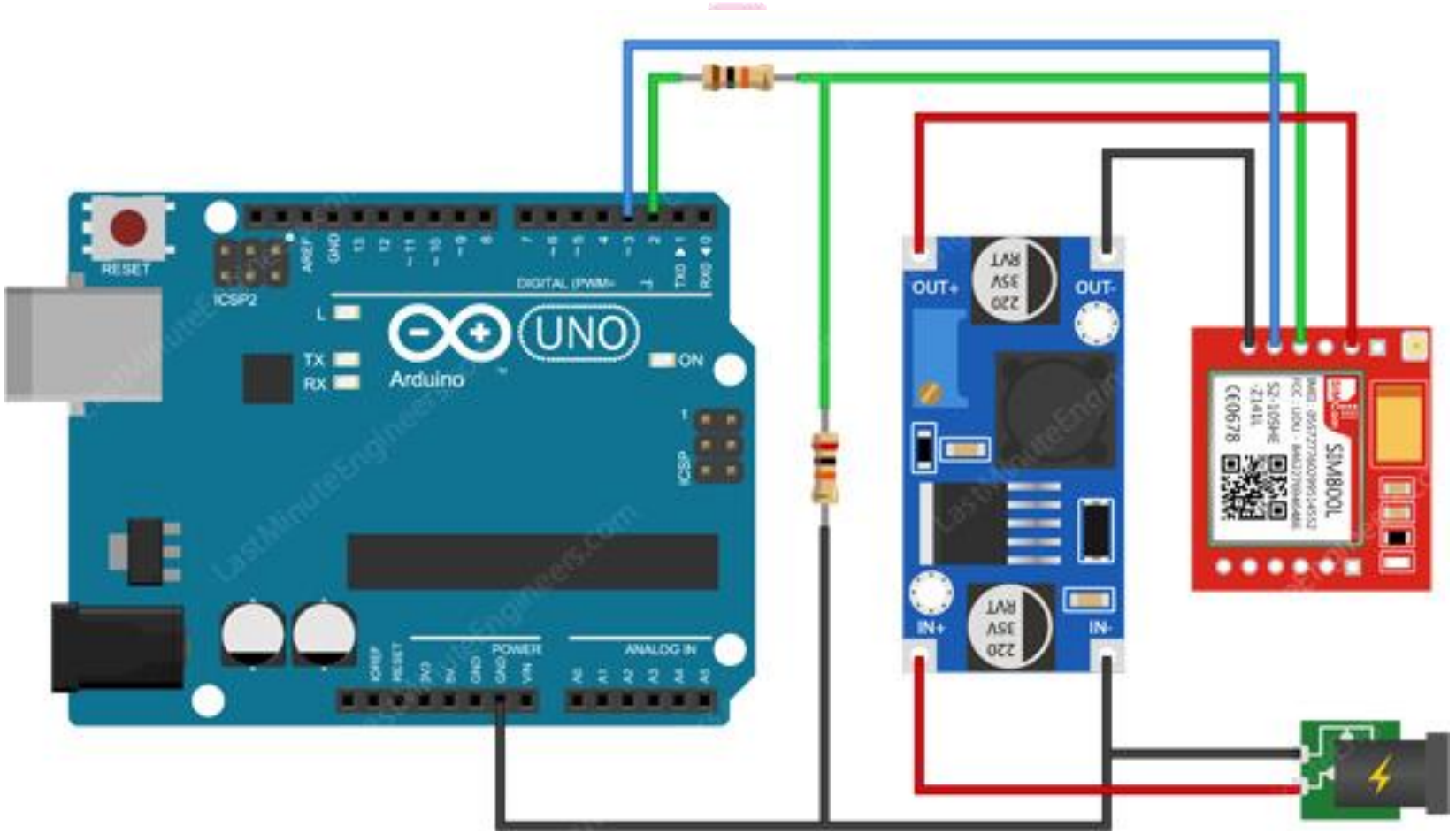
SIM800L Pin Config



GSM wiring connection



GSM wiring connection



- https://lastminuteengineers.com/sim800l-gsm-module-arduino-tutorial/

GPS vs. GPRS

- GPS
  - Global Positioning system, is a navigation technology reliant on a satellite system based in space.
  - GPS technology can pinpoint any position or address on Earth. Though it requires multiple satellites to operate, it can be used practically anywhere around the world, making it extremely accessible and advantageous to businesses with many different geographic locations.

GPS vs. GPRS

- GPRS
  - General Packet Radio Service, is the most commonly used wireless data service.
  - Older versions relied on 2G cellular networks, but now most use 3G technology.
  - GPRS enables cellular devices to perform functions such as multimedia messaging and Internet surfing. It generally operates on an "as used" payment model in fleet tracking.

- Pros & cons:
- As far as managing fleet operations, GPRS and GPS tracking systems offer varied benefits.

- GPRS tracking systems are often more cost-efficient than GPS systems.

- A GPRS vehicle tracking system only uses the data network when data is being transmitted. Therefore, charges apply strictly for actual usage.

References

- https://en.wikipedia.org/wiki/Arduino
- http://arduinohistory.github.io/

Thank you for listening…