kitsNspares.com

For the 'engineer' in you

DO-IT-YOURSELF

ARDUINO SELF-LEARNING KIT

You now have an awesome development board on your hands and are raring to start building some interesting projects with it! That is just what this Arduino Self-Learning Kit will help you with. It comes with all the required components and modules to help you get started with building cool electronics projects with the Arduino development board.

Kits ‘n’ Spares (KnS) exclusively provides you with a detailed manual, which covers the basics in a short and easily understandable manner, with circuit diagrams and programs.

Happy Inventing!!

Contents

Introduction to the World of ARDUINO

What is Arduino?

Arduino is an open-source prototyping platform based on easy-to-use hardware and software. Arduino boards are capable of reading inputs such as light on a sensor, a finger on a button, or a Twitter message, and causing an output, such as activation of a motor, turning on a light-emitting diode (LED) or publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. This is done using the Arduino Programming Language, which is based on Wiring, and the Arduino Software (Integrated Development Environment or IDE), which is based on Processing.

Why Arduino?

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac , Windows and Linux OS.

Teachers and students use it to build low-cost scientific instruments, to demonstrate different principles of chemistry and physics, or to get started with programming and robotics. Designers and architects use it to build interactive prototypes, while musicians and artists use it for to install and experiment with new musical instruments. Makers, of course, use it to build exciting projects, some of which are exhibited at events like the Maker Faire. Indeed, Arduino is a wonderful tool to learn new things. Anyone – children, hobbyists, artists, programmers – can start tinkering by just following the simple, step-by-step instructions provided with a kit, and by learning from ideas and content shared online by the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools hide the messiness of microcontroller programming, and wrap it up in an easy-to-use package. Arduino does something similar, but is preferred by teachers, students and amateurs due to the following reasons:

- **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 US$ 50.

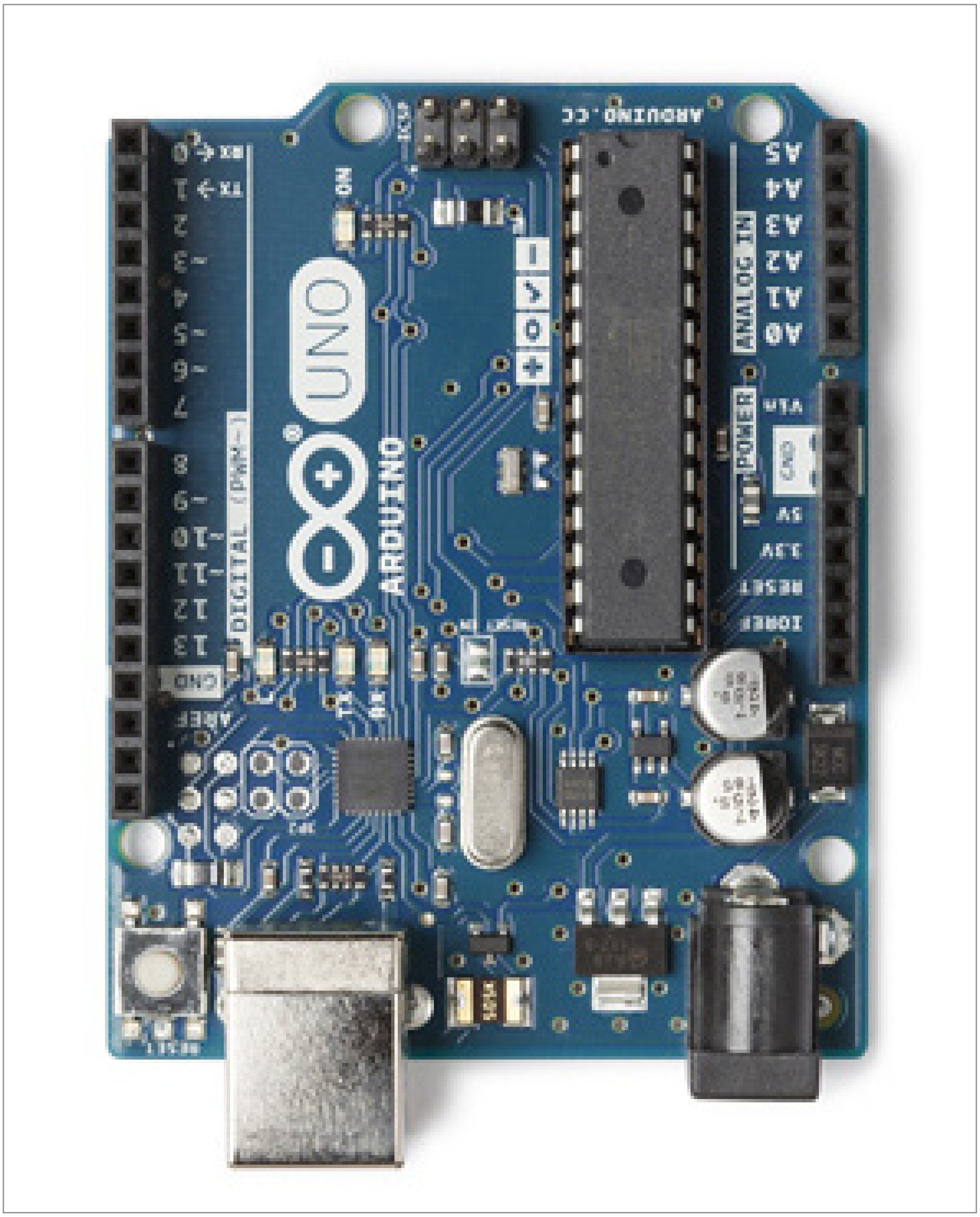
- **CROSS-PLATFORM** - The Arduino Software (IDE) runs on Windows, Mac OS, OSX and Linux operating systems. Most microcontroller systems are limited to Windows .

- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users also to harness its capabilities to their advantage. It is preferred by teachers because it is conveniently based on the Processing programming environment. So, students learning to program in that environment will be familiar with how the Arduino IDE works.

- **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 is 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 (CC) 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.

ARDUINO UNO


UNO
ARDUINO
DIGITAL (PWM~)
ANALOG IN
POWER
ICSP
ARDUINO.CC
A5
A4
A3
A2
A1
A0
VIN
GND
5V
3.3V
RESET
IOREF
AREF
GND
ON
TX
RX
0
1
2
~3
4
~5
~6
7
8
~9
~10
~11
12
13
RESET

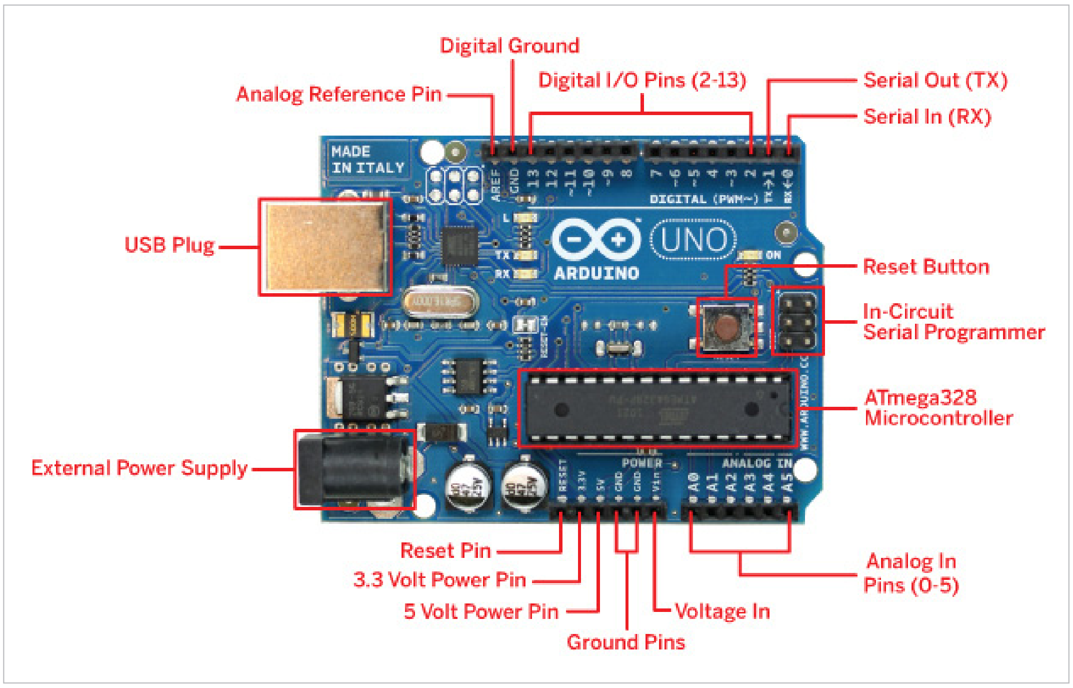
Overview

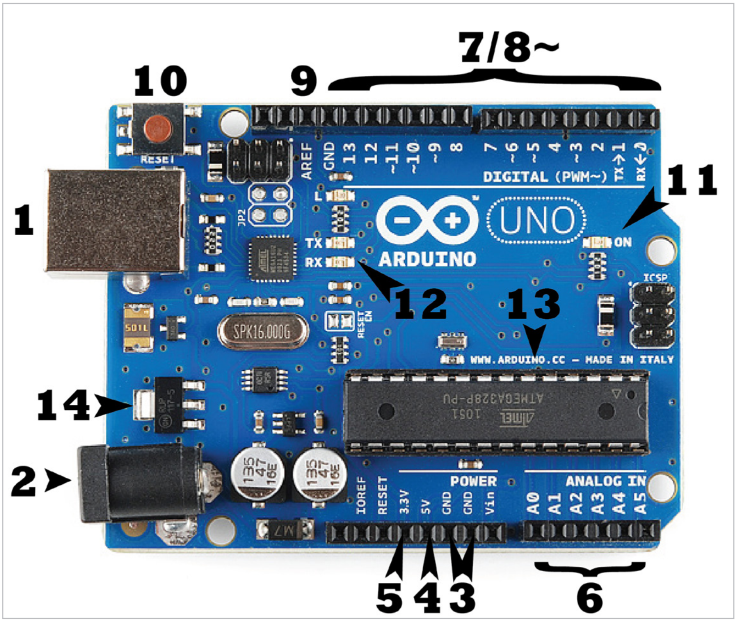
The Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output (I/O) pins, of which six can be used as pulse-width modulation (PWM) outputs, six analog inputs, a 16MHz quartz crystal, a Universal Serial Bus (USB) connection, a power jack, an in-circuit serial programming (ICSP) header and a reset button. It contains everything needed to support the microcontroller. You just need to connect it to a computer with a USB cable or power it with an alternating-current-to-direct-current (AC-to-DC) adapter or battery to get started. You can tinker with your Uno without worrying too much about doing something wrong, because even in the worst case, you can just replace the chip economically and start over again.

Technical Specs

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB is used by the bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

A Detailed Description of What's Inside the Arduino Uno





Power (USB/ Barrel Jack)

The Arduino Uno can be powered through a USB cable connected to your computer or a wall power supply that terminates in a barrel jack. In the picture above the USB connection is labelled (1) and the barrel jack is labelled (2).

The USB connection is also how you will load code onto your Arduino board.

NOTE: Do not use a power supply greater than 20V as you will overpower (and thereby destroy) your Arduino. The recommended voltage for most Arduino models is between 6V and 12V.

Pins (5V, 3.3V, GND, Analog, Digital, PWM, AREF)

You will be using the pins on your Arduino for connecting wires, to construct a circuit, generally also using a breadboard and some wire. They usually have black plastic 'headers' that allow you to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labelled on the board and used for different functions.

- GND (3): Short for 'Ground'. There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- 5V (4) & 3.3V (5): As you might guess, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run comfortably on 5 or 3.3 volts.
- Analog (6): The set of pins under the 'Analog In' label (A0 through A5 on the Uno) can read signals from an analog sensor (like a temperature sensor) and convert it into a digital value that we can read.
- Digital (7): On the other side, across the analog pins are the 'Digital Pins' (0 through 13 on the Uno). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED).
- PWM (8): You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the Uno). These pins act as normal digital pins, but can also be used for pulse-width modulation (PWM). We will discuss it later, but for now, think of these pins as being able to simulate analog output (like fading an LED in and out).
- AREF (9): Stands for Analog Reference. Most of the time you can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 volts) as the upper limit for the analog input pins.

