



# ***Laboratory Manual*** **ON** **Internet Of Things**

**(For 6<sup>th</sup> Semester CSE/IT)**

***Prepared by:***

***Mrs. Naina Patel***

**PTGF (CSE&IT)**

**UCP Engineering School**

**Berhampur**

***Pabitra Kumar Maharana***

**PTGF(CSE/IT)**

**UCP Engineering School**

**Berhampur**

## Understanding basics of Arduino IDE

Arduino programs are written in the Arduino Integrated Development Environment (IDE). Arduino IDE is a special software running on your system that allows you to write sketches (synonym for program in Arduino language) for different Arduino boards. The Arduino programming language is based on a very simple hardware programming language called processing, which is similar to the C language. After the sketch is written in the Arduino IDE, it should be uploaded on the Arduino board for execution.

The first step in programming the Arduino board is downloading and installing the Arduino IDE. The open-source Arduino IDE runs on Windows, Mac OS X, and Linux. Download the Arduino software (depending on your OS) from the official website and follow the instructions to install.

Now let's discuss the basics of Arduino programming.

The structure of Arduino program is pretty simple. Arduino programs have a minimum of 2 blocks,

Preparation & Execution

Each block has a set of statements enclosed in curly braces:

```
void setup()  
{  
statements-1;  
.  
.  
.  
statement-n;  
}  
void loop ( )  
{  
statement-1;  
.  
.  
.  
statement-n;  
}
```

Here, setup ( ) is the preparation block and loop ( ) is an execution block.

The setup function is the first to execute when the program is executed, and this function is called only once. The setup function is used to initialize the pin modes and start serial communication. This function has to be included even if there are no statements to execute.

```
void setup ( )
{
pinMode (pin-number, OUTPUT); // set the 'pin-number' as output
pinMode (pin-number, INPUT); // set the 'pin-number' as output
}
```

After the setup ( ) function is executed, the execution block runs next. The execution block hosts statements like reading inputs, triggering outputs, checking conditions etc..

In the above example loop ( ) function is a part of execution block. As the name suggests, the loop( ) function executes the set of statements (enclosed in curly braces) repeatedly.

```
Void loop ( )
{
digitalWrite (pin-number,HIGH); // turns ON the component connected to 'pin-
number'
delay (1000); // wait for 1 sec
digitalWrite (pin-number, LOW); // turns OFF the component connected to
'pin-number'
delay (1000); //wait for 1sec
}
```

Note: Arduino always measures the time duration in millisecond. Therefore, whenever you mention the delay, keep it in milli seconds.

### **The Arduino language**

The Arduino language is C++.

Most of the time, people will use a small subset of C++, which looks a lot like C. If you are familiar with Java, then you will find C++ easy to work with and to recognize. If you have never programmed before, do not worry, and do not be afraid. In the next few paragraphs, you will learn everything you need to get started.

The most important “high level” characteristic of C++ is that it is object-oriented. In such a language, an object is a construct that combines functional code (the code that does things like calculations and memory operations), with “state” (the results of such calculations, or simply values, stored in variables).

Object orientation made programming much more productive in most types of applications when compared with earlier paradigms because it allowed programmers to use abstractions to create complicated programs.

For example, you could model an Ethernet adaptor as an object that contains attributes (like its IP and MAC addresses) and functionality (like asking a DHCP server for network configuration details). Programming with objects became

the most common paradigm in programming, and most modern languages, like Java, Ruby, and Python, have been influenced heavily by C++.

Much of the sketch code you will be writing and reading will be referencing libraries containing definitions for objects (these definitions are called “classes”). Your original code, to a large extent, will consist of “glue” code and customizations. This way, you can be productive almost right away by learning a small subset of C++.

The code that makes up your sketch must be compiled into the machine code that the microcontroller on the Arduino can understand. This compilation is done by a special program, the compiler. The Arduino IDE ships with an open-source C++, so you don’t have to worry about the details. Imagine: every time you click the “Upload” button, the IDE starts up the compiler, which converts your human-readable code into ones and zeros, and then sends it to the microcontroller via the USB cable.

As every useful programming language, C++ is made up of various keywords and constructs. There are conditionals, functions, operators, variables, constructors, data structures, and many other things.

In this lesson, you will learn about the structure of an Arduino program, functions, and variables.

### The structure of an Arduino sketch

The simplest possible Arduino sketch is this

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

This code contains two functions in it.

The first one is **setup()**. Anything you put in this function will be executed by the Arduino just once when the program starts.

The second one is **loop()**. Once the Arduino finishes with the code in the **setup()** function, it will move into a **loop()**, and it will continue running it in a loop, again and again, until you reset it or cut off the power.

Notice that both **setup()** and **loop()** have open and close parenthesis? Functions can receive parameters, which is a way by which the program can pass data between its different functions. The setup and loop functions don’t have any parameters passed to them. If you add anything within the parenthesis, you will cause the compiler to print out a compilation error and stop the compilation process.

Every single sketch you write will have these two functions in it, even if you don't use them.

In fact, if you remove one of them, the compiler again will produce an error message. These are two of the few expectations of the Arduino language.

These two functions are required, but you can also make your own. Let's look at this next.

### Custom functions

A function is merely a group of instructions with a name. The Arduino IDE expects that the **setup()** and **loop()** functions will be in your sketch, but you can make your own. Group instructions inside functions is a good way of organizing your sketches, especially as they tend to get bigger in size and complexity as you become a more confident programmer.

To create a function, you need a definition and the code that goes inside the curly brackets.

The definition is made up of at least:

- a return type
- a name
- a list of parameters

Here's an example

```
int do_a_calc(int a, int b)
{
  int c = a + b;
  return c;
}
```

The return type here is **int** in the first line. It tells the compiler that when this function finishes its work, it will return an integer value to the caller (the function that called it).

The name (also known as the "identifier") of the function is **do\_a\_calc**. You can name your functions anything you like as long as you don't use a reserved word (that is, a word that the Arduino language already uses), it has no spaces or other special characters like %, \$ and #. You can't use a number as the first character. If in doubt, remember only to use letters, numbers, and the underscore in your function names.

In the first line of the body, we create a new variable, **c**, of type integer (**int**). We add **a** and **b** and then assign the result to **c**.

And finally, in the second line of the body of the function, we return the value stored in **c** to the caller of **do\_a\_calc**.

Let's say that you would like to call **do\_a\_calc** from your setup function. Here's a complete example showing how to do that:

```
void setup()
{
  // put your setup code here, to run once:
  int a = do_a_calc(1,2);
}

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

int do_a_calc(int a, int b)
{
  int c = a + b;
  return c;
}
```

In the **setup()** function, the second line defines a new variable, **a**. In the same line, it calls the function **do\_a\_calc**, and passes integers 1 and 2 to it. The **do\_a\_calc** function calculates the sum of the two numbers and returns the value 3 to the caller, which is the second line of the **setup()** function. Then, the value 3 is stored in variable **a**, and the **setup()** function ends.

