

Chapter 4 Notes

- 4.1 Introduction of Arduino board Layout with pin function & Detail of IDE of Arduino Software. Compare different Arduino boards: Arduino UNO, NANO, MINI, and MEGA.
- 4.2 Functions used in Arduino: Math, Analog I/O, Digital I/O, Timer.
- 4.3 Peripheral interfacing with Arduino : Keypad, LCD, Seven segment LED, Relay, Stepper Motor, DC Motor, (connection diagram, working & programming). .
- 4.4 Sensor Interfacing with Arduino: LM35 Temperature sensor and Ultrasonic sensor (connection diagram, working & programming.)

4.1 Introduction of Arduino board Layout with pin function & Detail of IDE of Arduino Software. Compare different Arduino boards: Arduino UNO, NANO, MINI, and MEGA

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are –

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Arduino Boards

A wide range of boards is manufactured by Arduino. These have different sizes, different microcontrollers, and different processing capabilities.

There are entry level boards like the UNO, LEONARDO, NANO etc; boards with enhanced feature like the MEGA, PRO, ZERO etc; boards for Internet of Things like the YUN, TIAN etc; and wearable boards like the LILYPAD, GEMMA etc.

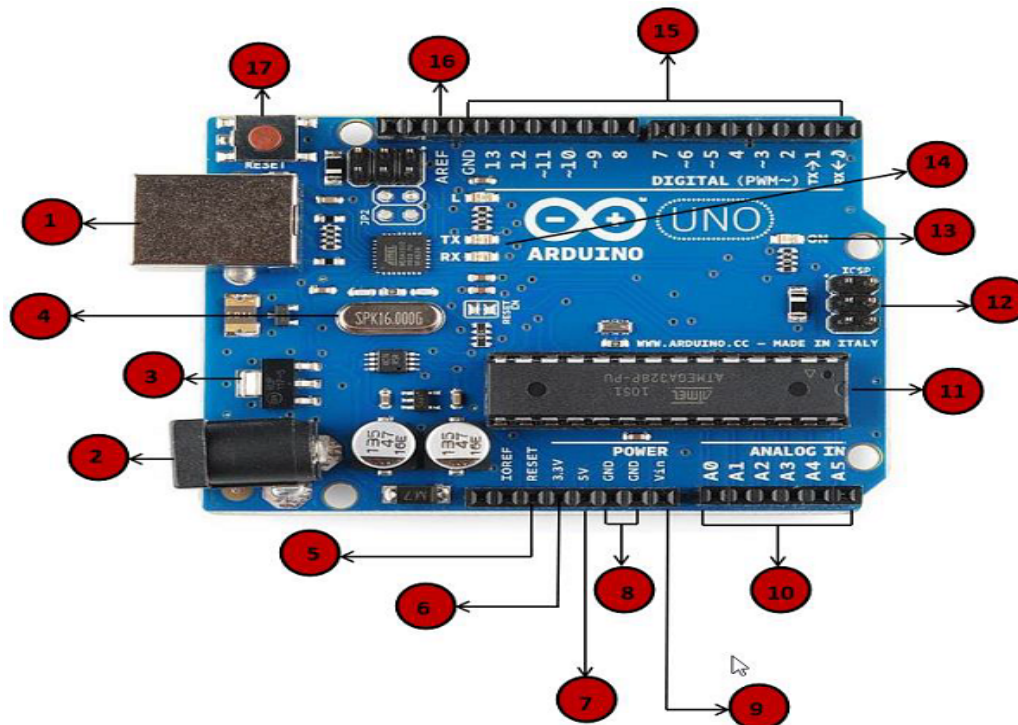
Depending on the need of application and the processing requirements, users can choose from any of these.







Comparison of different boards







Boards	Microcontroller	Operating Voltage/s (V)	Digital I/O Pins	PWM Enabled Pins	Analog I/O Pins	DC per I/O (mA)	Flash Memory (KB)	SRAM (KB)	EEPROM (KB)	Clock (MHz)	Length (mm)	Width (mm)	Cable
Uno	ATmega328	5	14	6	6	20	32	2	1	16	68.6	53.4	USB A-B
Leonardo	ATmega32u4	5	20	7	12	40	32	2.5	1	16	68.6	53.3	micro-USB
Micro	ATmega32u4	5	20	7	12	40	32	2.5	1	16	48	18	micro-USB
Nano	ATmega328	5	22	6	8	40	32	2	0.51	16	45	18	mini-B USB
Mini	ATmega328	5	14		6	20	32	2	1	16	30	18	USB-Serial
Due	Atmel SAM3X8E ARM Cortex-M3 CPU	3.3	54	12	12	800	512	96	X	84	102	53.3	micro-USB
Mega	ATmega2560	5	54	15	16	20	256	8	4	16	102	53.3	USB A-B

Arduino Uno Layout

Arduino UNO board is the most popular board in the Arduino board family. In addition, it is the best board to get started with electronics and coding. Some boards look a bit different from the one given below, but most Arduinos have the majority of these components in common.



	<p>Power USB</p> <p>Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).</p>
	<p>Power (Barrel Jack)</p> <p>Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).</p>
	<p>Voltage Regulator</p> <p>The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.</p>
	<p>Crystal Oscillator</p> <p>The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.</p>
	<p>Arduino Reset</p> <p>You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).</p>
	<p>Pins (3.3, 5, GND, Vin)</p> <p>3.3V (6) – Supply 3.3 output volt 5V (7) – Supply 5 output volt Most of the components used with Arduino board works fine with 3.3 volt and 5 volt. GND (8)(Ground) – There are several GND pins on the Arduino, any of which can be used to ground your circuit. Vin (9) – This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.</p>

	<p>Analog pins</p> <p>The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.</p>
	<p>Main microcontroller</p> <p>Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.</p>
	<p>ICSP pin</p> <p>Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.</p>
	<p>Power LED indicator</p> <p>This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.</p>
	<p>TX and RX LEDs</p> <p>On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.</p>
	<p>Digital I/O</p> <p>The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled “~” can be used to generate PWM.</p>

16

AREF

AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Arduino IDE Features

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

The main features are

Board Module Options: The tool is armed with a board management module, wherein users can choose which board they want to use.

Direct Sketching : Arduino IDE lets users to come up with sketches from within its text editor. The process is simple and straightforward.

Documentation: The tool gives users an option to have their projects documented. The feature makes it possible for them to track their progress and be aware of any changes made.

Sketch Sharing: Arduino IDE allows users to share their sketches to other programmers. Each sketch comes with their own online link for users to share with their colleagues or friends. This feature is only available in the cloud version.

Integrated Libraries: The software has hundreds of integrated libraries. These libraries were made and openly shared by the Arduino community.

External Hardware Support: While the tool itself is specifically intended for Arduino boards, it also has native connection support for third-party hardware.

4.2 Functions used in Arduino: Math, Analog I/O, Digital I/O, Timer.**Setup and Loop**

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

setup : It is called only when the Arduino is powered on or reset. It is used to initialize variables and pin modes

loop : The loop functions run continuously till the device is powered off. The main logic of the code goes here. Similar to while (1) for microcontroller programming.

Digital I/O functions**1. pinMode**

Configures the specified pin to behave either as an input or an output. To configure a pin as INPUT, either INPUT or INPUT_PULLUP may be used. it is possible to enable the internal

pullup resistors with the mode `INPUT_PULLUP`. The `INPUT` mode explicitly disables the internal pullups.

Syntax: `pinMode(pin, mode)`

pin: the Arduino pin number to set the mode of.

mode: `INPUT`, `OUTPUT`, or `INPUT_PULLUP`.

Example:

```
pinMode(13, OUTPUT); // sets pin 13 as output pin
```

```
pinMode(13, INPUT); // sets pin 13 as input pin
```

2. DigitalWrite

`digitalWrite(pin, value)`

pin: the Arduino pin number.

value: `HIGH` or `LOW`.