Reset Button

The Arduino has a reset button (10). Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if your code does not repeat, but you want to test it multiple times.

Power LED Indicator

Just beneath and to the right of the word "UNO" on your circuit board, there is a tiny LED next to the word 'ON' (11). This LED should light up whenever you plug your Arduino into a power source. If this light does not turn on, something might be wrong, and you may have to re-check your circuit!

TX RX LEDs

TX is short for transmit, RX is short for receive. These markings appear quite often in electronics to indicate the pins responsible for serial communication. There are two places on the Arduino Uno where TX and RX appear – near the digital pins 0 and 1, and next to the TX and RX indicator LEDs (12). These LEDs give visual indications whenever the Arduino receives or transmits data (e.g. when we are loading a new program onto the board).

Main Integrated Circuit

The black thing with metal legs is an IC or integrated circuit (13). Think of it as the brains of our Arduino. The main IC on the Arduino differs according to the type of the board, but is usually from the ATmega line of ICs from ATMEL. It might be necessary to know the type of IC (along with your board type) before loading a new program from the Arduino Software. This information can usually be found in writing on the top of the IC. If you want to know more about the differences between various ICs, reading the datasheets might help you further.

Voltage Regulator

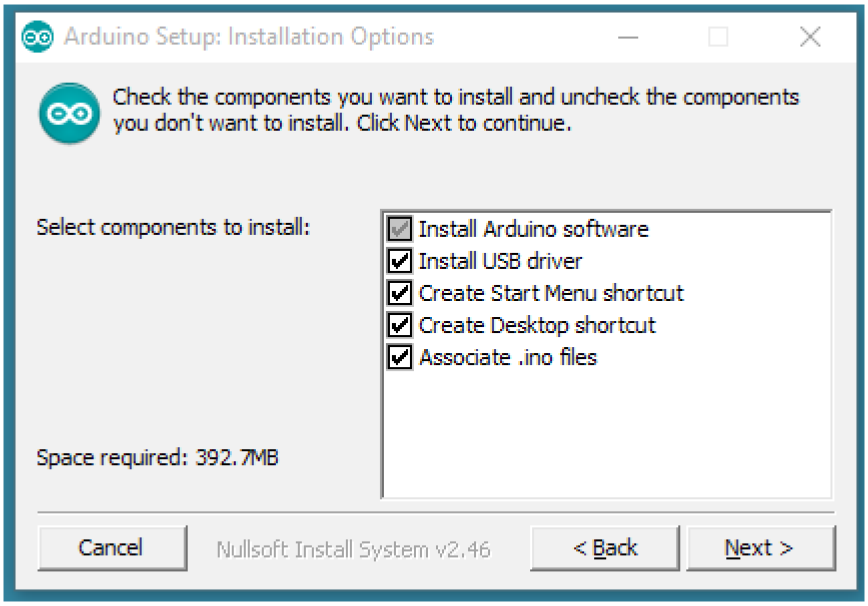
The voltage regulator (14) is not actually something you can (or should) interact with on the Arduino. However, it is useful to know what it is and why it is there. The voltage regulator controls the amount of voltage that is let into the Arduino board. Think of it as a kind of gatekeeper; it will turn away an extra voltage that might harm the circuit. Of course, it has its limits, and cannot manage more than 20 volts.

Getting Started with the Arduino

Now that you have bought the Arduino Self-Learning Kit, it time to get your hands dirty and start learning. So let us get started:

Installation of Arduino Software

1. First you need to copy the setup file of the Arduino IDE from the given DVD into your PC.
2. Then double-click the setup file and perform the following steps:



Arduino Setup: Installation Options

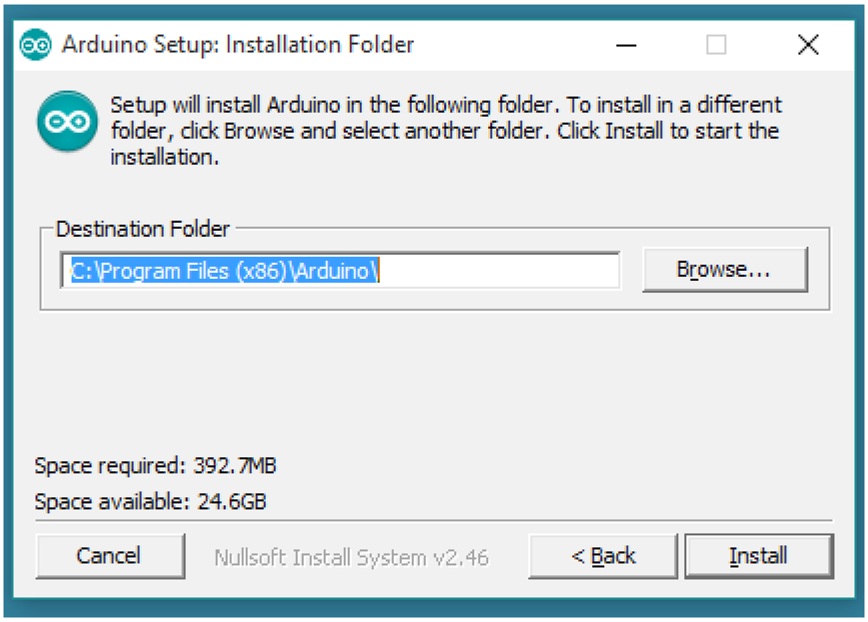
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] Install Arduino software
- [✓] Install USB driver
- [✓] Create Start Menu shortcut
- [✓] Create Desktop shortcut
- [✓] Associate .ino files

Space required: 392.7MB

Choose the components to install.

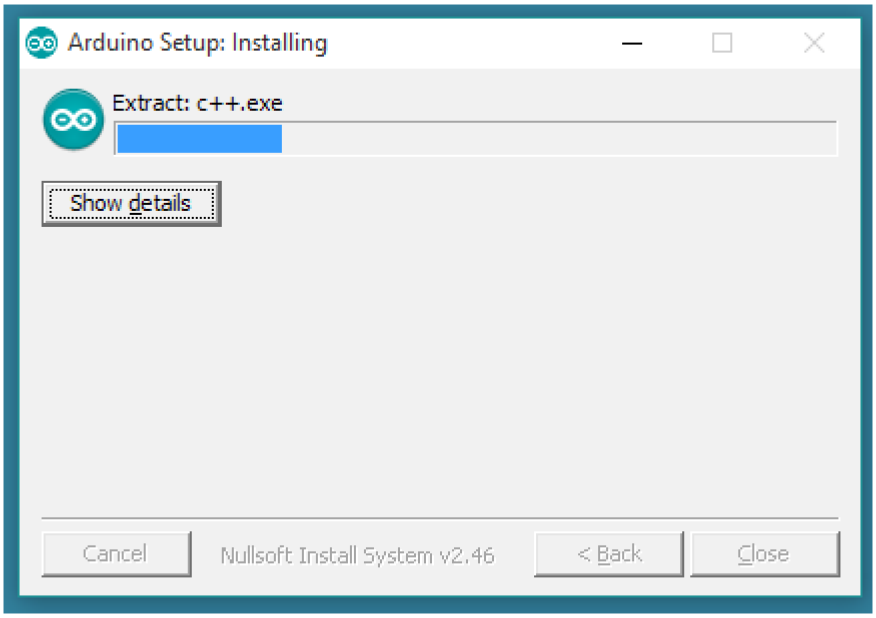

Arduino Setup: Installation Folder

Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino\

Space required: 392.7MB
Space available: 24.6GB

Choose the installation directory (prefer the default one).


Arduino Setup: Installing

Extract: c++.exe

The process will extract and install all the files required to properly execute the Arduino Software (IDE).

3. Connect the board to your computer using the USB cable. The green power LED (labelled PWR) should turn on.

Installation of Board Drivers

If you used the Installer, Windows (from XP up to 10) will install the drivers automatically as soon as you connect your board.
If you downloaded and expanded the Zip package or, for some reason, the board was not properly recognised, follow the procedure described below.

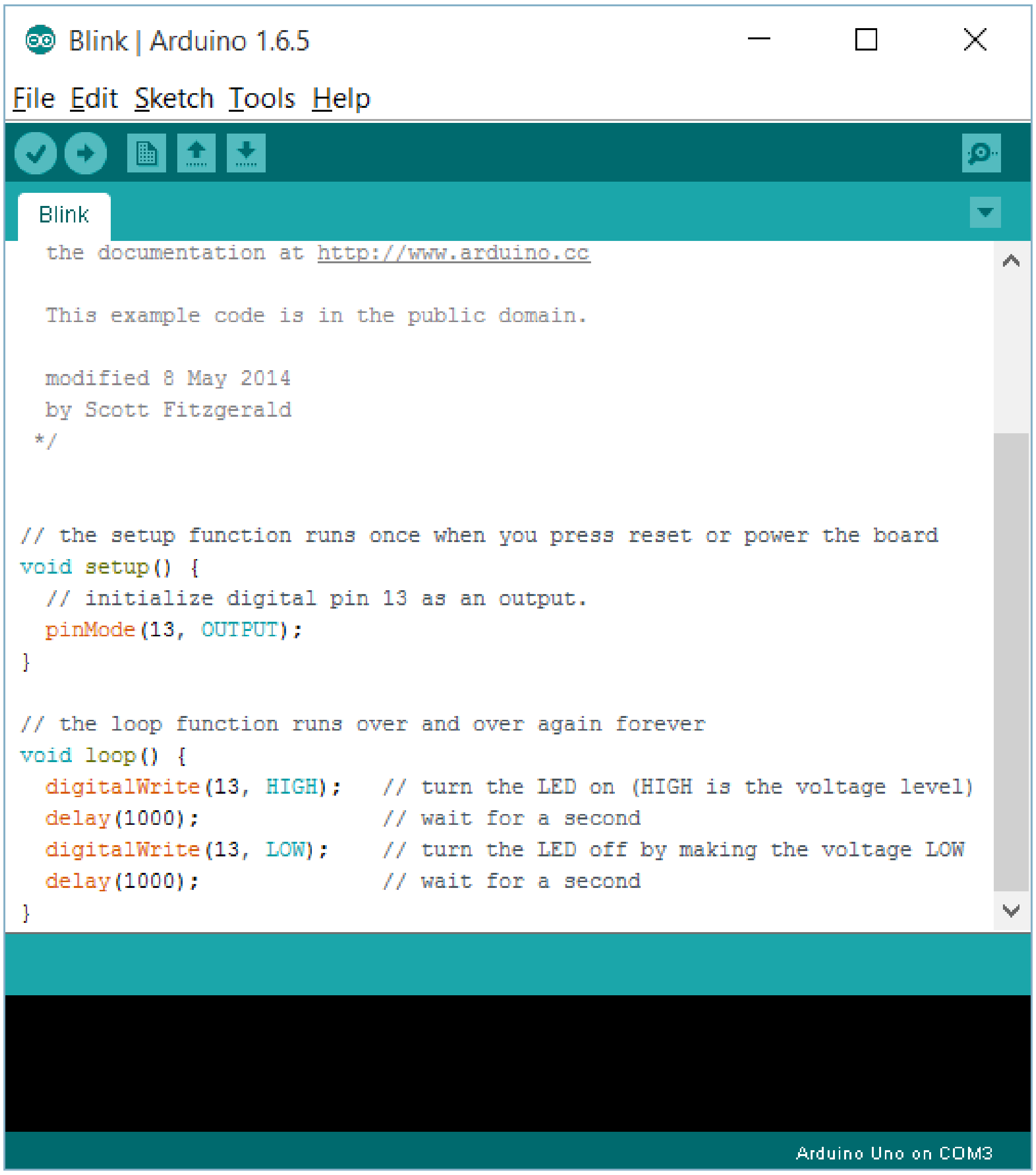
- Click on the Start Menu, and open Control Panel.
- In the Control Panel, navigate to System and Security. Click on System, and from there open Device Manager.
- Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)". If there is no COM & LPT section, look under "Other Devices" for "Unknown Device".
- Right click on the "Arduino UNO (COmxx)" port and choose the "Update Driver Software" option.
- Next, choose the "Browse my Computer for Driver software" option.
- Finally, navigate to and select the driver file named "arduino.inf", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory). If you are using an old version of the IDE (1.0.3 or older), choose the Uno driver file named "Arduino UNO.inf"
- Windows will finish the driver installation from there.