There's a couple of things to notice and remember.

### Comments

Any line that starts with **//** or multiple lines that start with **/\*** and finish with **\*/** contain comments.

Comments are ignored by the compiler. They are meant to be read by the programmer.

Comments are used to explain the functionality of code or leave notes to other programmers (or to self).

### Scope

In the **setup()** function, there is a definition of a variable with an identifier **a**. In function **do\_a\_calc**, there is also a definition of a variable with the same identifier (it makes no difference that this definition is in the function definition line).

Having variables with the same name is not a problem as long as they are not in the same scope. A scope is defined by the curly brackets. Any variable between an open and close curly bracket is said to be within that scope. If there is a variable with the same name defined within another scope, then there is no conflict.

Be careful when you choose a name for your variables. Problems with scopes can cause headaches: you may expect that a variable is accessible at a particular part of your sketch, only to realize that it is out of scope.

Also, be careful to use good descriptive names for your variables. If you want to use a variable to hold the number of a pin, call it something like:

```
int digital_pin = 1;
instead of
int p = 1;
```

You will thank yourself later.

## Variables

Programs are useful when they process data. Processing data is what programs do, all the time. Programs will either get some data to process from a user (perhaps via a keypad). From a sensor (like a thermistor that measures temperature), the network (like a remote database), a local file system (like an SD Card), a local memory (like an EEPROM), and so many other places.

Regardless of the place where your program gets its data from, it must store them in memory to work with it. To do this, we use variables. A variable is a programming construct that associates a memory location with a name (an identifier). Instead of using the address of the memory location in our program, we use an easy to remember a name. You have already met a variable. In the earlier section on custom functions, we defined a bunch of variables, **a**, **b** and **c**, that each holds an integer.

Variables can hold different kinds of data other than integers. The Arduino language (which, remember, is C++) has built-in support for a few of them (only the most frequently used and useful are listed here):

boolean	1 byte	Holds only two possible values, <b>true</b> or <b>false</b> , even though it occupies a byte in memory.
char	1 byte	Holds a number from -127 to 127. Because it is marked as a "char," the compiler will try to match it to a character from the <a href="#">ASCII table of characters</a> .
byte	1 byte	Can hold numbers from 0 to 255.
int	2 byte	Can hold numbers from -32768 to 32767.
unsigned int	2 byte	Can hold numbers from 0-65535

word	2 byte	Same as the “unsigned int.” People often use “word” for simplicity and clarity.
long	4 byte	Can hold numbers from -2,147,483,648 to 2,147,483,647.
unsigned long	4 byte	Can hold numbers from 0-4,294,967,295
float	4 byte	Can hold numbers from -3.4028235E38 to 3.4028235E38. Notice that this number contains a decimal point. Only use float if you have no other choice. The ATMEGA CPU does not have the hardware to deal with floats, so the compiler has to add a lot of code to make it possible for your sketch to use them, making your sketch larger and slower.
string - char array	-	A way to store multiple characters as an array of chars. C++ also offers a String object that you can use instead that provides more flexibility when working with strings in exchange for higher memory use.
array	-	A structure that can hold multiple data of the same type.

To create a variable, you need a valid name and a type. Just like with functions, a valid name is one that contains numbers, letters, and an underscore starts with a letter and is not reserved. Here is an example:

```
byte sensor_A_value;
```

This line defines a variable named **sensor\_A\_value**, which will hold a single byte in memory. You can store a value in it like this:

```
sensor_A_value = 196;
```

You can print out this value to the serial monitor like this:

```
Serial.print(sensor_A_value);
```

The serial monitor is a feature of the Arduino IDE that allows you to get a text from the Arduino displayed on your screen. More about this later, here I want to show you how to retrieve the value stored in a variable. Just call its name. Also, remember the earlier discussion about scope: the variable has to be within scope when it is called.

Another beautiful thing about a variable is that you can change the value stored in it. You can take a new reading from the sensor and update the variable like this:

```
sensor_A_value = 201;
```

No problem, the old value is gone, and the new value is stored.

### Constants

If there is a value that will not be changing in your sketch, you can mark it as a constant.



Constants have benefits regarding memory and processing speed, and it is a good habit to use these.

You can declare a constant like this:

```
const int sensor_pin = 1;
```

Here, you define the name of the variable **sensor\_pin**, mark it as constant, and set it to 1. If you try to change the value later, you will get a compiler error message, and your program will not even get uploaded to the Arduino.

### Operators

Operators are special functions that operate one or more pieces of data.

Most people are familiar with the basic arithmetic functions, = (assignment), +, -, \* and /, But there are a lot more.

For example, here are the most commonly used operators:

%	Modulo operator. It returns the remainder of a division.	5%2=1
+=, -=, *=, /=	Compound operator. It performs an operation on the current value of a variable.	<pre>int a = 5; a+= 2;</pre> <p>This will result in a containing 7 (the original 5 plus a 2 from the addition operation).</p>
++, --	Increment and decrement by 1.	<pre>int a = 5; a++;</pre> <p>This will result in a becoming 6.</p>
==, !=, <, >, <=, >=	<p>Comparison operators. Will return a boolean (true or false) depending on the comparison result.</p> <ul style="list-style-type: none"> <li>• == → equality</li> <li>• != → un-equality</li> <li>• &lt; → less than</li> <li>• &gt; → greater than</li> <li>• &lt;= → less or equal than</li> <li>• &gt;= → greater or equal than</li> </ul>	<pre>int a = 5; int b = 6; boolean c = a == b;</pre> <p>This will result in variable c contains a false boolean value.</p>

```
boolean a = true;
boolean b = true;
boolean c = false;
```

```
boolean x = !a; // x → false
boolean y = b && c; // y → false
boolean z = b || c; // z → true
```

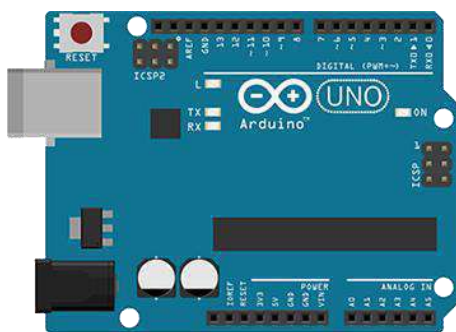
Logical operators. The “!” operator will invert a boolean value.

!, &&, ||  
! → NOT (invert) of a boolean value  
&& → AND of two booleans  
|| → OR of two booleans

## Practical using Arduino-interfacing sensors

### Arduino Programming in C

Arduino is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices.



### Board Components

These are the components that make up an Arduino board and what each of their functionalities are.