Description

Write 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 pullup on the input pin.

3. digitalRead

Reads the value from a specified digital pin, either `HIGH` or `LOW`.

Syntax : `digitalRead(pin)`

pin: the Arduino pin number you want to read

Example Code

```
val = digitalRead(7); // read the input pin D7 to the variable val
```

Analog I/O Functions

1. analogRead()

Description

Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage (5V or 3.3V) into integer values between 0 and 1023. On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV)

On ATmega based boards (UNO, Nano, Mini, Mega), it takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax : `analogRead(pin)`

Parameters

pin: the name of the analog input pin to read from (A0 to A5 on most boards, A0 to A6 on MKR boards, A0 to A7 on the Mini and Nano, A0 to A15 on the Mega).

Returns

The analog reading on the pin. Although it is limited to the resolution of the analog to digital converter (0-1023 for 10 bits or 0-4095 for 12 bits). Data type: `int`.

Example Code

The code reads the voltage on `analogPin` and displays it.

```
int analogPin = A3; // potentiometer wiper (middle terminal) connected to analog pin 3
```

```
int val = 0; // variable to store the value read
void setup() {
  val = analogRead(analogPin); // read the input pin
}
```

2. analogWrite

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to `analogWrite()`, the pin will generate a steady rectangular wave of the specified duty cycle until the next call to `analogWrite()` (or a call to `digitalRead()` or `digitalWrite()`) on the same pin.

Syntax : `analogWrite(pin, value)`

pin: the Arduino pin to write to. Allowed data types: `int`.

value: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: `int`.

Example Code

Sets the output to the LED proportional to the value read from the potentiometer.

```
int ledPin = 9; // LED connected to digital pin 9 (PWM pin)
int val = 0; // variable to store the read value
void setup() {
  pinMode(ledPin, OUTPUT); // sets the pin as output
}
void loop() {
  analogWrite(ledPin, 64); // analogWrite values from 0 to 255
}
```

3. analogReference()

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

Arduino AVR Boards (Uno, Mega, Leonardo, etc.)

- **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 ATmega328P and 2.56 volts on the ATmega32U4 and 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)`

type: which type of reference to use (as given above)

TIME Functions

1. delay()

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

Syntax : `delay(ms)`

ms: the number of milliseconds to pause. Allowed data types: unsigned long.

Example Code

The code pauses the program for one second before toggling the output pin.


```

int ledPin = 13;           // LED connected to digital pin 13
void setup() {
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}
void loop() {
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000);                // waits for a second
  digitalWrite(ledPin, LOW);  // sets the LED off
  delay(1000);                // waits for a second
}

```

2. delayMicroseconds()

Pauses the program for the amount of time (in microseconds) specified by the parameter.

Syntax : delayMicroseconds(us)

us: the number of microseconds to pause. Allowed data types: unsigned int.

3. micros()

Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes.

Syntax : time = micros()

4. millis()

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

Syntax : time = millis()

Math Functions in Arduino

1. abs()

Description:

Calculates the absolute value of a number.

Syntax:

abs(x)

Parameters:

x: the number

Returns:

x: if x is greater than or equal to 0.

-(x): if x is less than 0.

2. constrain()

Description : Constrains a number to be within a range.

Syntax : constrain(x, a, b)

Parameters:

x: the number to constrain Allowed data types: all data types.

a: the lower end of the range. Allowed data types: all data types.

b: the upper end of the range. Allowed data types: all data types.

Returns:

x: if x is between a and b.

- a: if x is less than a.
- b: if x is greater than b.

3. max()

Description : Calculates the maximum of two numbers.

Syntax : max(x, y)

Parameters

x: the first number. Allowed data types: any data type.

y: the second number. Allowed data types: any data type.

Returns : The larger of the two parameter values.

4. min()

Description : Calculates the minimum of two numbers.

Syntax : min(x, y)

Parameters

x: the first number. Allowed data types: any data type.

y: the second number. Allowed data types: any data type.

Returns : The smaller of the two numbers.

5. sq()

Description

Calculates the square of a number: the number multiplied by itself.

Syntax

sq(x)

Parameters

x: the number. Allowed data types: any data type.

Returns

The square of the number.

6. sqrt()

Description

Calculates the square root of a number.

Syntax

sqrt(x)

Parameters

x: the number. Allowed data types: any data type.

Returns

The number's square root. Data type: double.

7. pow()

Description

Calculates the value of a number raised to a power. pow() can be used to raise a number to a fractional power. This is useful for generating exponential mapping of values or curves.

Syntax

pow(base, exponent)

Parameters

base: the number. Allowed data types: float.

exponent: the power to which the base is raised. Allowed data types: float.

Returns

The result of the exponentiation. Data type: double.

8. **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.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The constrain() function may be used either before or after this function, if limits to the ranges are desired.

Note that the "lower bounds" of either range may be larger or smaller than the "upper bounds" so the map() function may be used to reverse a range of numbers, for example

y = map(x, 1, 50, 50, 1);

4.3 Peripheral interfacing with Arduino : Keypad, LCD, Seven segment LED, Relay, Stepper Motor, DC Motor, (connection diagram, working & programming)

1. LCD

Pin out

Pin No.	Symbol	Function
1	V _{SS}	Ground
2	V _{DD}	Power supply
3	V ₀	Power Supply for LCD
4	RS	Select Display Data("H") or Instructions("L")
5	R/W	Read or Write Select Signal
6	E	Read/Write Enable Signal
7	DB0	Display Data Signal
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	LED – (K)	Please also refer to 6.1 PCB drawing and description.
16	LED + (A)	Please also refer to 6.1 PCB drawing and description.

The use of pins listed below

GND(VSS) : Connect the ground pin of the power supply to this pin.

VCC : Connect this pin to 5v

Contrast (VEE) : This pin is used to adjust the contrast of Display. Connect a potentiometer (POT) to this pin. Rotate the knob of the POT to adjust the contrast.

RS : RS pin means Register select pin. Selects the command register when the pin is LOW. And select the data register when this pin is HIGH.

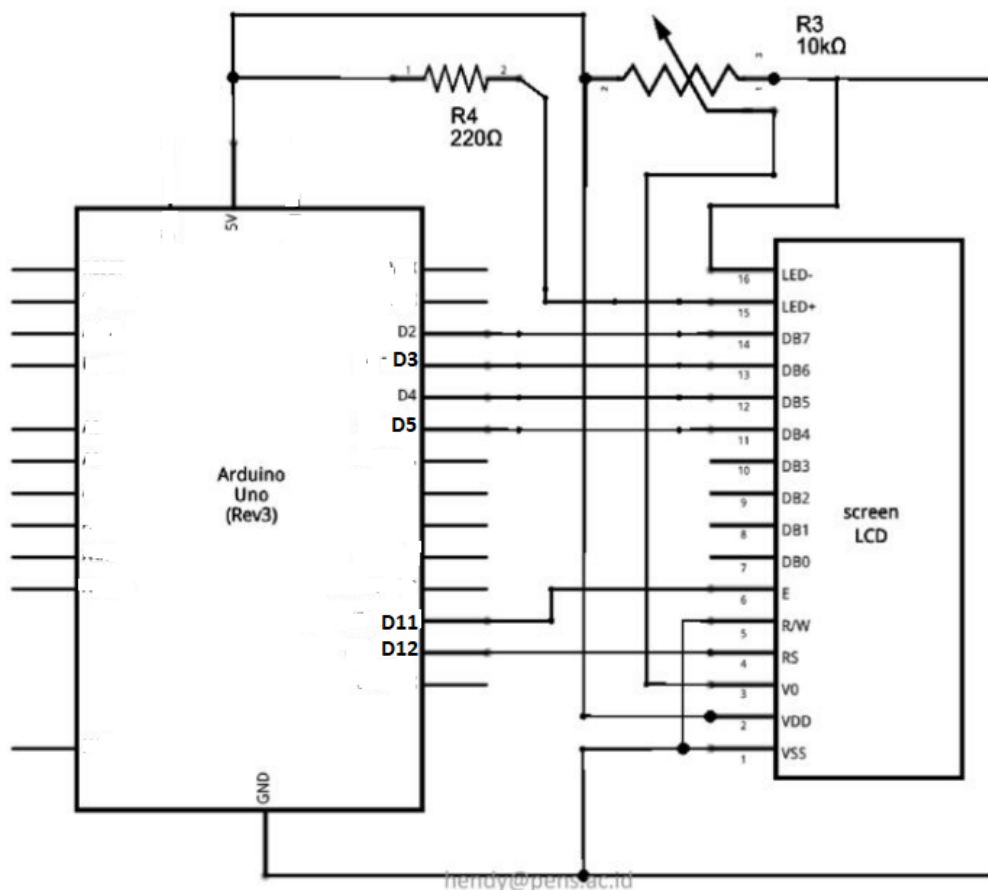
RW : It represents the Read Write pin. When this pin is LOW, the MCU writes to register. And when the pin is HIGH, MCU reads from the register. Here we want to write. It connects it permanently to GND.