Launching the Arduino Software (IDE)

Double-click the Arduino icon (arduino.exe) created by the installation process.

Opening the Blink Example

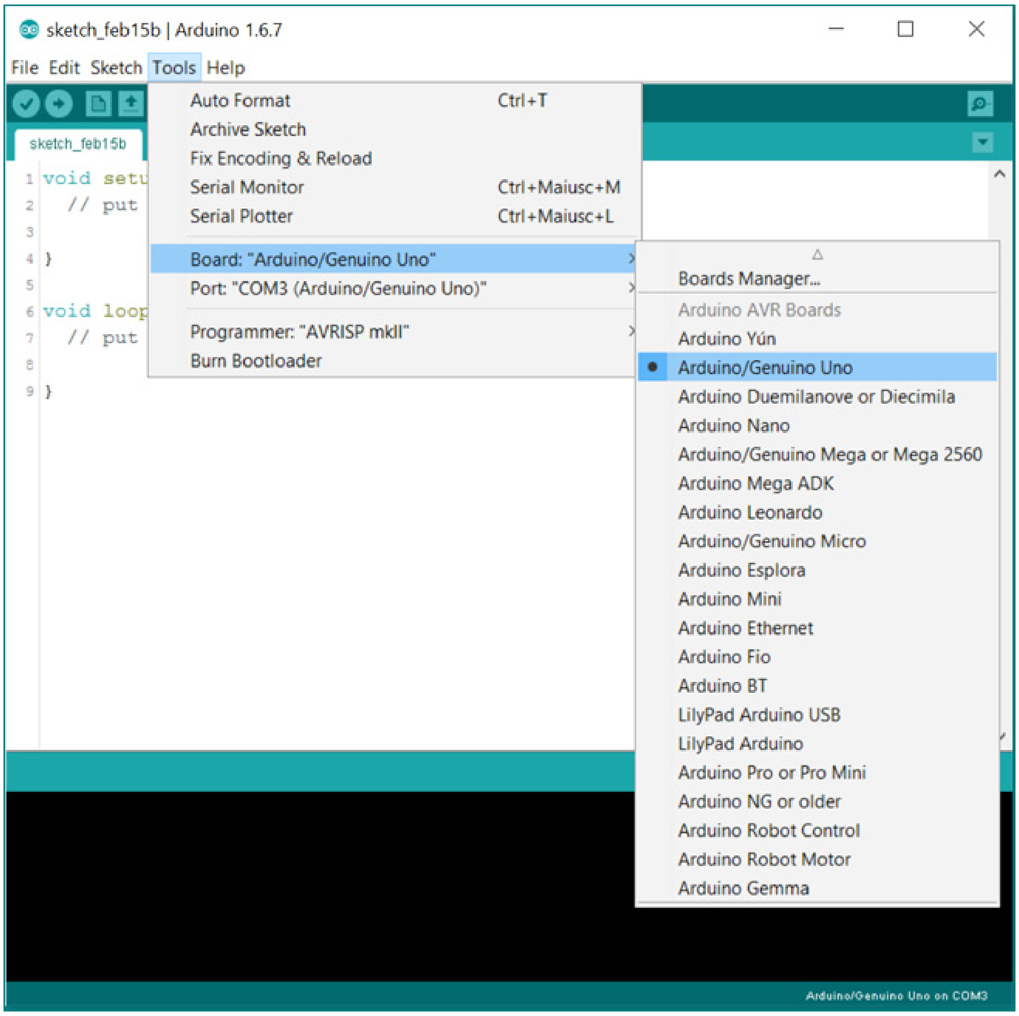
Open the LED blink example sketch: File > Examples > 01.Basics > Blink.



```
  the documentation at http://www.arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
 */


// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Selecting your Board

You will need to select the entry in the Tools > Board menu that corresponds to your Arduino Board.



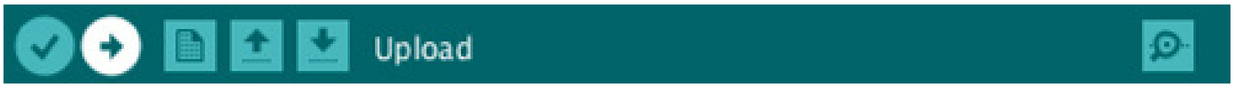
```
void setu
  // put

}

void loop
  // put

}
```

Selecting your Serial Port

Select the serial device of the board from the Tools > Serial Port menu. This is likely to be COM3 or higher (COM1 andCOM2 are usually reserved for hardware serial ports). To find out, you can disconnect your board and re-open the menu; the entry that disappears should be the Arduino. Reconnect the board and select that serial port.

Uploading the Program

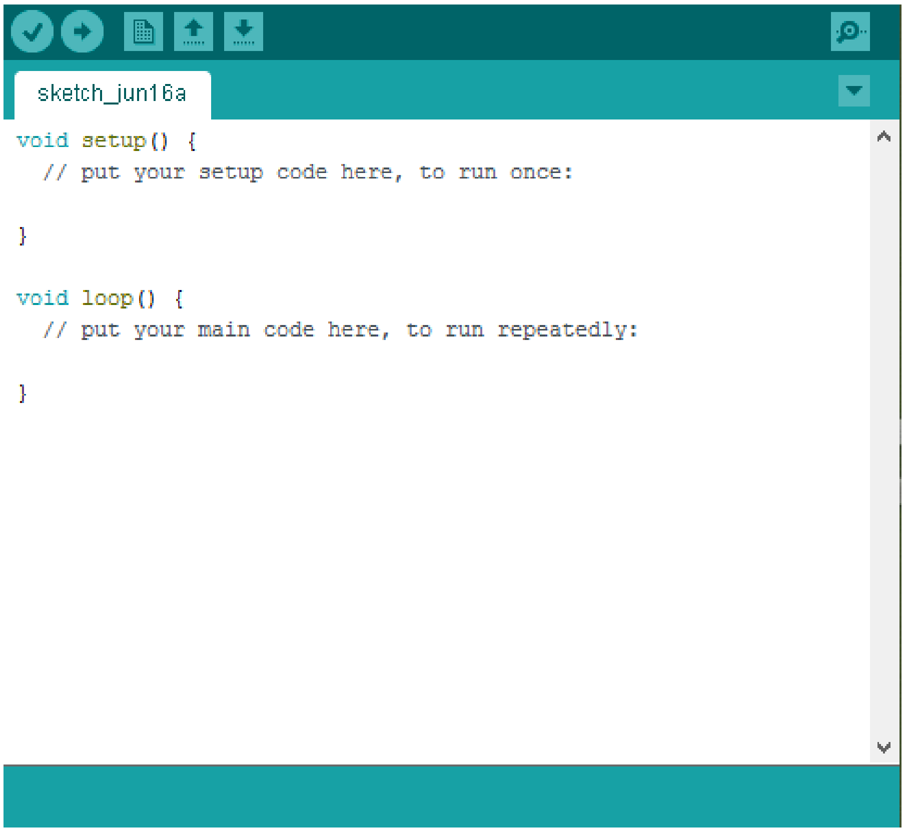
Now, simply click the “Upload” button in the environment. Wait for a few seconds – you should see the RX and TX LEDs on the board flashing. If the upload is successful, the message “Done uploading” will appear in the status bar.



Basics of Arduino Programming

Structure

The basic structure of the Arduino Programming Language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of statements.



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

}

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

}
```

The **setup() function** is called when a sketch starts. Use it to initialise variables, pin modes, start using libraries, etc. The setup function will only run once, after each power-up or reset of the Arduino board.

After creating a setup() function, which initialises and sets the initial values, the **loop() function** loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Functions

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program.

There are two required functions in an Arduino sketch, setup() and loop(). Other functions must be created outside the brackets of those two functions.

Digital I/O Functions:

pinMode()
Configures the specified pin to behave either as an input or an output. (See the description of digital pins for details on the functionality of the pins.)

Syntax:
pinMode(pin, mode);

Parameters:
pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT, or INPUT_PULLUP (See the digital pins page for a more complete description of the functionality.)

digitalWrite()

Writes a HIGH or a LOW value to a digital pin.
If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for

LOW.
If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pull-up on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

Syntax
digitalWrite(pin, value)

Parameters
pin: the pin number

value: HIGH or LOW

digitalRead()
Reads the value from a specified digital pin, either HIGH or LOW.

Syntax
digitalRead(pin)

Parameters
pin: the number of the digital pin you want to read (int)

Returns
HIGH or LOW

Analog I/O Functions:

analogReference()
Configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options are:

DEFAULT: The default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)

INTERNAL: A built-in reference, equal to 1.1 volts on
the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (not available on the Arduino Mega)
INTERNAL1V1: A built-in 1.1V reference (Arduino Mega only)
INTERNAL2V56: A built-in 2.56V reference (Arduino Mega only)

EXTERNAL: The voltage applied to the AREF pin (0 to 5V only) is used as the reference

Syntax
analogReference(type)

Parameters
type: which type of reference to use (DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, or EXTERNAL)

analogRead()

Reads the value from the specified analog pin.

The Arduino board contains a 6-channel (eight channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts/1024 units or .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().

It takes about 100 microseconds (0.0001s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax
analogRead(pin)

Parameters
pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns
int (0 to 1023)

analogWrite()

Writes an analog value (PWM wave) to a pin.

Can be used to light an LED at varying brightness or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite() on the same pin). The frequency of the PWM signal on most pins is approximately 490Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980Hz. Pins 3 and 11 on the Leonardo also run at 980Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10 and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11.

You do not need to call pinMode() to set the pin as an output before calling analogWrite().

The analogWrite function has nothing to do with the analog pins or the analogRead function.

Syntax
analogWrite(pin, value)

Parameters
pin: the pin to write to
value: the duty cycle, between 0 (always off) and 255 (always on)

Time Functions

delay()
Pauses the program for the amount of time (in miliseconds) specified as a parameter. Note: There are 1000 milliseconds in a second.

Syntax
delay(ms)

Parameters
ms: the number of milliseconds to pause (unsigned long)

delayMicroseconds()

Pauses the program for the amount of time (in microseconds) specified as parameter. Note: There are a thousand microseconds in a millisecond, and a million microseconds in a second.

Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

Syntax
delayMicroseconds(us)

Parameters
us: the number of microseconds to pause (unsigned int)

millis()

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

Parameters
None

Returns
Number of milliseconds since the program started (unsigned long)

For more information, refer Arduino- > Help- > Reference.