- **Reset Button** ~ This will restart any code that is loaded to the Arduino board

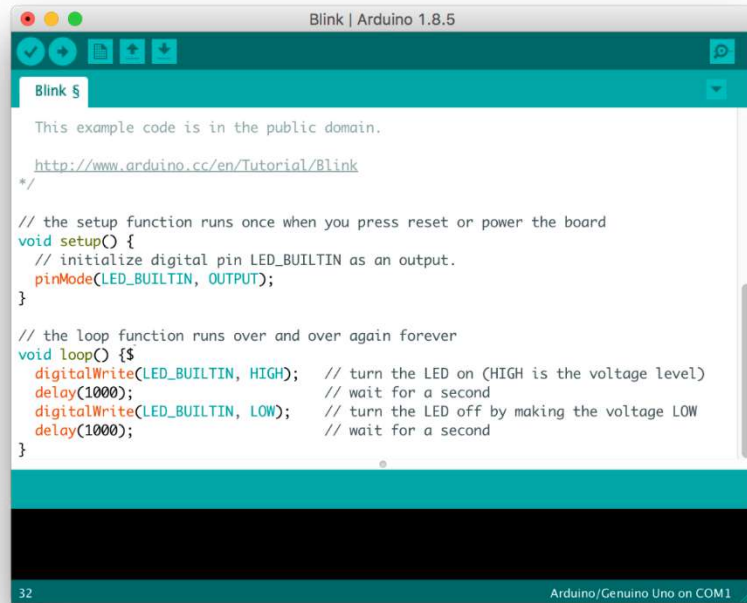
- **AREF** ~ Stands for “Analog Reference” and is used to set an external reference voltage
- **Ground Pin** ~ There are a few ground pins on the Arduino and they all work the same
- **Analog Pins** ~ These pins read the signal from an analog sensor and convert it to digital
- **Digital Input/Output** ~ Pins 0-13 can be used for digital input or output
- **PWM** ~ The pins marked with the (~) symbol can simulate analog output
- **USB Connection** ~ Used for powering up your Arduino and uploading sketches
- **TX/RX** ~ Transmit and receive data indication LEDs
- **ATmega Microcontroller** ~ Popular microcontroller chip, this is where the programs are stored
- **Power LED Indicator** ~ This LED lights up anytime the board is plugged in a power source
- **Voltage Regulator** ~ Controls the amount of voltage going into the Arduino board
- **DC Power Barrel Jack** ~ This is used for powering your Arduino with a power supply
- **3.3V Pin** ~ This pin supplies 3.3 volts of power
- **5V Pin** ~ This pin supplies 5 volts of power

### **Install Arduino Code Editor**

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

### **Download Arduino Code Editor**

- Navigate to [arduino.cc](http://arduino.cc) and download The open-source Arduino Software (IDE) for Windows, MacOSX& Linux.
- Use the Online Code Editor, or scroll down on the arduino download page to download the version for Windows, MacOSX& Linux.

A screenshot of the Arduino IDE interface. The title bar reads "Blink | Arduino 1.8.5". The main text area contains the following code:

```
Blink $  
  
This example code is in the public domain.  
  
http://www.arduino.cc/en/Tutorial/Blink  
*/  
  
// the setup function runs once when you press reset or power the board  
void setup() {  
  // initialize digital pin LED_BUILTIN as an output.  
  pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000); // wait for a second  
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW  
  delay(1000); // wait for a second  
}
```

The status bar at the bottom shows "32" on the left and "Arduino/Genuino Uno on COM1" on the right.

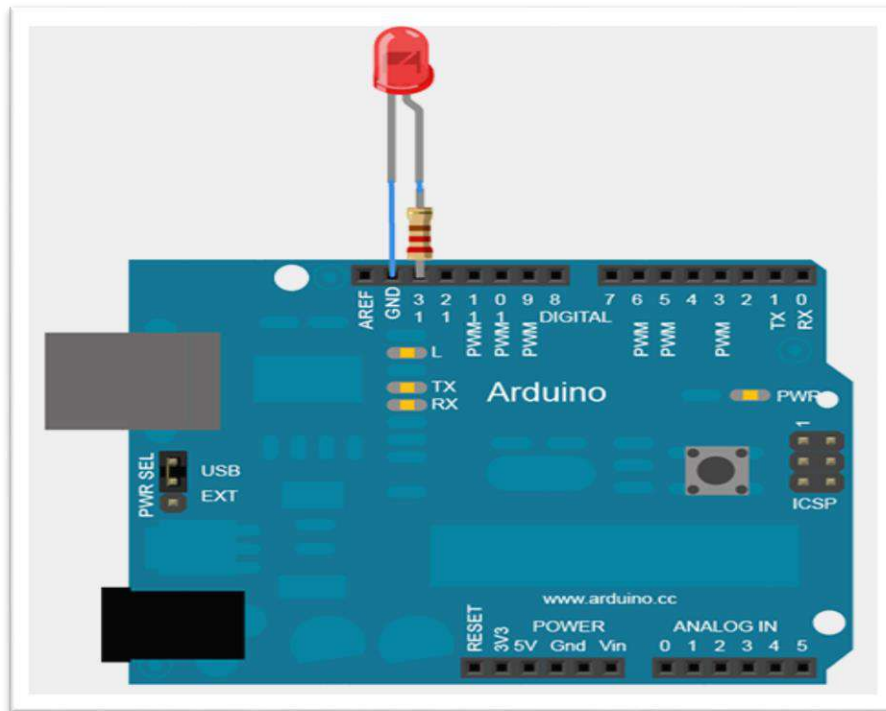
## Blink a LED light

### Example Arduino Schema

A step by step example showing how to blink a LED light with an Arduino, this example provides the board schematics, code and a list of components that are required.