EN (E) : EN pin means the Enable pin. Send data to data pins when a HIGH to LOW pulse is given.

D0-D7 (DB0-DB7) : These are 8 data pins.

Backlight(+) : This is the anode pin of the backlight of the display

Backlight(-) : This is the cathode pin of the backlight of the display



```
#include<LiquidCrystal.h> // include the library code:
```

```
liquidCrystal lcd(12, 11, 5, 4, 33, 2); // initialize the library with the numbers of the interface pins
```

```
void setup()
```

```
{
```

```
  lcd.begin(16, 2); // set up the LCD's number of columns and rows:
```

```
  lcd.clear(); // Clear the LCD
```

```
}
```

```
void loop()
```

```
{
```

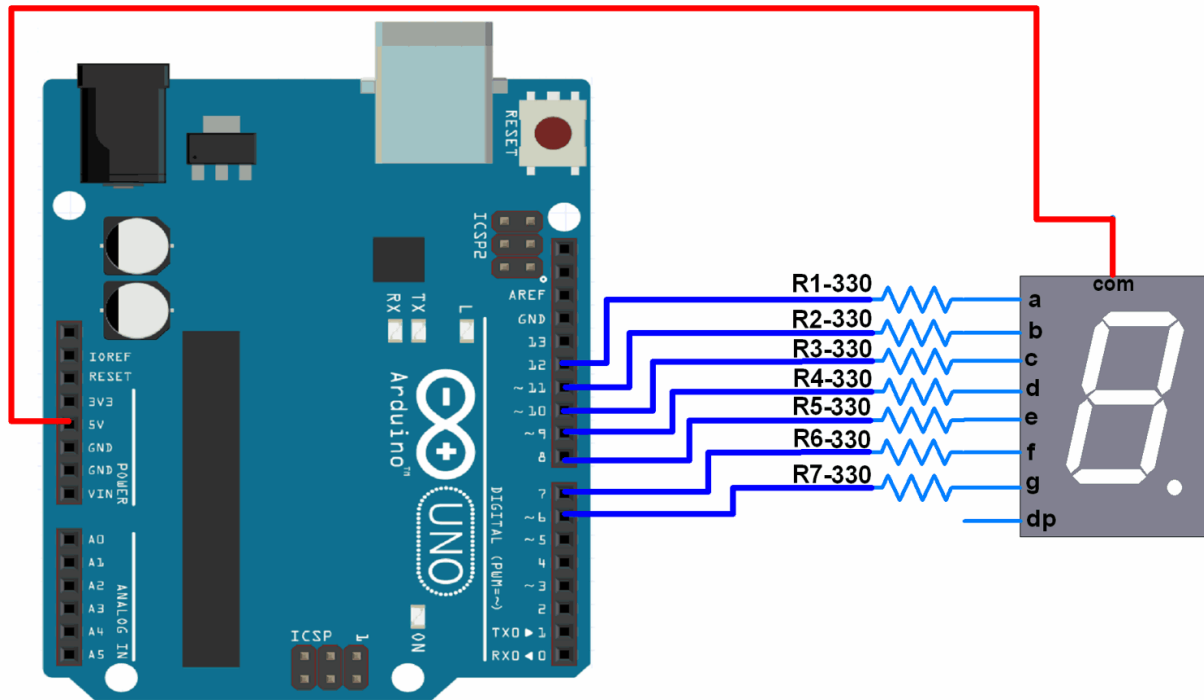
```
  lcd.setCursor(0, 0); // set the cursor to column 0, line 0 /
```

```

lcd.print(" MSBTE ");
lcd.setCursor(0,1); //set the cursor to column 0, line 1
lcd.print(" VESP ");
}

```

2. Seven segment LED



```

int a = 12; //Arduino pins connected with 7 segment pins
int b = 11;
int c = 10;
int d = 9;
int e = 8;
int f = 7;
int g = 6;

```

```

void setup() {
//Declaring all the pins as output
pinMode(a, OUTPUT);   pinMode(b, OUTPUT);   pinMode(c, OUTPUT);
pinMode(d, OUTPUT);   pinMode(e, OUTPUT);   pinMode(f, OUTPUT);
pinMode(g, OUTPUT);
}

```

```

// the loop routine runs over and over again forever:
void loop() {

```

```

digitalWrite(a, HIGH);
digitalWrite(b, HIGH);
digitalWrite(c, HIGH);
digitalWrite(d, HIGH);    //Generating 1
digitalWrite(e, LOW);
digitalWrite(f, LOW);
digitalWrite(g, HIGH);
delay(1000);              // wait for a second
digitalWrite(a, LOW);
digitalWrite(b, LOW);
digitalWrite(c, HIGH);    //Generating 2
digitalWrite(d, LOW);
digitalWrite(e, LOW);
digitalWrite(f, HIGH);
digitalWrite(g, LOW);
delay(1000);              // wait for a second
digitalWrite(a, LOW);
digitalWrite(b, LOW);
digitalWrite(c, LOW);
digitalWrite(d, LOW);    //Generating 3
digitalWrite(e, HIGH);
digitalWrite(f, HIGH);
digitalWrite(g, LOW);
delay(1000);              // wait for a second
digitalWrite(a, HIGH);
digitalWrite(b, LOW);
digitalWrite(c, LOW);
digitalWrite(d, HIGH);
digitalWrite(e, HIGH);    //Generating 4
digitalWrite(f, LOW);
digitalWrite(g, LOW);
delay(1000);              // wait for a second
digitalWrite(a, LOW);
digitalWrite(b, HIGH);
digitalWrite(c, LOW);
digitalWrite(d, LOW);
digitalWrite(e, HIGH);    //Generating 5
digitalWrite(f, LOW);
digitalWrite(g, LOW);
delay(1000);              // wait for a second
digitalWrite(a, LOW);
digitalWrite(b, HIGH);
digitalWrite(c, LOW);
digitalWrite(d, LOW);    //Generating 6
digitalWrite(e, LOW);