Projects on Arduino

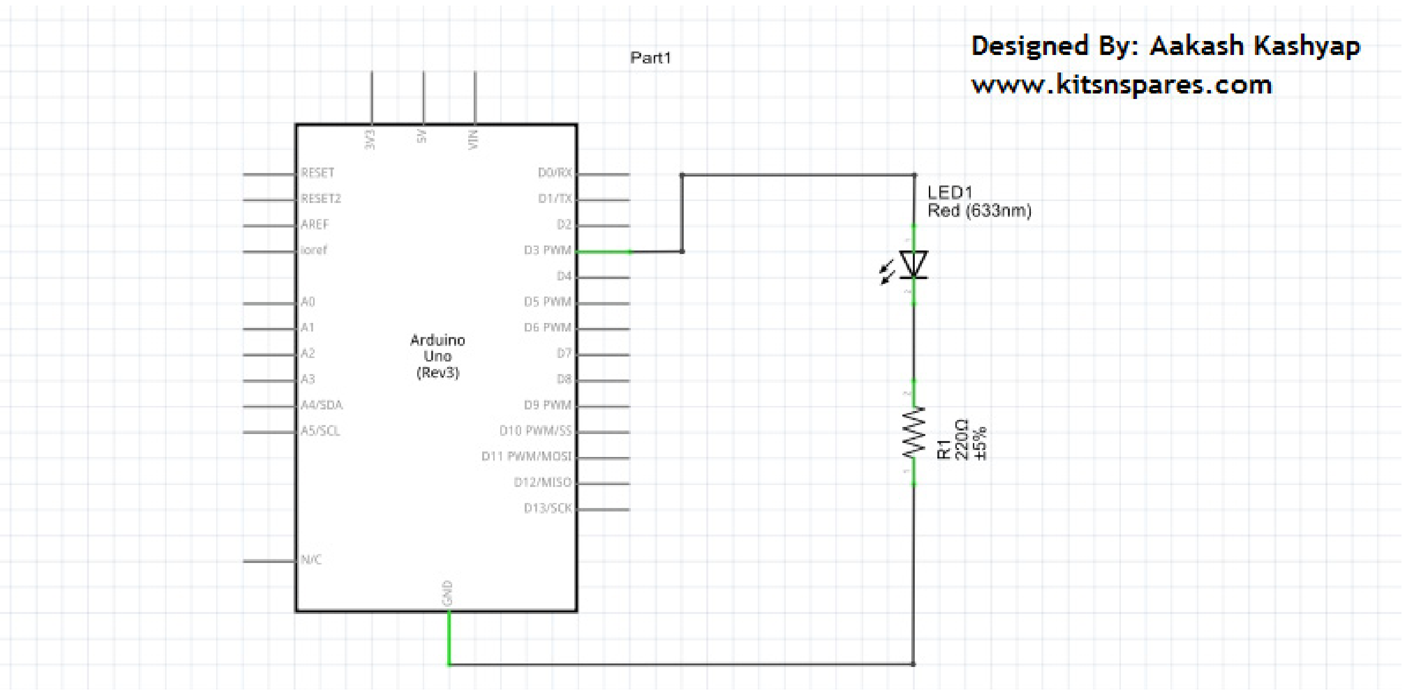
LED Blinking

Here is your first program involving hardware. Although a simple project, you can learn a lot from it.

What we will do:

We will connect an LED to the Arduino and make it blink.

Circuit Diagram:



Note: All code is provided in the DVD, as part of the kit.

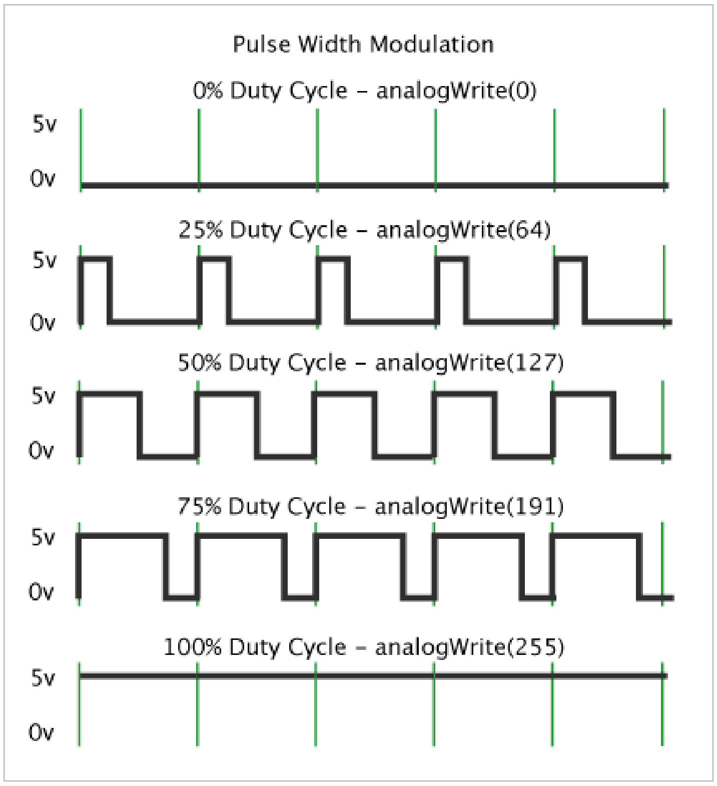
Further experimentation:

You can add more LEDs (RGB) and program them to blink at the same time, or at different timings.

PWM using LED Fading

Pulse-width modulation (PWM) is a technique for getting analog results through digital means. Digital control is used to create a square wave, a signal switched between ON and OFF. This on-off pattern can simulate voltages in between full ON (5V) and OFF (0V) by changing the portion of the time the signal spends on ON versus the time that the signal spends on OFF. The duration of "on time" is called the pulse width. To get varying analog values, you change or modulate that pulse width. If you repeat this on-off pattern fast enough with an LED, for example, the result is as if the signal is a steady voltage between 0 and 5 volts controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. For example, a call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100 per cent duty cycle (always on), and analogWrite(127) is a 50 per cent duty cycle (on for half the time).


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

Once you get this example running, grab your Arduino and shake it back and forth. What you are doing here is essentially mapping time across the space. To our eyes, the movement blurs each LED blink into a line. As the LED fades in and out, those little lines will grow and shrink in length. Now you are seeing the pulse width.

The program is given in the File > Sketchbook > Examples > Analog menu of the Arduino Software.

Note: Circuit diagram is the same as for the Blink program.

Interfacing an LCD

What we will do:

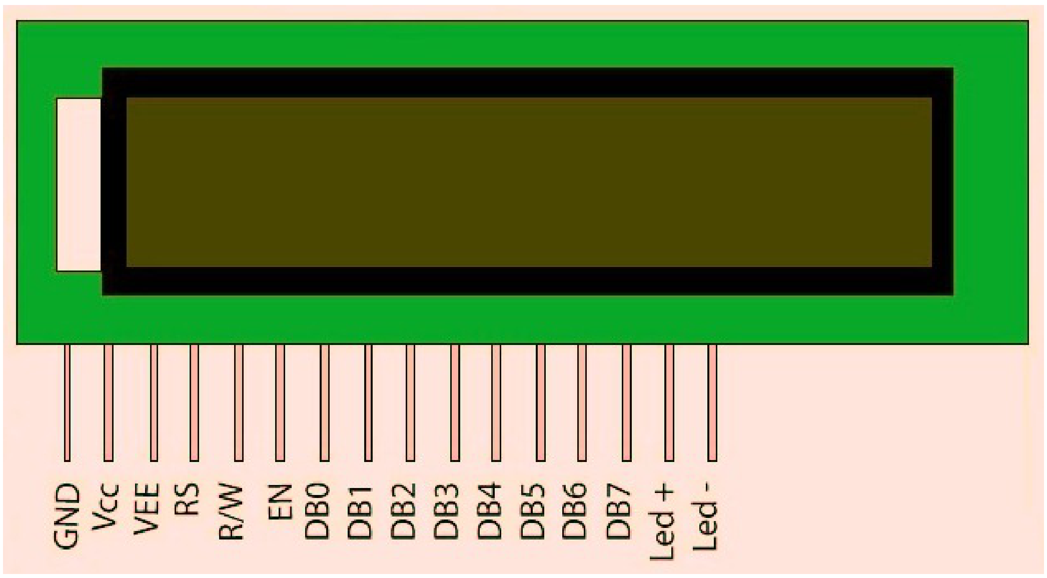
We will interface a 16x2 liquid crystal display (LCD) with the Arduino and write a program to display ‘Hello World’ on the screen.

Liquid Crystal Display (LCD): An LCD screen is an electronic display module, which finds a wide range of applications. A 16x2 LCD display is a very basic module and is very commonly used in various devices and circuits. These modules are preferred over 7-segment and other multi-segment LEDs. The reasons being: LCDs are economical; easily programmable; have no limitations while displaying special or even custom characters (unlike in 7-segment), animations and so on.

A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD, each character is displayed in a 5x7 pixel matrix. This LCD has two registers, namely, Command and Data.

The Command register stores the command instructions given to the LCD. A command is an instruction given to the LCD to do a predefined task like initialising it, clearing its screen, setting the cursor position, controlling the display etc. The Data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD.

Pin Diagram:

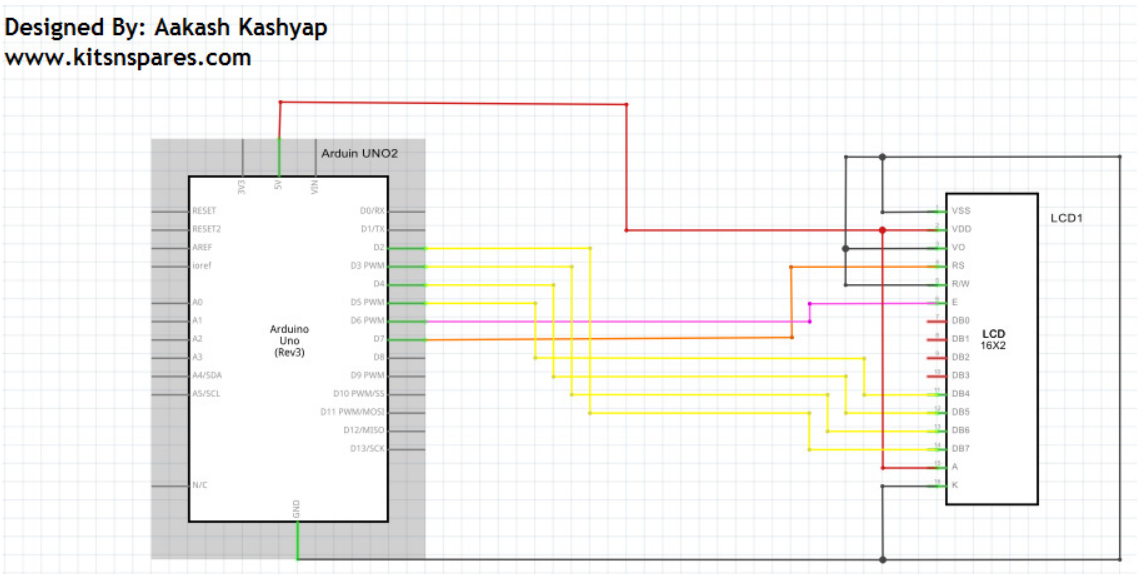


Pin Description:

Pin No	Function	Name
1	Ground (0V)	Ground
2	Supply voltage; 5V (4.7V – 5.3V)	Vcc
3	Contrast adjustment; through a variable resistor	V_{EE}
4	Selects command register when low; and data register when high	Register Select
5	Low to write to the register; High to read from the register	Read/write
6	Sends data to data pins when a high to low pulse is given	Enable
7	8-bit data pins	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7
15	Backlight V_{CC} (5V)	Led+
16	Backlight Ground (0V)	Led-

Usually the device requires eight data lines to provide data to Bits 0-7. However, the device can be set to a 4-bit mode, which allows you to send data in two chunks (or nibbles) of four bits. This reduces the number of digital pin connections you require when interfacing with your Arduino.