### Components Required

- 1 × Arduino Uno R3
- 1 × LED
- 1 × 330Ω Resistor



### Example Code (Blink a LED light)

```
// The pin the LED is connected to  
int ledPin = 13;
```

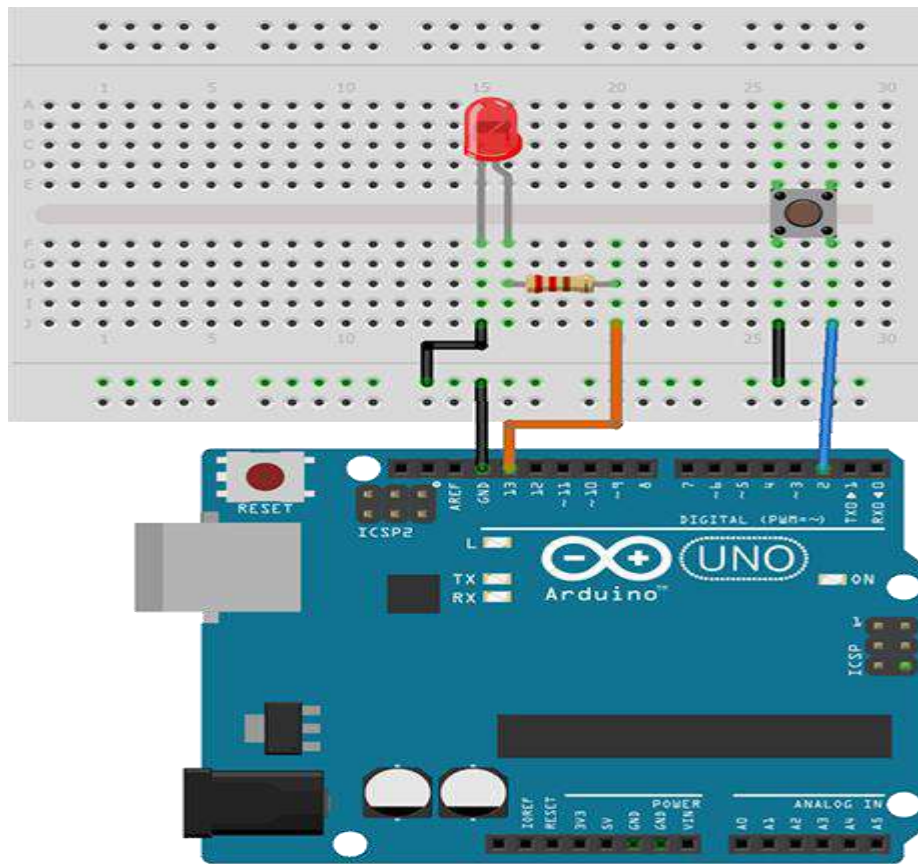
```
// Executes once when the arduino power button is pressed on  
void setup() {  
  pinMode(ledPin, OUTPUT); // Declare the LED as an output  
}
```

```
// This method repeats forever  
// This method makes a led light blink every 1 second  
void loop() {  
  digitalWrite(ledPin, HIGH); // Turn the LED on  
  delay(1000); // Delay 1000 milliseconds  
  digitalWrite(ledPin, LOW); // Turn the led on  
  delay(1000); // Delay 1000 milliseconds  
}
```

### Turn On, LED light with a Push Button

A step by step example showing how to Turn On, LED light with a Push Button on an Arduino, this example provides the board schematics, code and a list of components that are required.

### Example Arduino Schema



### Components Required

- 1 x Breadboard
- 1 x Arduino Uno R3
- 1 x LED
- 1 x Push Button
- 1 x 330Ω Resistor
- 5 x Jumper Cables

### Example Code (Turn On, LED light with a Push Button)

```
int ledPin = 13; // The pin the LED is connected to
int inputPin = 2; // The input pin (for a push button)
int inputStatus = 0; // Variable for reading the pin status

// Executes once when the arduino power button is pressed on
void setup() {
  pinMode(ledPin, OUTPUT); // Declare the LED as an output
  pinMode(inputPin, INPUT); // Declare pushbutton as input
}

// This method repeats forever
// This method makes a led light when button is pushed
void loop() {
```

```

// Reads incoming input value from our inputPin
inputStatus = digitalRead(inputPin);

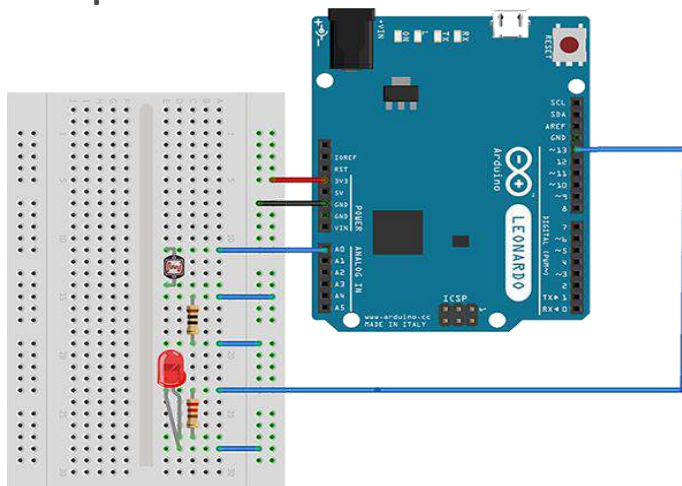
// Check if the input is HIGH (pushbutton released)
if (inputStatus == HIGH) {
  digitalWrite(ledPin, LOW); // Turn the LED off
} else {
  digitalWrite(ledPin, HIGH); // Turn the LED on
}
}

```

### Using a LRD Light Sensor

A step by step example showing how to Turn On, LED light using a LDR Light Sensor on an Arduino, this example provides the board schematics, code and a list of components that are required.

### Example Arduino Schema



### Components Required

- 1 x Breadboard
- 1 x Arduino Uno R3
- 1 x LED
- 1 x LRD Light Sensor
- 1 x 330Ω Resistor
- 1 x 100k Ohm Resistor
- 7 x Jumper Cables

### Example Code (Using a LRD Light Sensor, to turn on a LED light)

```

constintledPin = 13; // The pin the LED is connected to
constintldrPin = A0; // The pin the LDR sensor is connected to
intldrStatus = 0; // Variable for reading the pin status

```

```
// Executes once when the arduino power button is pressed on
```

```
voidsetup()
```

```
{
```

```
pinMode(ledPin, OUTPUT); // Declare the LED as an output
```

```
pinMode(ldrPin, INPUT); // Declare LDR Sensor as an input
```

```
}
```

```
// This method repeats forever
```

```
voidloop()
```

```
{
```

```
// Read incoming data from LDR sensor pin
```

```
ldrStatus = analogRead(ldrPin);
```

```
if (ldrStatus <= 400) {
```

```
digitalWrite(ledPin, HIGH); //It's dark, Turn on LED
```

```
  } else {
```

```
digitalWrite(ledPin, LOW); // It's bright, Turn off LED
```