```

```

digitalWrite(f, LOW);
digitalWrite(g, LOW);
delay(1000);          // wait for a second
digitalWrite(a, LOW);
digitalWrite(b, LOW);
digitalWrite(c, LOW);
digitalWrite(d, HIGH);
digitalWrite(e, HIGH); //Generating 7
digitalWrite(f, HIGH);
digitalWrite(g, HIGH);
delay(1000);          // wait for a second
digitalWrite(a, LOW );
digitalWrite(b, LOW);
digitalWrite(c, LOW);
digitalWrite(d, LOW);
digitalWrite(e, LOW);  //Generating 8
digitalWrite(f, LOW);
digitalWrite(g, LOW);
delay(1000);          // wait for a second
digitalWrite(a, LOW);
digitalWrite(b, LOW);
digitalWrite(c, LOW);
digitalWrite(d, HIGH);
digitalWrite(e, HIGH); //Generating 9
digitalWrite(f, LOW);
digitalWrite(g, LOW);
delay(1000);          // wait for a second
}

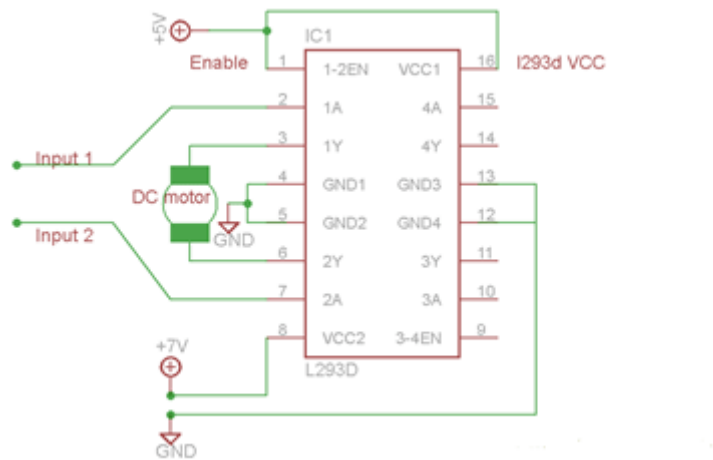
```

3. DC Motor

A DC motor (Direct Current motor) is the most common type of motor. DC motors normally have 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.



L293D is an H-BRIDGE IC designed for driving low power DC motors and is shown in figure. This IC consists two h-bridges and so it can drive two DC motors. So this IC can be used to drive robot's motors from the signals of microcontroller.



This IC has ability to change the direction of rotation of DC motor. This is achieved by controlling the voltage levels at INPUT1 and INPUT2.

Enable Pin	Input Pin 1	Input Pin 2	Motor Direction
High	Low	High	Turn Right
High	High	Low	Turn Left
High	Low	Low	Stop
High	High	High	Stop

So as shown in above figure, for clockwise rotation 2A should be high and 1A should be low. Similarly for anti clockwise 1A should be high and 2A should be low.

L293D motor driver IC is used for controlling the direction of the motor.

PWM wave generated on the Arduino UNO is used to provide a variable voltage to the motor through L293D. In Arduino, analogWrite function is used to generate PWM wave.

Program for DC motor direction control

```
const int in_1 = 8 ;
```



```

const int in_2 = 10 ;

const int en=11;

//For providing logic to L293 IC to choose the direction of the DC motor

void setup()

{

pinMode(en,OUTPUT) ;    //we have to set PWM pin as output

pinMode(in_1,OUTPUT) ;  //Logic pins are also set as output

pinMode(in_2,OUTPUT) ;

}

void loop()

{

analogWrite(en,255);  //enable pin high

//For Clock wise motion , in_1 = High , in_2 = Low

digitalWrite(in_1,HIGH) ;

digitalWrite(in_2,LOW) ;

//Clockwise for 3 secs

delay(3000) ;

//For brake

digitalWrite(in_1,HIGH) ;

digitalWrite(in_2,HIGH) ;

delay(1000) ;

//For Anti Clock-wise motion - IN_1 = LOW , IN_2 = HIGH

digitalWrite(in_1,LOW) ;

```

```
digitalWrite(in_2,HIGH) ;

delay(3000) ;

//For brake

digitalWrite(in_1,HIGH) ;

digitalWrite(in_2,HIGH) ;

delay(1000) ;

}
```

Program 2 (speed control)

This program will rotate the motor repeatedly at full speed and $\frac{1}{4}$ (25%) speed. Speed is controlled by controlling the power of the pulse at the Enable pin of L293D using PWM. Give a count of 255 for full speed, 192 for 75% speed, 128 for 50% speed and 64 for 25% speed.

```
#define EN 11 // PWM pin 11

#define IN1 10 // DC motor control pin

#define IN2 8 // DC motor control pin

void setup()

{

  pinMode(EN, OUTPUT);

  pinMode(IN1, OUTPUT);

  pinMode(IN2, OUTPUT);

}

void loop()

{

  analogWrite(EN, 255); //full speed

  digitalWrite(IN1, HIGH); //forward
```

```

digitalWrite(IN2, LOW);

delay(5000);

digitalWrite(IN1, HIGH);

digitalWrite(IN2, HIGH); //Brake

delay(500);

analogWrite(EN,64); //1/4 speed

digitalWrite(IN1, HIGH);

digitalWrite(in2, LOW); // forward

delay(5000);

digitalWrite(IN1, HIGH);

digitalWrite(IN2, HIGH); //Brake

delay(500);

}

```

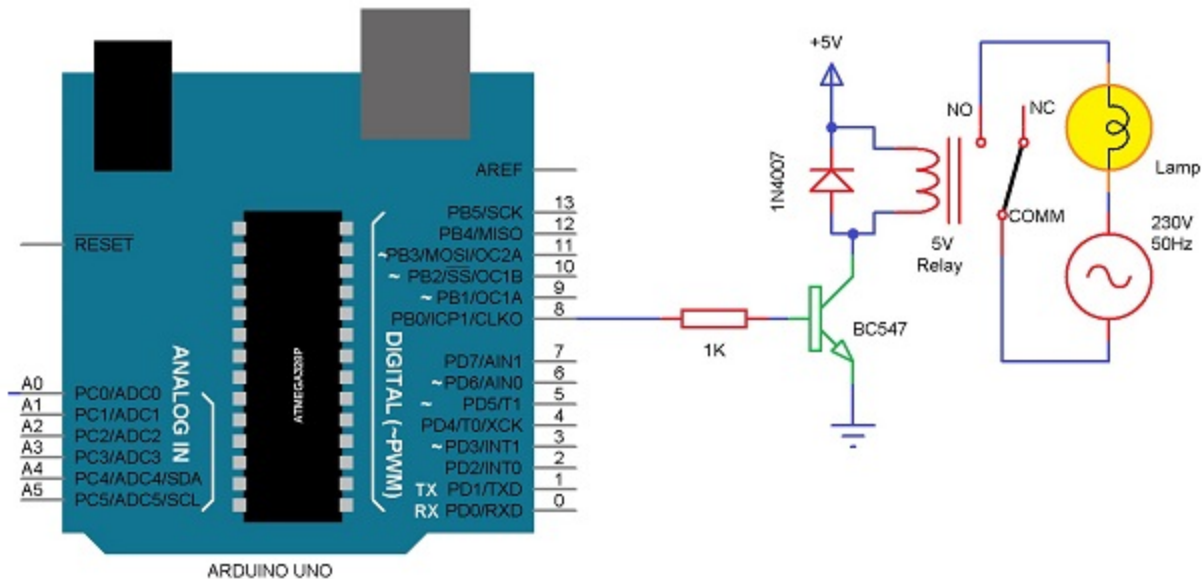
4. Relay

RELAY:

- Relay is an electrically operated switch.
- Relay is used to isolate electrical load.
- Two configurations 1. NO(normally open) 2.NC(normally close),
- Relay has a coil which is energized by 12v.
- when coil energized switching action takes place

BC547:

- BC547 is a general purpose NPN bipolar junction transistor .
- This transistor acts as a driver to pull the relay ON.
- Base is connected with a microcontroller in series with a resistor.
- Emitter to ground.
- Collector to relay.