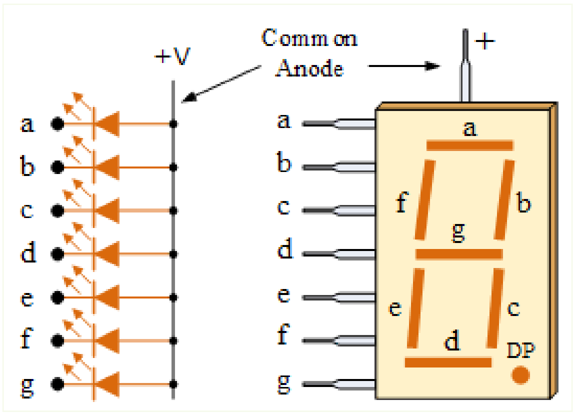
Circuit Diagram



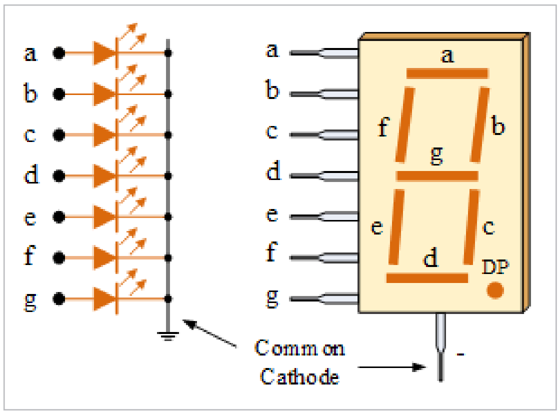
Interfacing a 7-Segment Display

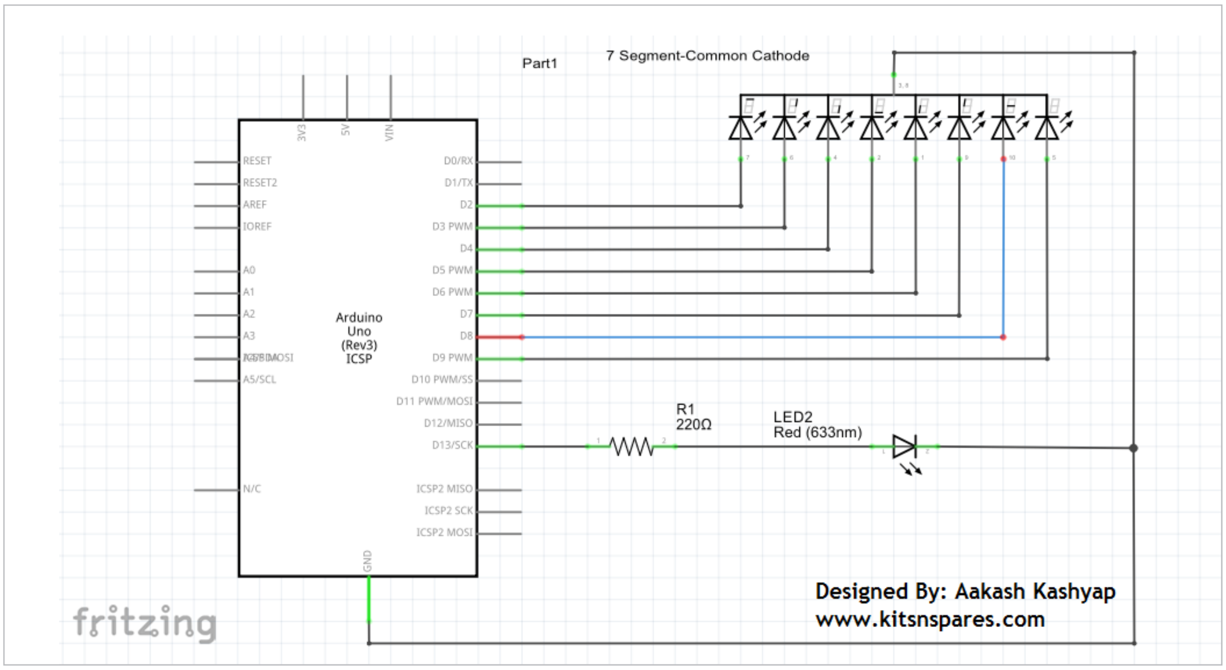
A 7-segment display (SSD) or 7-segment indicator is an electronic display device for displaying decimal numerals, as an alternative to more complex dot-matrix displays. 7-segment displays are widely used in digital clocks, electronic meters, basic calculators and other electronic devices that display numerical information.
There are two types of 7-segment displays:

- Common Anode (CA) Display - In a CA display, all the anode connections of the LED segments are joined together to logic "1". The individual segments are illuminated by applying a ground, logic "0" or "LOW" signal via a suitable current limiting resistor to the cathode of the particular segment (a-g). In general, CA displays are more popular as many logic circuits can sink more current than they can source.



- Common Cathode (CC) Display: In a CC display, the individual segments are illuminated by applying logic "1" or "HIGH" signal. Depending upon the decimal digit to be displayed, the particular set of LEDs is forward-biased.





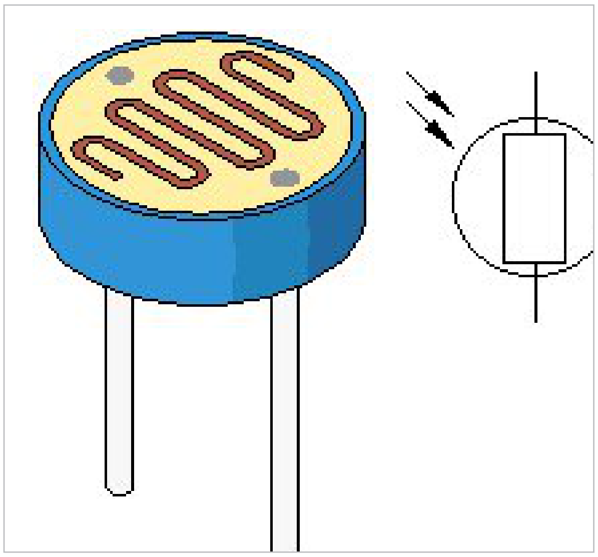
Lux Meter Using LDR

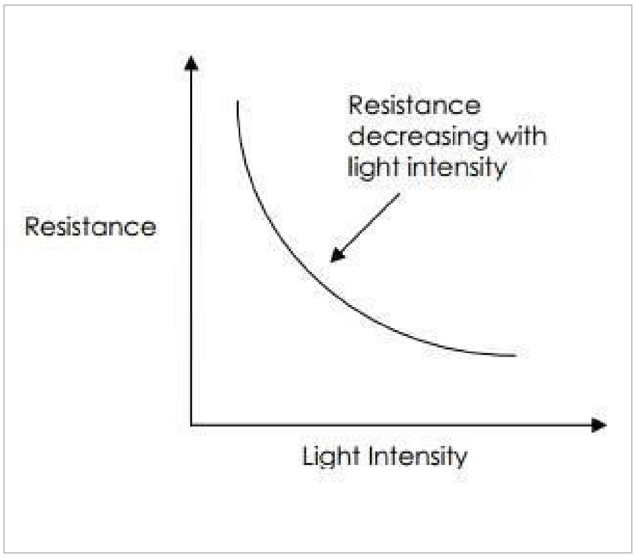
What we will do:

We will interface a photodiode (LDR) to the Arduino and make a lux meter using it.

Lux Meter: This is a device that measures illuminance and luminous emittance using the SI unit of lux. It effectively measures the amount of power from the light falling on a given unit of area, except that the power measurement is weighted to reflect the sensitivity of the human eye to varying wavelengths of light. A simpler way to describe a lux meter is to say that it measures how bright the light falling on the sensor is. Commercially available lux meters vary in price from around US$ 15 up to hundreds of dollars, but it is cheaper and much more fun to build one yourself.

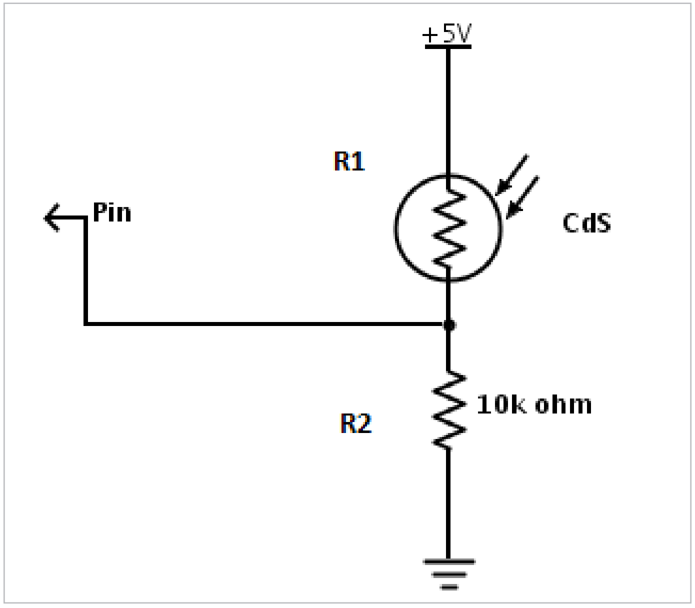
Light-Dependent Resistor (LDR): An LDR or photo resistor is a light-controlled variable resistor. The resistance of a photo resistor decreases with increasing incident light intensity, that is, it exhibits photoconductivity. It can be used in light-sensitive detector circuits and light- and dark-activated switching circuits.

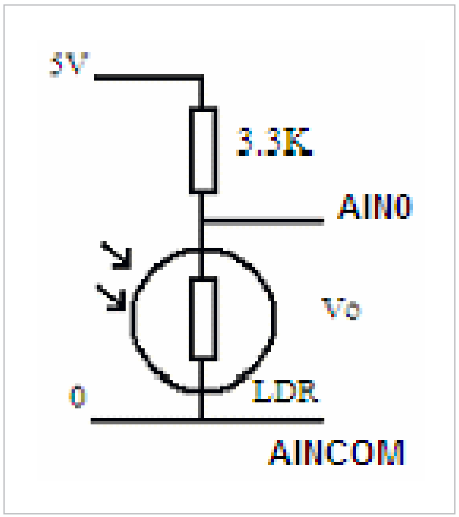




Since a microcontroller or microprocessor cannot directly measure the change in the resistance we use a voltage divider circuit, wherein the change in voltage corresponds to a change in the resistance of the LDR, according to the light intensity.

Vout = Vin* (R2/R1 + R2)





The resistance of the LDR varies according to the amount of light that falls on it. The relationship between the resistance (RL) and light intensity (Lux) for a typical LDR is:

$$R_L = \frac{500}{Lux}$$

If the LDR is connected to 5V through a 3.3K resistor, using the voltage divider rule, the output voltage of the LDR is:

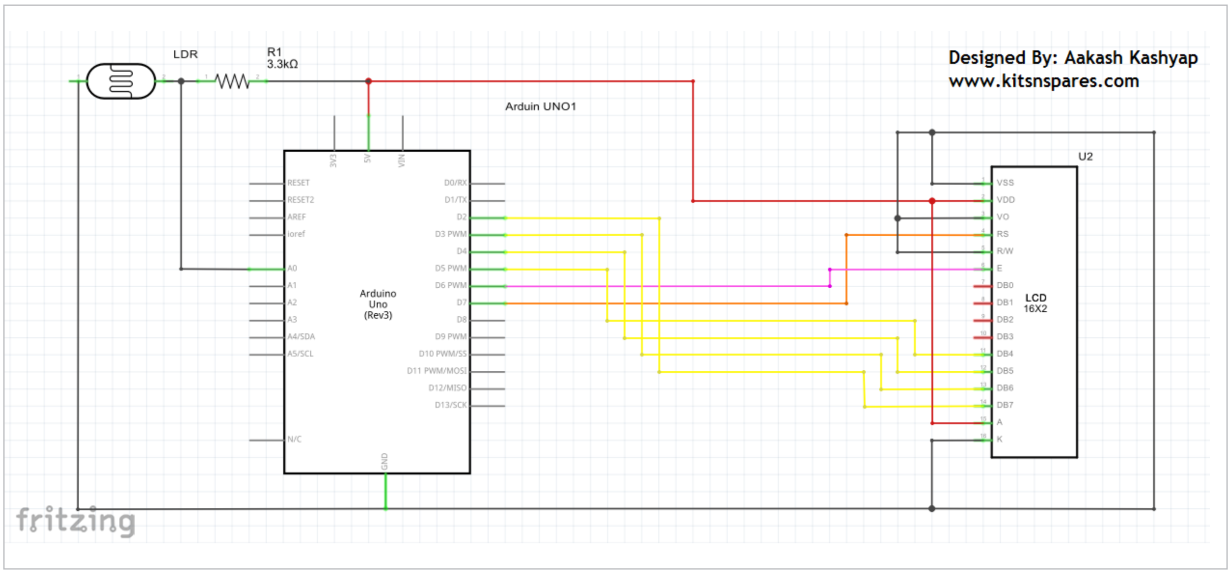
$$V_o = 5 \times \frac{R_L}{R_L + 3.3}$$

Substituting RL from equation 1 into equation 2, we obtain the light intensity:

$$Lux = \frac{\frac{2500}{V_o} - 500}{3.3}$$

For a low-cost LDR, at the same light intensity, the part-to-part variation in resistance can be as high as 50 per cent. Therefore, such a low-cost LDR is seldom used for measuring light intensity but more for light presence/absence detection.

Circuit Diagram:



Interfacing an LM35 Temperature Sensor

What we will do:

We will connect a temperature sensor and an LCD, to display the temperature value on the screen.