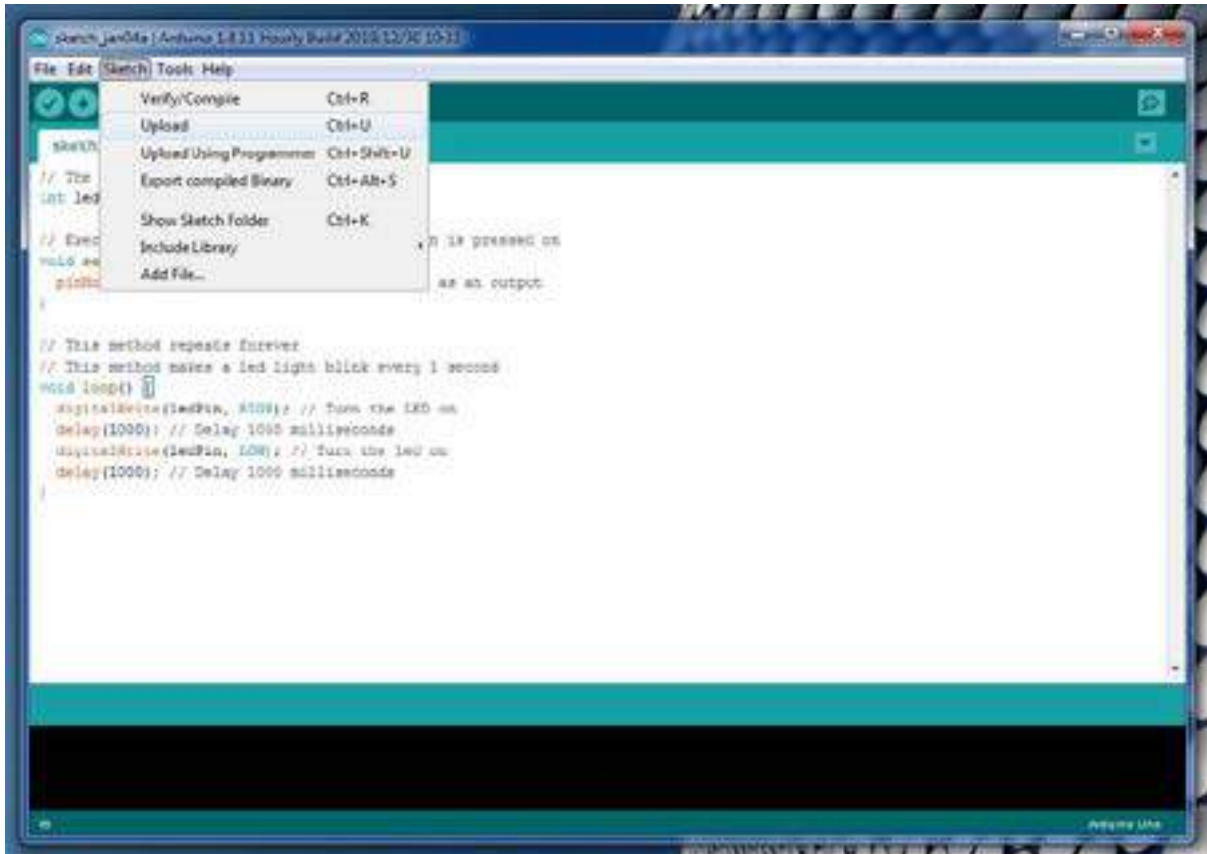
```
  }
```

```
}
```

### Run Project with Arduino IDE

- To run the code, on the menu bar, choose **Sketch > Upload**. The code will be uploaded to the ATmega Microcontroller
- Press the reset button to restart any code that is loaded to the Arduino board





## LDR – Automatic Night Lamp

### 1. Introduction:

A LDR (Light Dependent Resistor) or a photo resistor is a photo conductive sensor. It is a variable resistor and changes its resistance in a proportion to the light exposed to it. Its resistance decreases with the intensity of light.

It senses the light intensity from surroundings and finds whether it is day or night then it automatically turns ON when the surrounding is dark and it turns OFF when it receives light from surroundings.

### 2. Required Hardware

Following Hardware will be required to perform this LDR circuit.

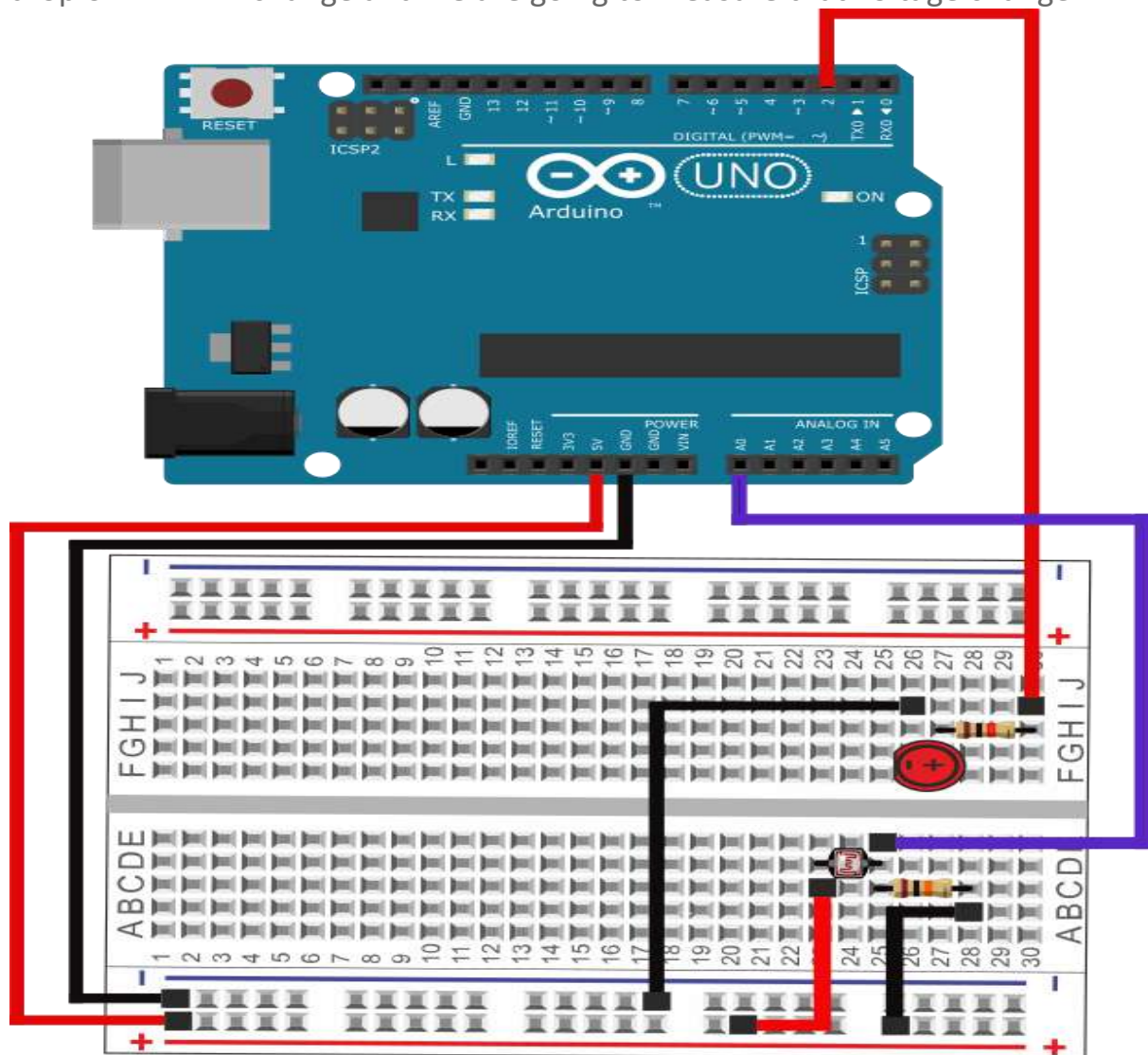
S.No.	Item	Quantity
1	<a href="#">Arduino UNO</a>	1
2	<a href="#">Breadboard</a>	1
3	<a href="#">LDR</a>	1
4	<a href="#">LED</a>	1