Program to switch ON and OFF a device connected to Arduino through relay

```
int relaypin =8;/*assigning Arduino pins for the relay signal*/
void setup() {

pinMode(relaypin,OUTPUT);/* assigning the relay pin as an output of Arduino*/

digitalWrite(relaypin,LOW);/* giving the relay pin state of LOW initially */

}

void loop() {

digitalWrite(relaypin,HIGH);/* assigning the relay pin if state HIGH to turn the device on */

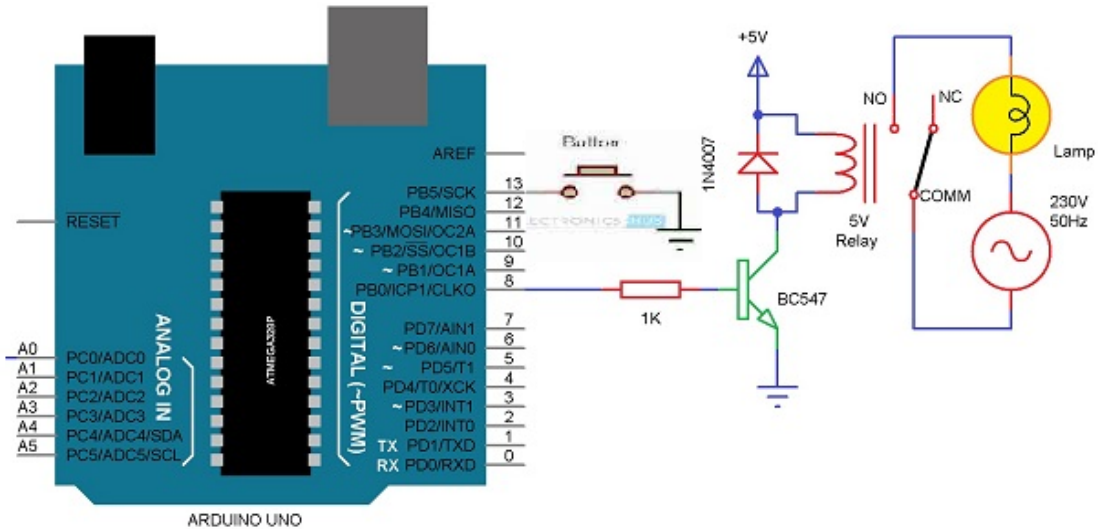
delay(2000);/*time for which the device will remain in on state*/

digitalWrite(relaypin,LOW);/* assigning the relay pin the LOW state to turn off the device*/

delay(2000);/*time for which the device will remain in off state*/

}
```

Program to control relay through a Push button



```
const int BUTTON_PIN = 13; // Arduino pin connected to button's pin
const int RELAY_PIN = 8; // Arduino pin connected to relay's pin
```

```
void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP); // set arduino pin to input pull-up mode
  pinMode(RELAY_PIN, OUTPUT);        // set arduino pin to output mode
}

void loop() {
  if (digitalRead(BUTTON_PIN) == LOW) {
    digitalWrite(RELAY_PIN, HIGH); // turn on
  }
  else
  {
    digitalWrite(RELAY_PIN, LOW); // turn off
  }
}
```

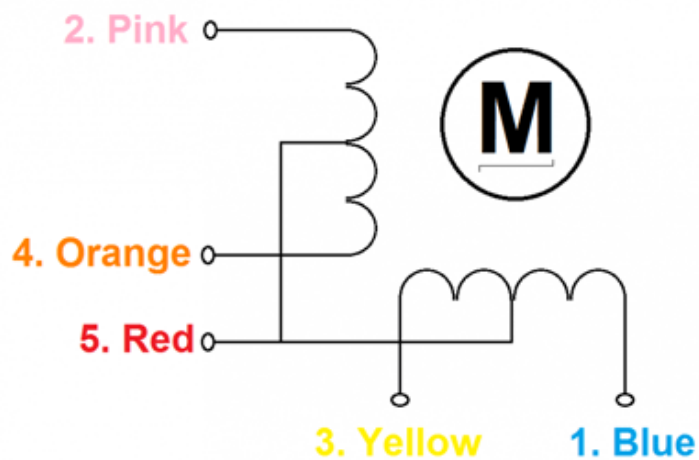
5. Stepper Motor

A Stepper Motor or a step motor is a brushless, synchronous motor, which divides a full rotation into a number of steps. Unlike a brushless DC motor, which rotates continuously when a fixed DC voltage is applied to it, a step motor rotates in discrete step angles.

The Stepper Motors therefore are manufactured with steps per revolution of 12, 24, 72, 144, 180, and 200, resulting in stepping angles of 30, 15, 5, 2.5, 2, and 1.8 degrees per step. The stepper motor can be controlled with or without feedback.

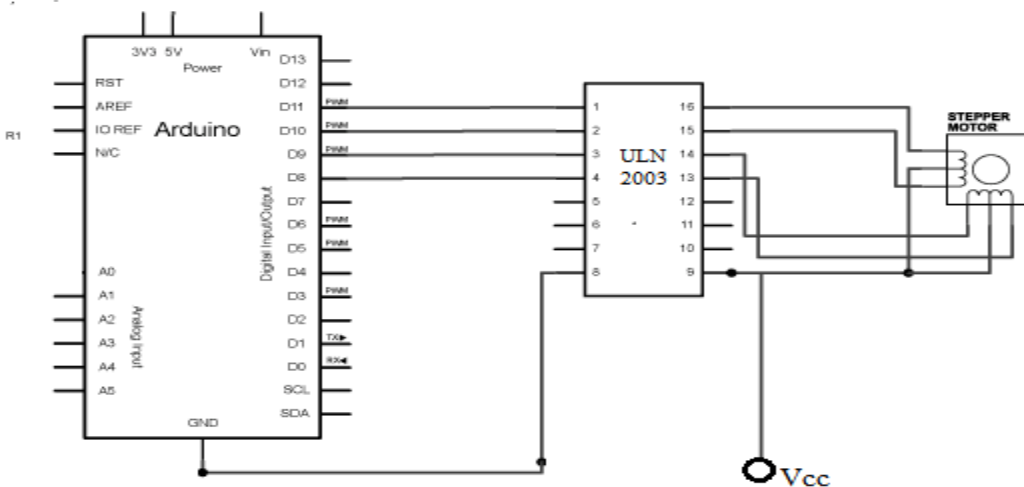
The motor has 5-lead coil arrangement. There are four coils which have to be energized in a particular sequence. The Red wires will be supplied with +5V and the remaining four wires will be pulled to ground for triggering the respective coil. We use a microcontroller

like Arduino to energize these coils in a particular sequence and make the motor perform the required number of steps.