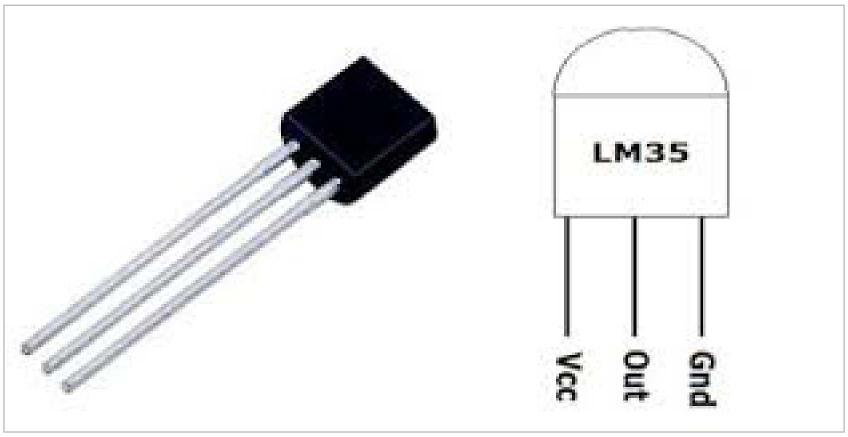
LM35: The LM35 is an integrated circuit sensor that can be used to measure temperature with an electrical output proportional to the temperature (in oC)

What is the LM35 and what does it do?

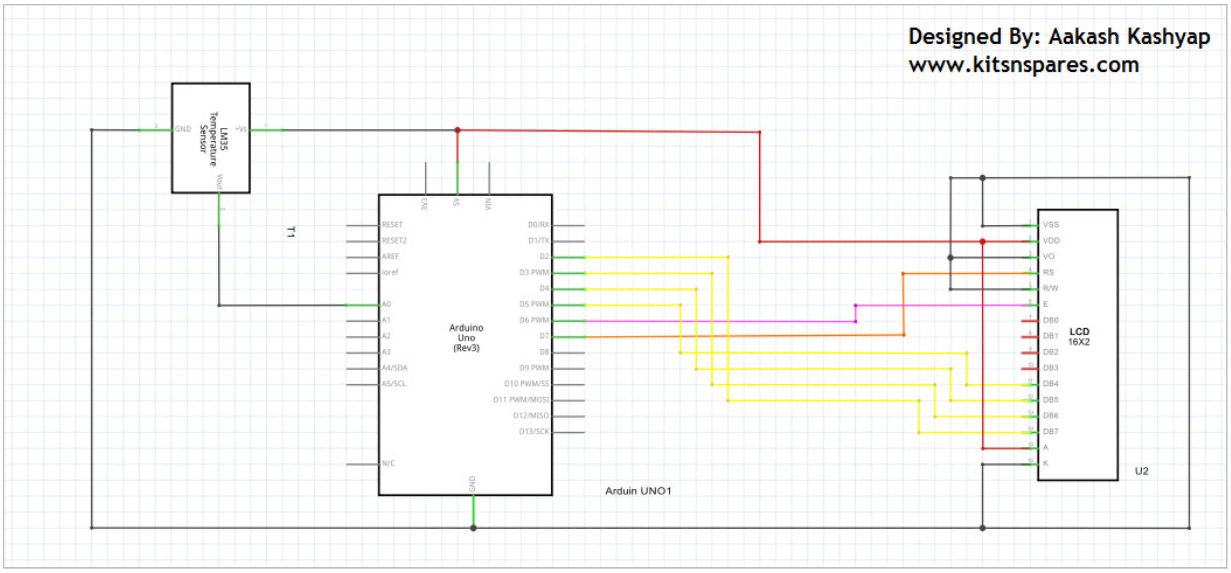
It has an output voltage that is proportional to the temperature (in degrees Celsius) The scale factor is .01V/oC

The LM35 does not require any external calibration or trimming and maintains an accuracy of +/- 0.4 oC at room temperature and +/- 0.8 oC over a range of 0 oC to +100 oC

The LM35DZ draws only 60 micro amps from its supply and possesses a low self-heating capability. The sensor self-heating causes less than 0.1 oC temperature rise in still air.



Circuit Diagram:



Interfacing a Passive Infrared Motion Sensor with Buzzer

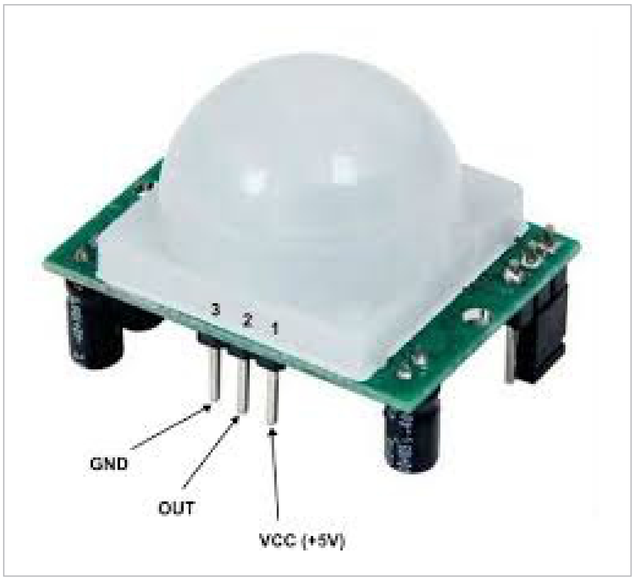
What we will do:
We will connect a PIR motion sensor and buzzer to the Arduino. If any motion is detected, the buzzer will turn on for some time.

Passive infrared sensor (PIR): This is an electronic sensor that measures infrared (IR) light radiating from objects in its field-of-view. They are most often used in PIR-based motion detectors.

All objects with a temperature above absolute zero emit heat energy in the form of radiation. Usually this radiation is invisible to the human eye because it radiates at infrared wavelengths, but it can be detected by electronic devices designed for such a purpose.

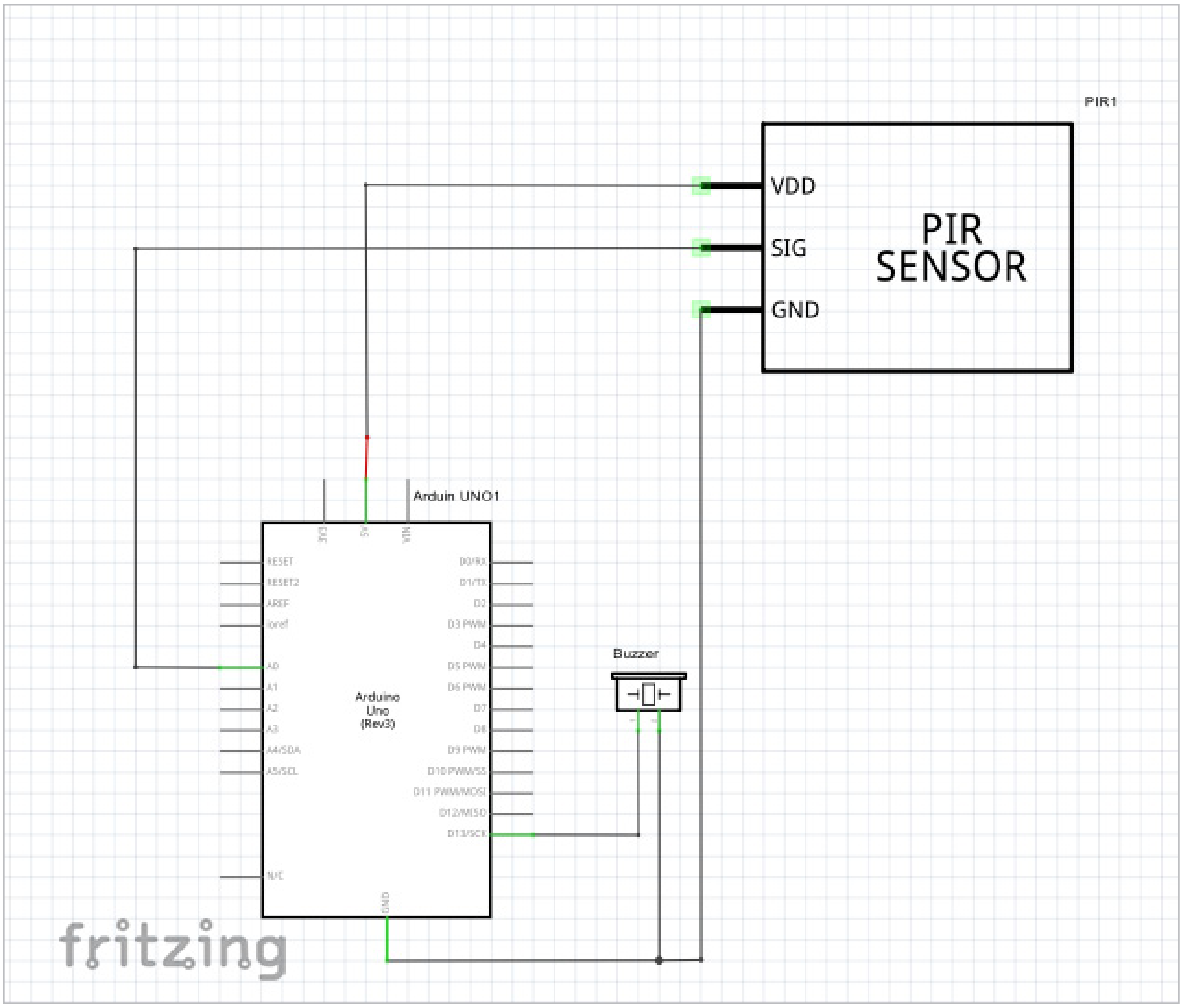
The use of the term passive indicates that PIR devices do not generate or radiate any energy for detection purposes. They work entirely by detecting the energy given off by other objects. They do not detect or measure heat – instead, they detect the infrared radiation emitted or reflected from an object.

PIR-based motion detectors are used to sense the movement of people, animals or other objects. They are commonly used in burglar alarms and automatically-activated lighting systems. They are commonly called as PIR, or sometimes as PID, which stands for passive infrared detector.



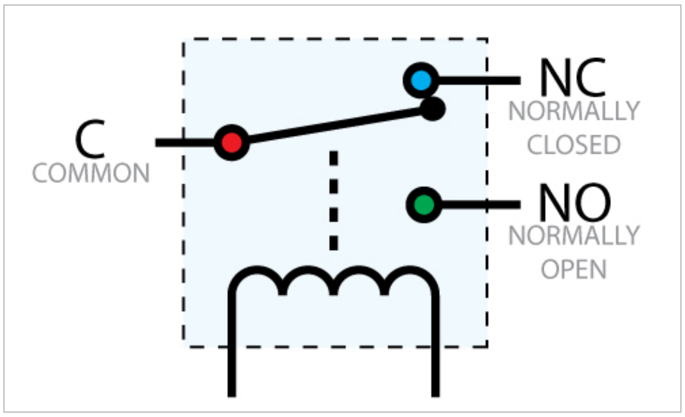
Buzzer: A buzzer or beeper is an audio signalling device, which may be mechanical, electromechanical or piezoelectric. They are commonly used in alarm devices, timers and for confirmation of user input such as a mouse click or keystroke.
In this project, we have used a piezoelectric buzzer. A piezoelectric buzzer is based on the inverse principle of piezoelectricity discovered in 1880 by Jacques and Pierre Curie. It is the phenomenon of generating electricity when mechanical pressure is applied to certain materials, and vice versa. Such materials are called piezoelectric materials.

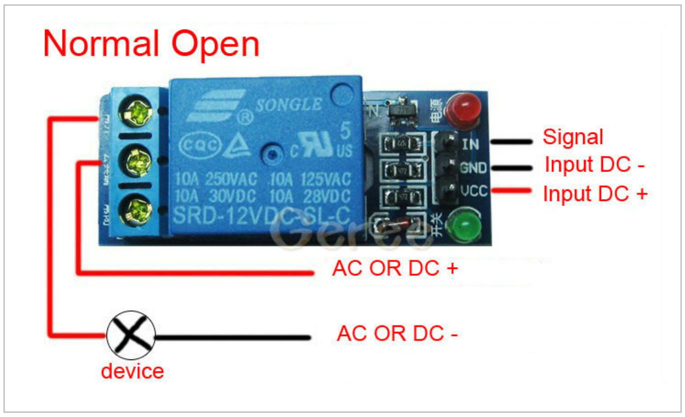
Circuit Diagram:



Interfacing a Relay

A relay is an electrically-operated switch. Many relays use an electromagnet to mechanically operate a switch. Other operating principles are also used, such as in solid-state relays. Relays are used to control a circuit with a low-power signal (with complete electrical isolation between control and controlled circuits), or to control several circuits by one signal.