5	<a href="#">Resistor 1k</a>	1
6	<a href="#">Resistor 10k</a>	1
7	<a href="#">Male to Male Jumper</a>	7

### 3. Building Circuit

Make following circuit with the help of above mentioned components. Some key points to understand about the circuit-

LDR is connected to a 10 Resistance in series. +5 Voltage is applied to this arrangement. As the light intensity changes LDR value changes thus the voltage drop on LDR will change and we are going to measure that voltage change.



### 4. Programming:

Once we are done with circuit part, here is our programme to this circuit.

/Robo India Tutorial on Night Lamp using LDR

```
//https://www.roboindia.com/tutorials
```

```
const int LED=2;      // LED connect to Digital Pin  
const int LDRSensor= A0; //Sensor pin connects to analog pin A0
```

```
int state;           //declaring variable to store the reading  
int threshold=600;   //threshold voltage declared
```

```
void setup()  
{  
  pinMode (LED, OUTPUT);  
  Serial.begin(9600);  
}
```

```
void loop()  
{  
  state= analogRead(LDRSensor); //sensor reading value stored in state variable  
  if (state < threshold)  
  {  
    digitalWrite(LED, HIGH); //if the light is below the threshold value, the LED  
will turns on  
    Serial.println(state);  
    delay(2000);  
  }  
  else  
  {  
    digitalWrite(LED, LOW); //otherwise, the LED is off  
    Serial.println(state);  
    delay(1000);  
  }  
}
```

```
/Robo India Tutorial on Night Lamp using LDR
```

```
//https://www.roboindia.com/tutorials
```

```
const int LED=2;      // LED connect to Digital Pin  
const int LDRSensor= A0; //Sensor pin connects to analog pin A0
```

```
int state;           //declaring variable to store the reading  
int threshold=600;   //threshold voltage declared
```

```

void setup()
{
  pinMode (LED, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  state= analogRead(LDRSensor); //sensor reading value stored in state variable
  if (state < threshold)
  {
    digitalWrite(LED, HIGH); //if the light is below the threshold value, the LED
will turns on
    Serial.println(state);
    delay(2000);
  }
  else
  {
    digitalWrite(LED, LOW); //otherwise, the LED is off
    Serial.println(state);
    delay(1000);
  }
}

```

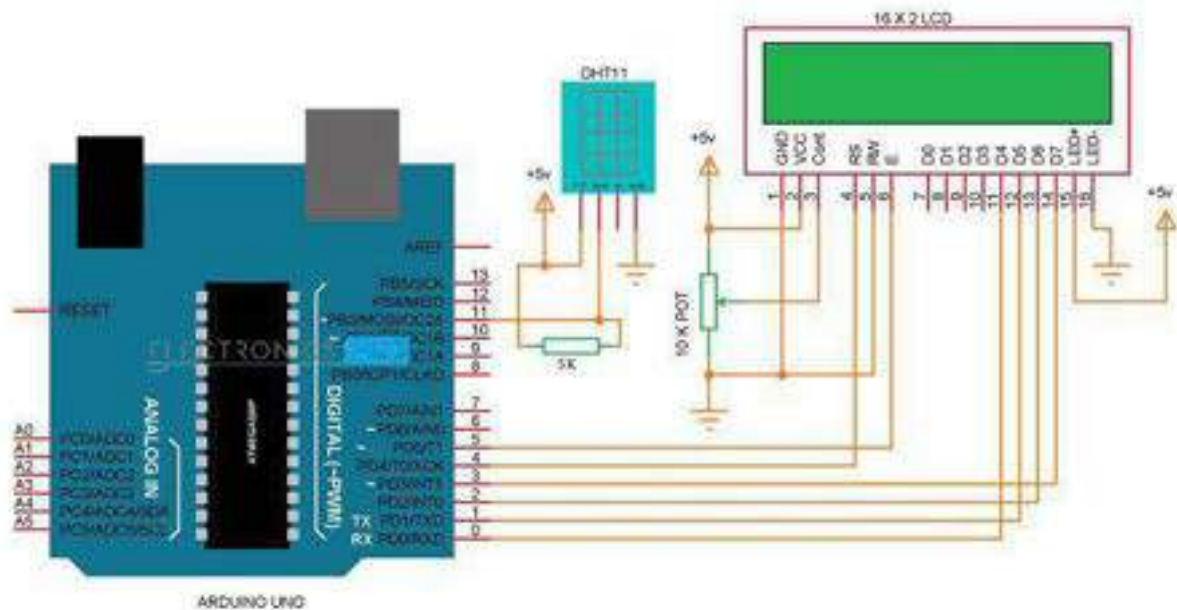
Control the light exposing on LDR by covering it. If it gets dark, the LED connected will get turn ON automatically.

### DHT11 Humidity Sensor on Arduino

DHT11 is a Humidity and Temperature Sensor, which generates calibrated digital output. DHT11 can be interface with any microcontroller like Arduino, Raspberry Pi, etc. and get instantaneous results. DHT11 is a low cost humidity and temperature sensor which provides high reliability and long term stability. In this project, we will build a small circuit to interface Arduino with DHT11 Temperature and Humidity Sensor. One of the main applications of connecting DTH11 sensor with Arduino is weather monitoring.

#### Circuit Diagram

The following circuit diagram shows all the necessary connections required to implement this project.



### Components Required

- Arduino UNO
- DHT11 Temperature and Humidity Sensor
- Breadboard (or perfboard)
- Power supply
- 16 x 2 LCD Display
- 10K Ohm Potentiometer
- 5K Ohm Resistor (1/4 W)
- Connecting wires

### Circuit Description

We will see the circuit design of DHT11 interfacing with Arduino. The DHT11 Humidity and Temperature sensor comes in two variants: just the sensor or a module.

The main difference is that the module consists of the pull – up resistor and may also include a power on LED. We have used a module in this project and if you wish to use the sensor itself, you need to connect a 5K  $\Omega$  pull – up resistor additionally.

Coming to the design, the data pin of the DHT11 Sensor is connected to the Pin 11 of Arduino. A 16 x 2 LCD display is used to display the results. The control pins of LCD i.e. RS and E (Pins 4 and 6 on LCD) are connected to pins 4 and 5 of Arduino. The data pins of LCD i.e. D4 to D7 (pins 11 to 14 on LCD) are connected to pins 0 to 3 on LCD.