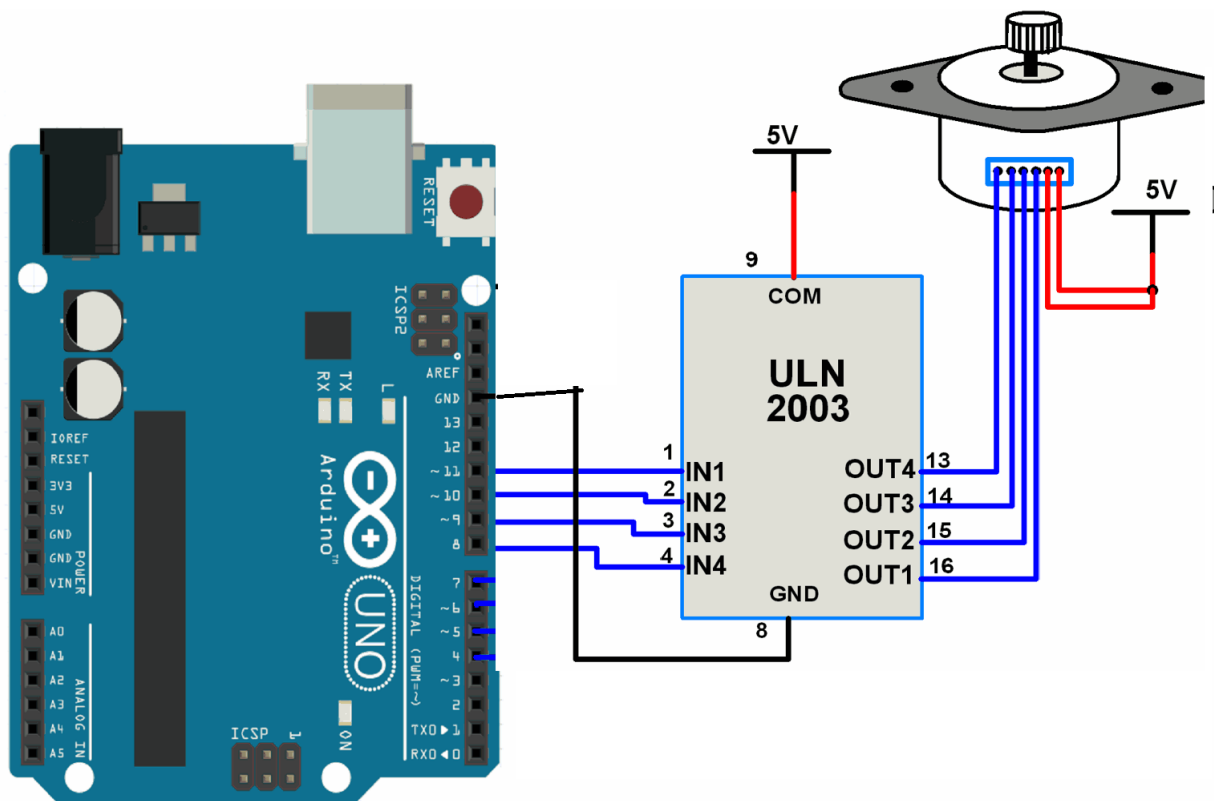


A regular DC motor spins in only direction whereas a Stepper motor can spin in precise increments.

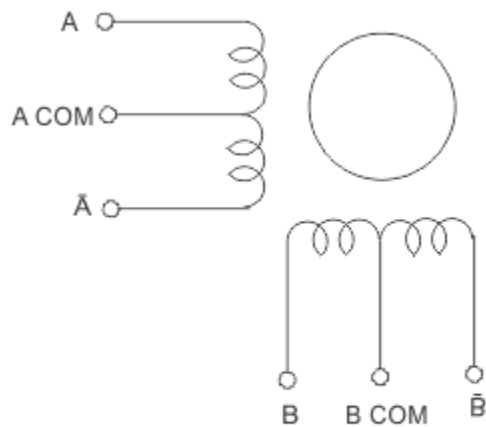
Stepper motors can turn an exact amount of degrees (or steps) as desired. This gives you total control over the motor, allowing you to move it to an exact location and hold that position. It does so by powering the coils inside the motor for very short periods of time. The disadvantage is that you have to power the motor all the time to keep it in the position that you desire.



OR



Pattern for clockwise rotation



Stepper Motor

When the sequence is followed from step 1 to 4, we get a clock wise rotation and when it is followed from step 4 to 1, we get a counter clockwise rotation.

Step No	A	Abar	B	Bbar
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Program for Clockwise

void setup()

```
{
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
}
```

void loop()

```
{
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  digitalWrite(10, HIGH);
  digitalWrite(11, LOW); //start the pattern with 0110
  delay(100);
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
  digitalWrite(10, LOW);
  digitalWrite(11, LOW); // next pattern 0011
  delay(100);
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
```

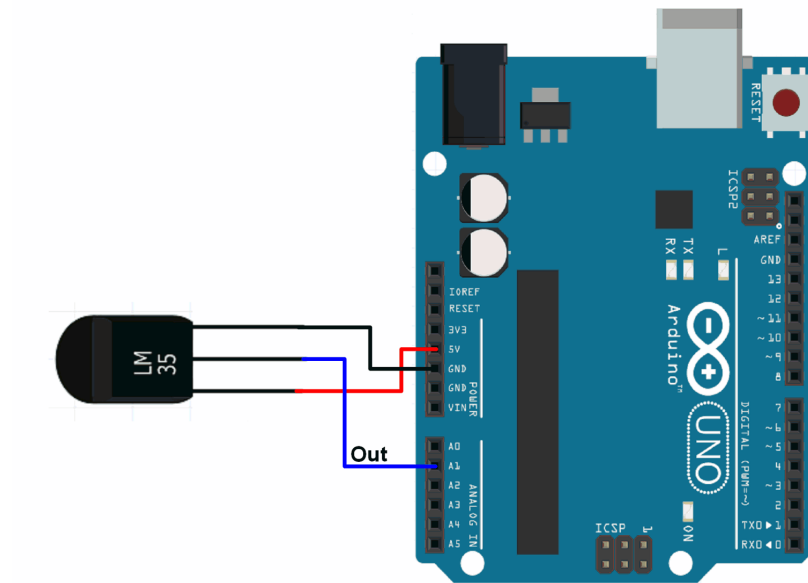
```
digitalWrite(10, LOW);  
digitalWrite(11, HIGH); // next pattern 1001  
delay(100);  
digitalWrite(8, LOW);  
digitalWrite(9, LOW);  
digitalWrite(10, HIGH);  
digitalWrite(11, HIGH); // next pattern 1100  
delay(100);  
}
```

4.5 Sensor Interfacing with Arduino: LM35 Temperature sensor and Ultrasonic sensor (connection diagram, working & programming.)

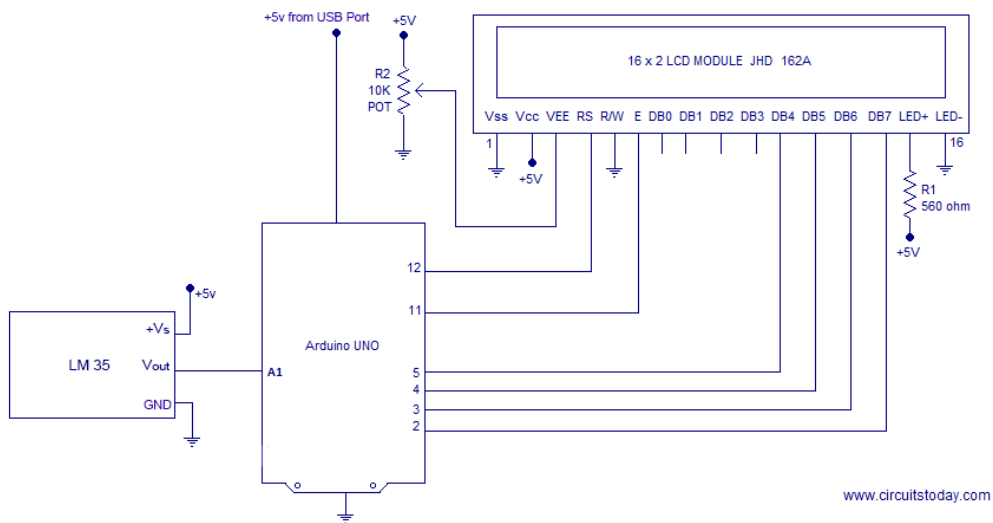
1. LM35

It is a 3-terminal device that provides analog voltage proportional to the temperature. Higher the temperature, higher is the output voltage. The output analog voltage can be converted to digital form using ADC so that a microcontroller can process it.