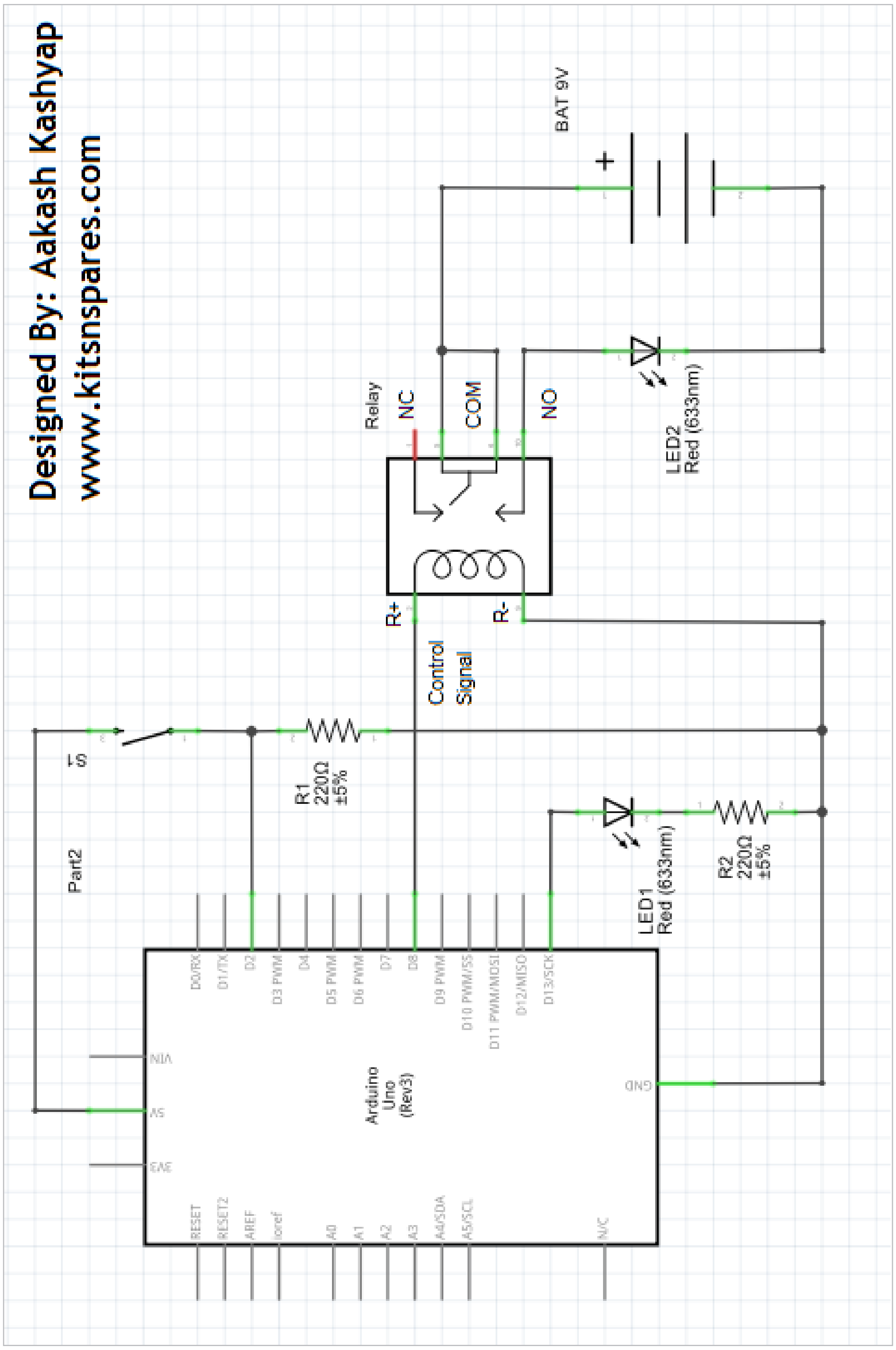




A ground connection is connected to common and the live wire is connected to NO or NC depending upon the application. The supply signal is applied to the coils. When the coils get energised, it pulls the lever towards the normally open condition.

We are going to use a relay module in this project, and it is very easy to connect. Just connect the input channels to one of the digital pins of the Arduino, connect the VCC and GND of the module to 5V and the GND of the Arduino and you are all set to go!

Circuit Diagram:



Interfacing an Ultrasonic Module

What we will do:

We will connect an ultrasonic sensor and an LCD to measure the distance up to an object.

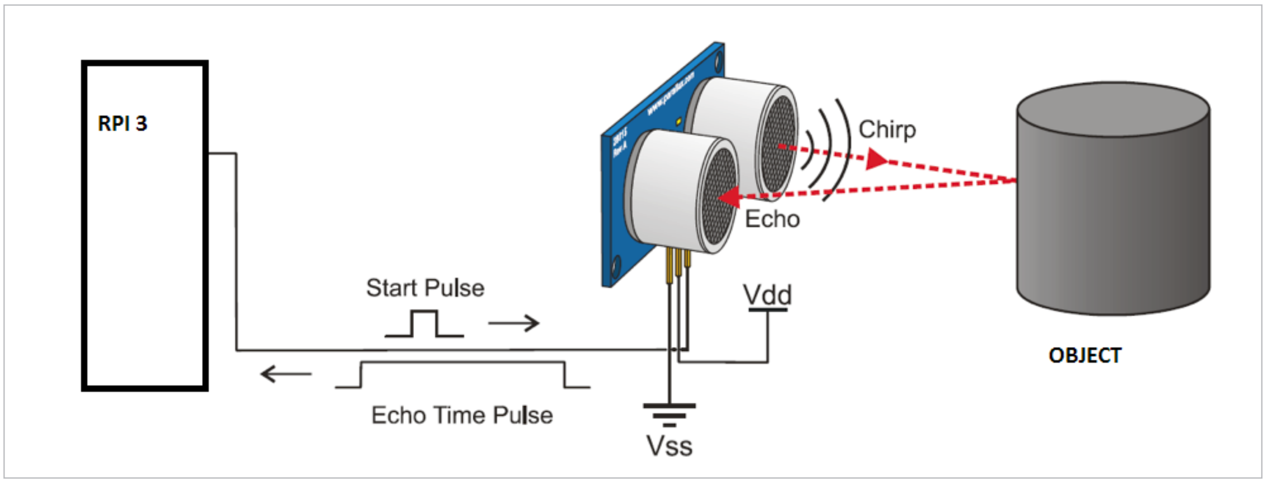
The HC-SR04 ultrasonic sensor uses sonar to determine the distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. It has a range of 2cm to 400cm, or 1 inch to 13 feet. Its operation is not affected by sunlight or black material like Sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module.

Working Principle:

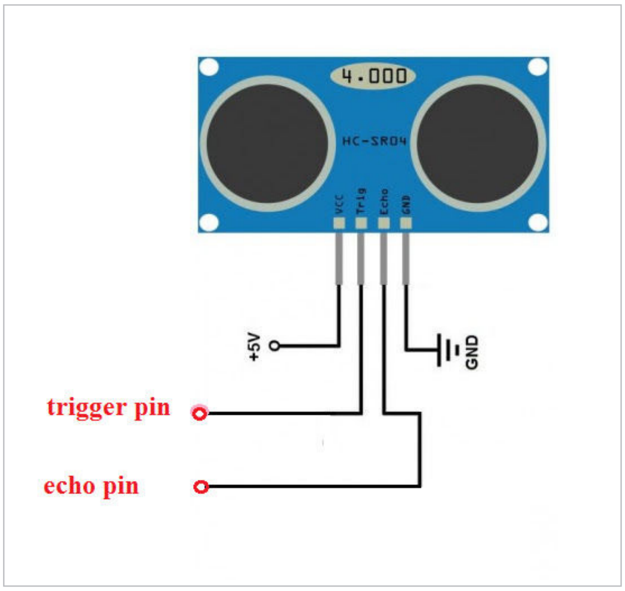
The ultrasonic transmitter emits ultrasound signals, which travel towards the object in front of it and get reflected back. The timing of both transmission and receiving of signals are calculated and then divided by 2 to get the time taken by the ultrasound to reach the object.
We know that the speed of sound in air is 340 m/s and distance can be calculated by the formula:

Distance = Speed * Time.

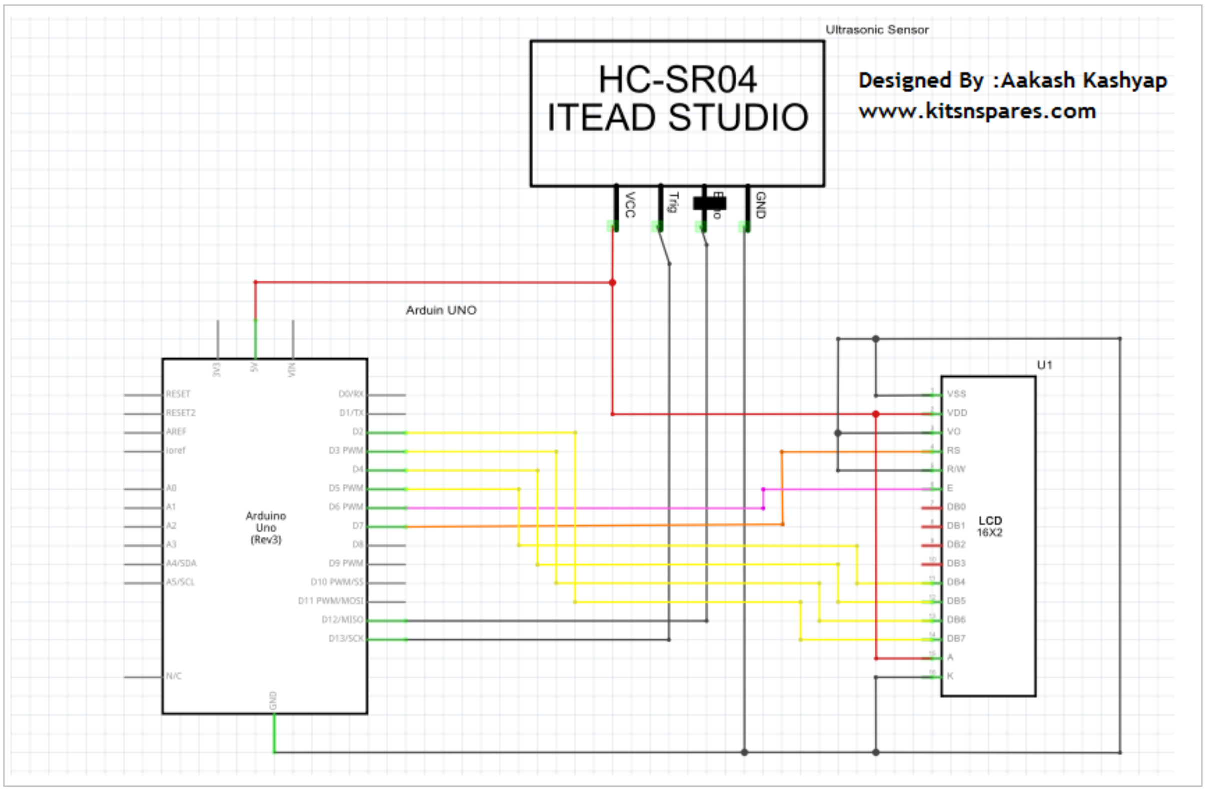
Knowing the time and the speed of sound, we can easily calculate the distance.