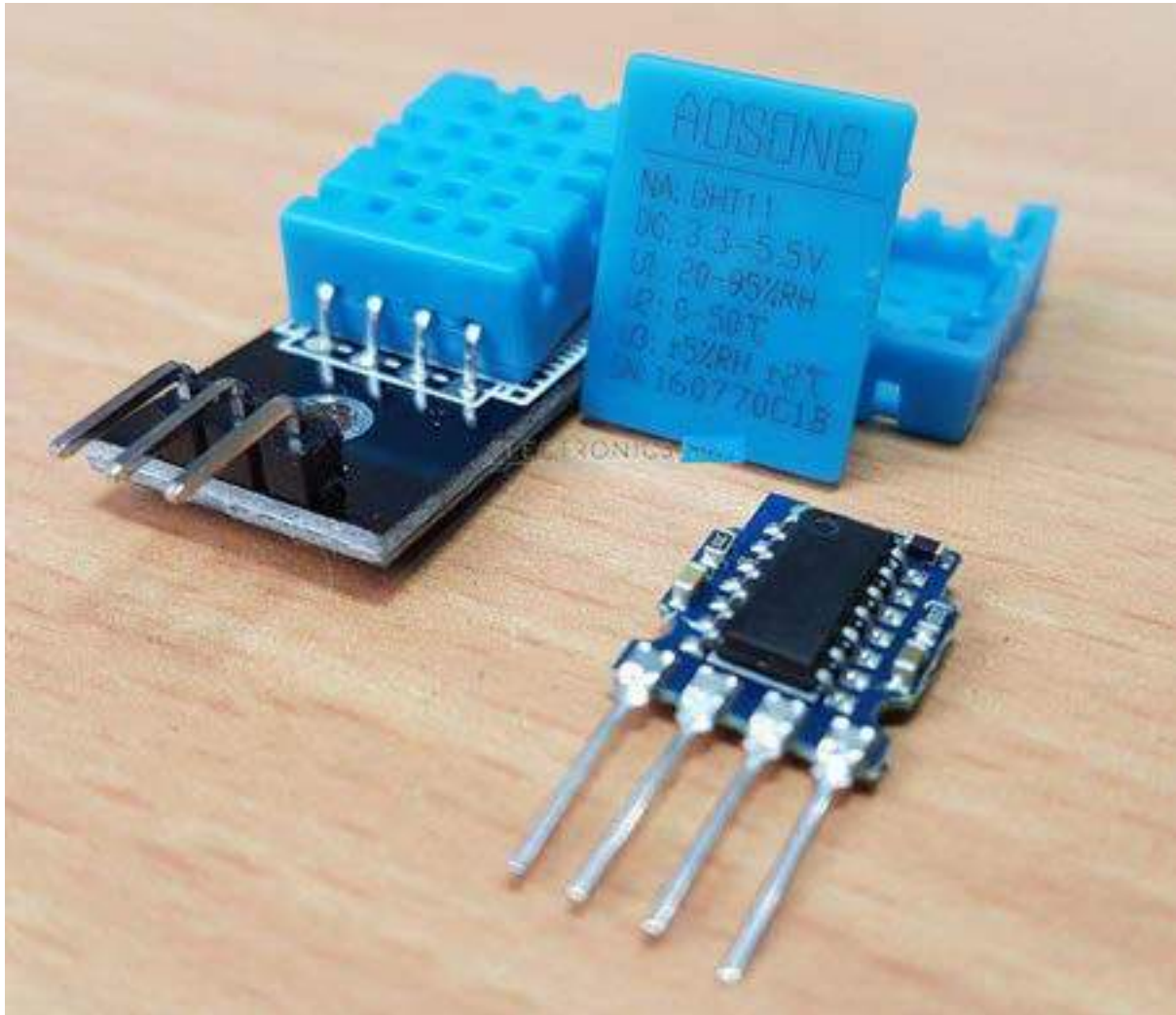
**NOTE:** For ease of connection, we have connected the DHT11 Sensor Module at the ICSP pins of the Arduino as it provides adjacent VCC, DATA and GND

pins. This type of connection is not necessary and you can connect the data pin of sensor to normal Digital I/O pins.

#### Component Description

##### *DHT11 Temperature and Humidity Sensor*

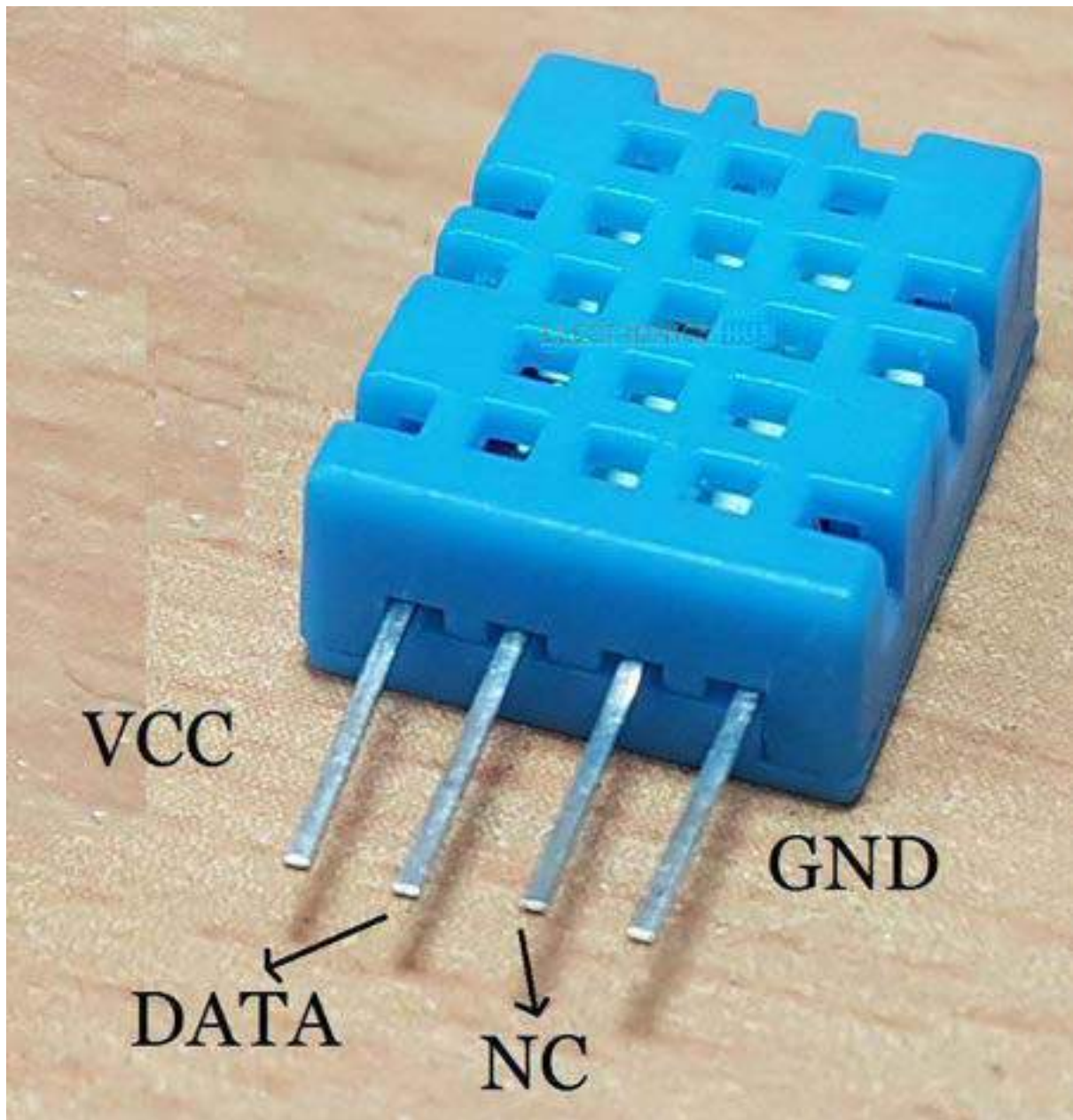
DHT11 is a part of DHTXX series of Humidity sensors. The other sensor in this series is DHT22. Both these sensors are Relative Humidity (RH) Sensor. As a result, they will measure both the humidity and temperature. Although DHT11 Humidity Sensors are cheap and slow, they are very popular among hobbyists and beginners.