LM35 and Arduino - Temperature Display on 16x2 LCD Module



```

const int lm35Vout=0;
#include LiquidCrystal.h
LiquidCrystal lcd(12,11,10,5,4,3,2);
const byte degreeSymbol="B11011111";
void setup()
{
  lcd.begin(16,2);
}
void loop()
{

```

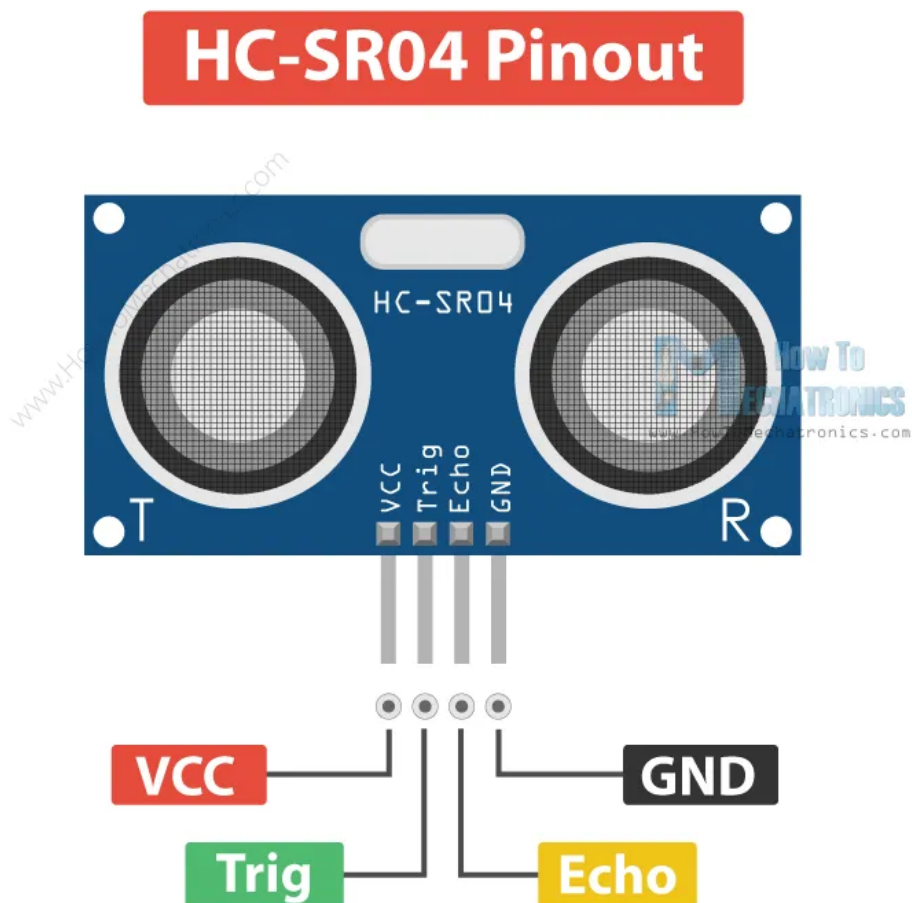
```

long degree;
int value;
long degree;
value=analogRead(lm35Vout);
degree=(long)value*500/1024;
lcd.setCursor(0,0);
lcd.print(degree,DEC);
lcd.print(degreeSymbol);
lcd.print("C");
delay(1000);
}

```

2. Ultrasonic sensor

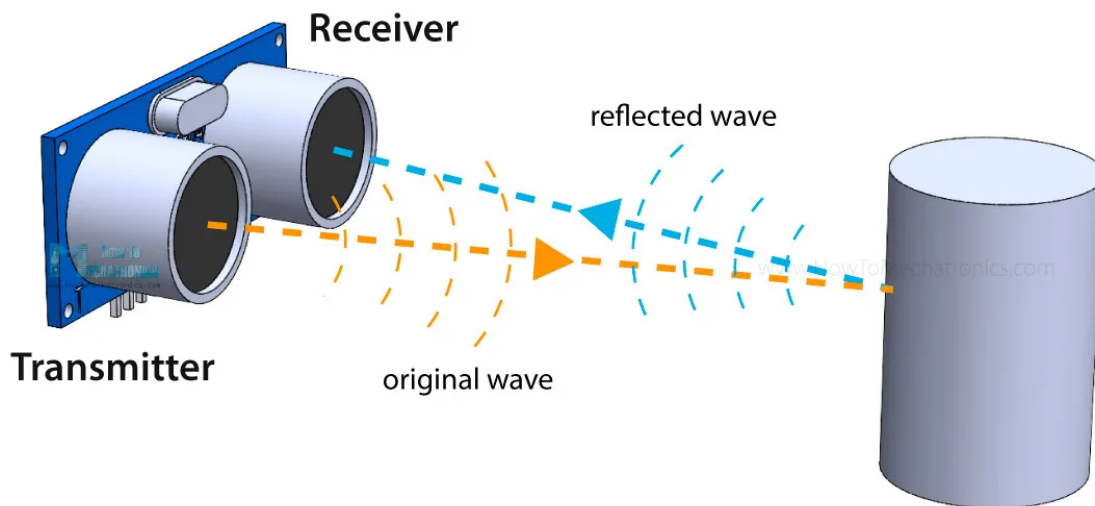
HC-SR04 Ultrasonic Sensor Pinout



The sensor has 4 pins. VCC and GND go to 5V and GND pins on the Arduino, and the Trig and Echo go to any digital Arduino pin. Using the Trig pin we send the ultrasound wave from the transmitter, and with the Echo pin we listen for the reflected signal.

How the HC-SR04 Ultrasonic Distance Sensor Works?

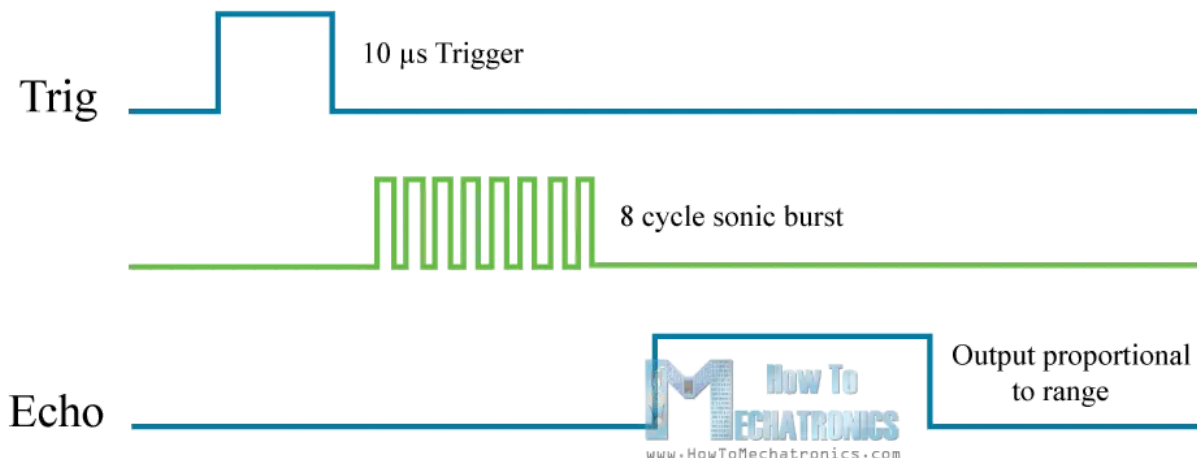
It emits an ultrasound at 40 000 Hz which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.



How Ultrasonic Sensor Working Principle - Explained

In order to generate the ultrasound we need to set the Trig pin on a High State for 10 μ s. That will send out an 8 cycle ultrasonic burst which will travel at the speed of sound. The Echo pins goes high right away after that 8 cycle ultrasonic burst is sent, and it starts listening or waiting for that wave to be reflected from an object.

If there is no object or reflected pulse, the Echo pin will time-out after 38ms and get back to low state.