PIN DIAGRAM



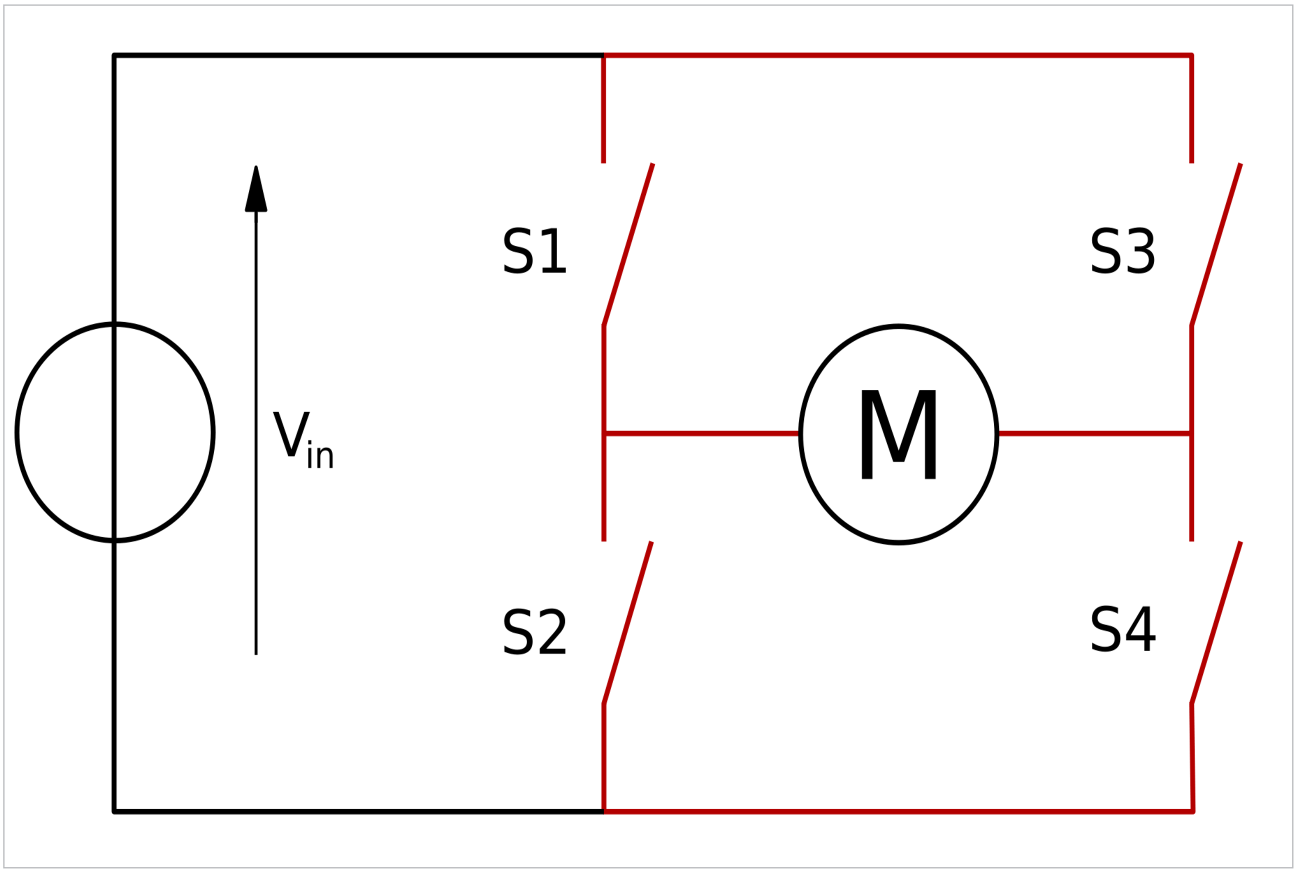
Circuit Diagram



Interfacing L293D Motor Driver

L293D is a typical motor driver or motor driver IC, which allows a direct current (DC) motor to drive on either direction. L293D is a 16-pin IC that can control a set of two DC motors simultaneously in any direction. It means that you can control two DC motors with a single L293D IC.

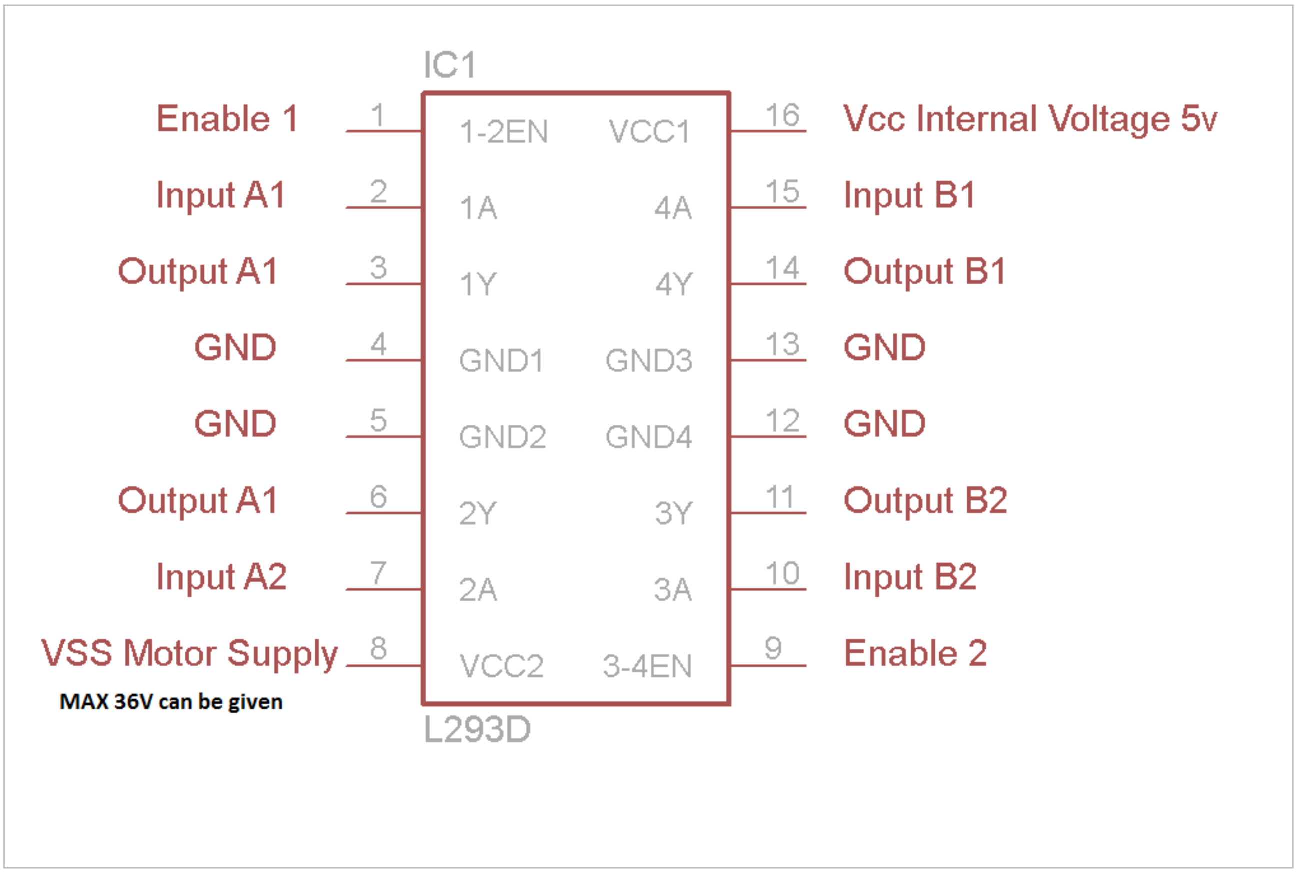
It works on the concept of H-bridge. An H-bridge is an electronic circuit that enables a voltage to be applied across a load in either direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards and backwards.



There are four inputs to this IC, which can be used to control two DC motors in any direction. The direction of motors depends upon the input voltage. Here is the logic table, which will help you to decide how to set up your motors.

Input1	Input2	Input3	Input4	Motor State
1	0	0	1	Clockwise rotation
0	1	1	0	Anticlockwise Rotation
0	0	0	0	Idle [High Impedence State]
1	1	1	1	Idle

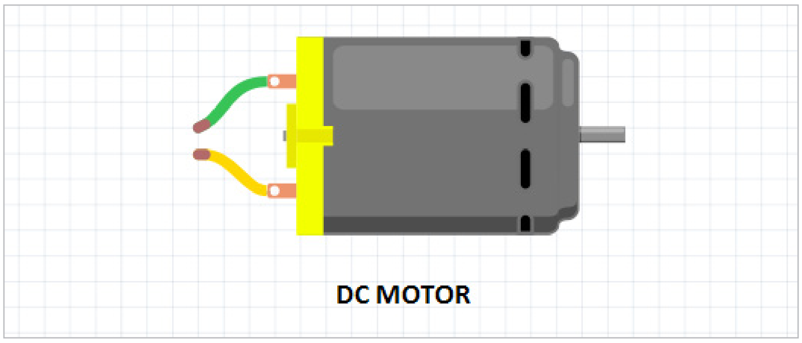
Pin Diagram:



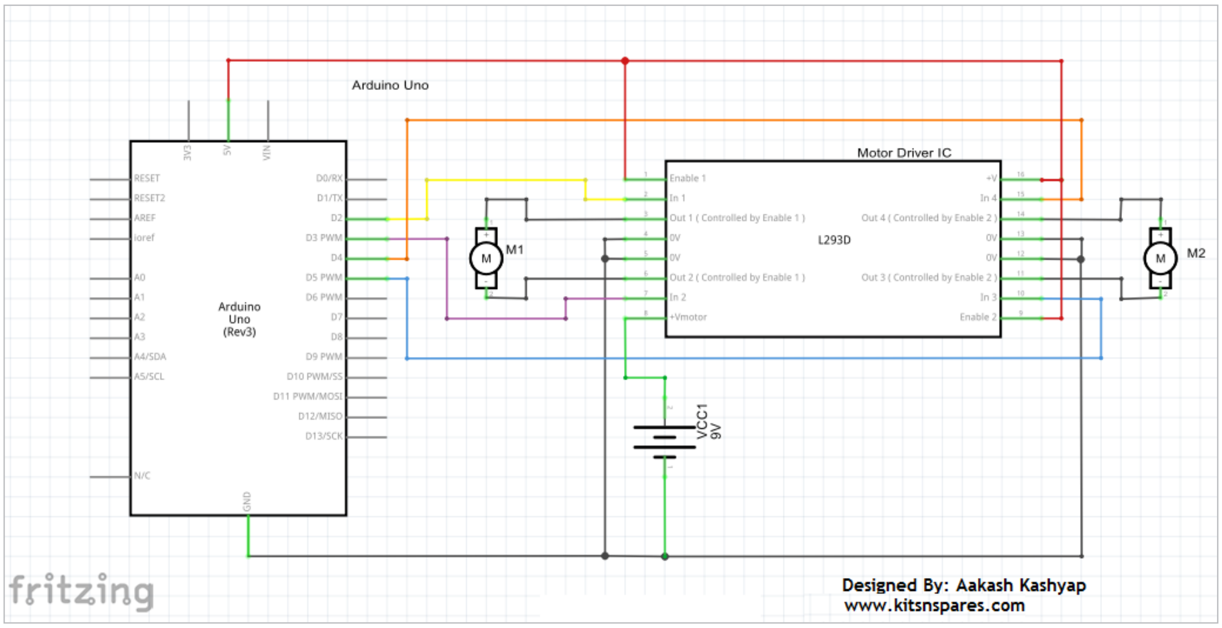
For this project, we will use two DC motors.

A DC motor is the most common type of motor. It normally has just two leads, one positive and one negative. If you connect these two leads directly to a battery, the motor will rotate. If you switch the leads, the motor will rotate in the opposite direction.

To control the direction of spin of the DC motor, without changing the way the leads are connected, you can use a circuit called an H-bridge. An H-bridge is an electronic circuit that can drive the motor in both directions.



Circuit Diagram:



Tips for Debugging and Efficient Project Building

As we near the end of this self-learning manual, we present you with some tips that will help you build your project faster and without errors.

- While connecting wires always use colour codes, such as red wire for power and black for GND. This will really help you debug your projects faster.
- Always check the polarity (+/-) to prevent damage to the IC or development board.
- Use proper value of resistance to limit current to your IC or development board.
- Before powering up the project, make sure that you check all your connections again.
- Use a multi-meter and perform a continuity check, if you feel some components are not working.

Now that you have learnt how to interface some of most commonly used sensors with Arduino, and are also armed with a few tips to build your project quickly and without errors, it is time to kindle your creativity and start building a good project, which will be useful to the society. Get started right away! We are eager to see your cool projects.

NOTES.

NOTES.

NOTES.