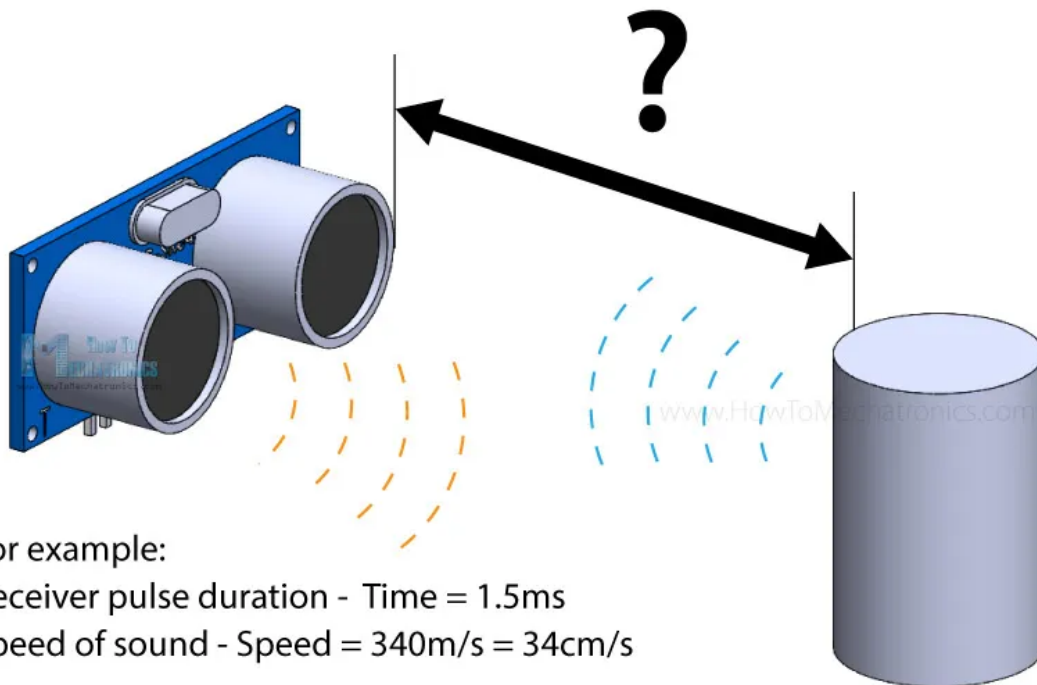


If we receive a reflected pulse, the Echo pin will go down sooner than those 38ms. According to the amount of time the Echo pin was HIGH, we can determine the distance the sound wave traveled, thus the distance from the sensor to the object.

For that purpose we are using the following basic formula for calculating distance:

$$\text{Distance} = \text{Speed} \times \text{Time}$$

We actually know both the speed and the time values. The time is the amount of time the Echo pin was HIGH, and the speed is the speed of sound which is 340m/s. There's one additional step we need to do, and that's divide the end result by 2. and that's because we are measuring the duration the sound wave needs to travel to the object and bounce back.



For example:

Receiver pulse duration - Time = 1.5ms

Speed of sound - Speed = 340m/s = 34cm/s

Distance = (Speed x Time) / 2

Distance = (34cm/ms x 1.5ms) / 2 = 25.5cm

Let's say the Echo pin was HIGH for 2ms. If we want to get the distance result in cm, we can convert the speed of sound value from 340m/s to 34cm/ms.

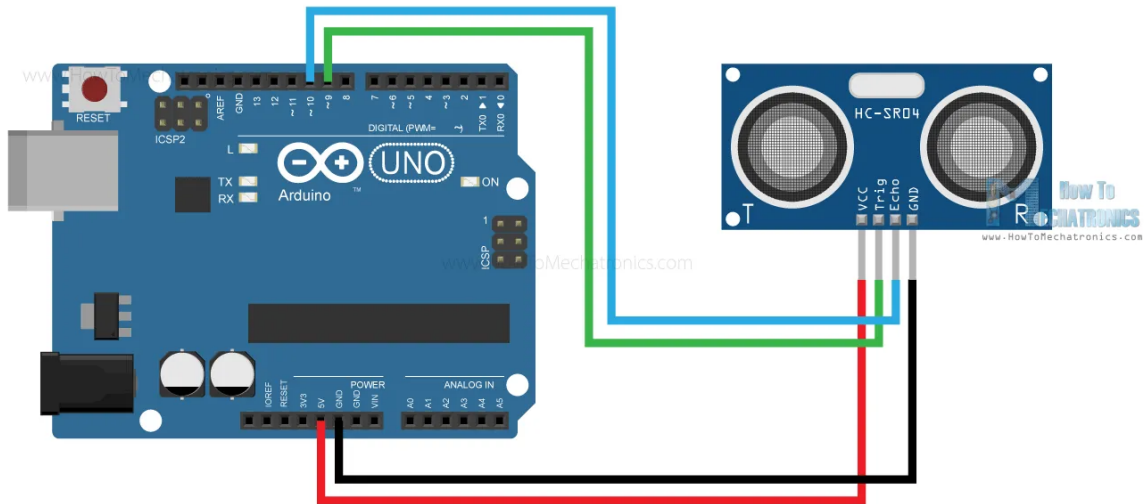
Distance = (Speed x Time) / 2 = (34cm/ms x 1.5ms) / 2 = 25.5cm.

So, if the Echo pin was HIGH for 2ms (which we measure using the pulseIn() function), the distance from the sensor to the object is 34cm.

How to Connect HC-SR04 Ultrasonic Sensor to Arduino

HC-SR04 Ultrasonic Sensor Arduino Connection - Wiring

HC-SR04 Ultrasonic Sensor and Arduino Wiring



The Ground and the VCC pins of the module needs to be connected to the Ground and the 5 volts pins on the Arduino Board respectively and the trig and echo pins to any Digital I/O pin on the Arduino Board.

HC-SR04 Ultrasonic Sensor Arduino Code

```
*/  
// defines pins numbers  
const int trigPin = 9;  
const int echoPin = 10;  
// defines variables  
long duration;  
int distance;  
void setup() {  
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output  
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input  
  Serial.begin(9600); // Starts the serial communication  
}  
void loop() {  
  // Clears the trigPin  
  digitalWrite(trigPin, LOW);  
  delayMicroseconds(2);  
  // Sets the trigPin on HIGH state for 10 micro seconds  
  digitalWrite(trigPin, HIGH);  
  delayMicroseconds(10);  
  digitalWrite(trigPin, LOW);
```

```

// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);
// Calculating the distance
distance = duration * 0.034 / 2;
// Prints the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.println(distance);
}

```

Code Explanation

First we have to define the Trig and Echo pins. In this case they are the pins number 9 and 10 on the Arduino Board and they are named trigPin and echoPin. Then we need a Long variable, named “duration” for the travel time that we will get from the sensor and an integer variable for the distance.

```

// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;

```

```

// defines variables
long duration;
int distance;

```

Code language: Arduino (arduino)

In the setup we have to define the trigPin as an output and the echoPin as an Input and also start the serial communication for showing the results on the serial monitor.

```

void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}

```

Code language: Arduino (arduino)

In the loop first we have to make sure that the trigPin is clear so you have to set that pin on a LOW State for just 2 μ s. Now for generating the Ultra sound wave we have to set the trigPin on HIGH State for 10 μ s.

```

// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);

```

```

// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

```

Code language: Arduino (arduino)

Using the pulseIn() function we read the travel time and put that value into the variable “duration”. This function has 2 parameters, the first one is the name of the Echo pin and for the second is the state of the pulse we are reading, either High or Low.

```
// Reads the echoPin, returns the sound wave travel time in microseconds  
duration = pulseIn(echoPin, HIGH);
```

Code language: Arduino (arduino)

In this case, we need this set to it HIGH, as the HC-SR04 sensors sets the Echo pin to High after sending the 8 cycle ultrasonic burst from the transmitter. This actually starts the timing and once we receive the reflected sound wave the Echo pin will go to Low which stops the timing. At the end the function will return the length of the pulse in microseconds.

For getting the distance we will multiply the duration by 0.034 and divide it by 2 as we explained this equation previously.

```
// Calculating the distance  
distance= duration*0.034/2;
```

```
// Prints the distance on the Serial Monitor  
Serial.print("Distance: ");  
Serial.println(distance);  
distance on the Serial Monitor.
```