The DHT11 Humidity and Temperature Sensor consists of 3 main components. A resistive type humidity sensor, an NTC (negative temperature coefficient) thermistor (to measure the temperature) and an 8-bit microcontroller, which converts the analog signals from both the sensors and sends out single digital signal.

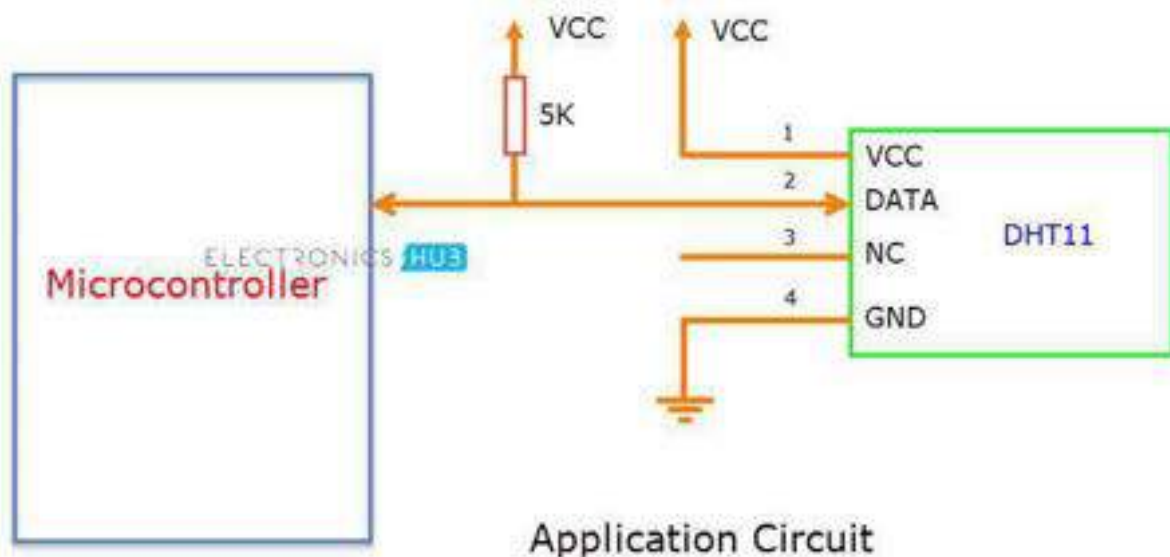
This digital signal can be read by any microcontroller or microprocessor for further analysis.





DHT11 Humidity Sensor consists of 4 pins: VCC, Data Out, Not Connected (NC) and GND. The range of voltage for VCC pin is 3.5V to 5.5V. A 5V supply would do fine. The data from the Data Out pin is a serial digital data.

The following image shows a typical application circuit for DHT11 Humidity and Temperature Sensor. DHT11 Sensor can measure a humidity value in the range of 20 – 90% of Relative Humidity (RH) and a temperature in the range of 0 – 50°C. The sampling period of the sensor is 1 second i.e.



All the DHT11 Sensors are accurately calibrated in the laboratory and the results are stored in the memory. A single wire communication can be established between any microcontroller like Arduino and the DHT11 Sensor. Also, the length of the cable can be as long as 20 meters. The data from the sensor consists of integral and decimal parts for both Relative Humidity (RH) and temperature.

The data from the DHT11 sensor consists of 40 – bits and the format is as follows:

8 – Bit data for integral RH value, 8 – Bit data for decimal RH value, 8 – Bit data for integral Temperature value, 8 – Bit data for integral Temperature value and 8 – Bit data for checksum.

#### Example

Consider the data received from the DHT11 Sensor is  
00100101 00000000 00011001 00000000 00111110.

This data can be separated based on the above mentioned structure as follows

<b>00100101</b>	<b>00000000</b>	<b>00011001</b>	<b>00000000</b>	<b>00111110</b>
High Humidity	Low Humidity	High Temperature	Low Temperature	Checksum (Parity)

In order to check whether the received data is correct or not, we need to perform a small calculation. Add all the integral and decimals values of RH and Temperature and check whether the sum is equal to the checksum value i.e. the last 8 – bit data.

$$00100101 + 00000000 + 00011001 + 00000000 = 00111110$$

This value is same as checksum and hence the received data is valid. Now to get the RH and Temperature values, just convert the binary data to decimal data.

$$\text{RH} = \text{Decimal of } 00100101 = 37\%$$



**Temperature = Decimal of 00011001 = 25°C**

Working of the Project

A simple project is built using Arduino UNO and DHT11 Humidity and Temperature Sensor, where the Humidity and Temperature of the surroundings are displayed on an LCD display.

After making the connections, we need not do anything as the program will take care of everything. Although there is a special library for the DHT11 module called "DHT", we didn't use it. If you want to use this library, you need to download this library separately and add it to the existing libraries of Arduino.

The program written is based on the data timing diagrams provided in the datasheet. The program will make the Arduino to automatically read the data from the sensor and display it as Humidity and Temperature on the LCD Display.

CODE

```
#include
LiquidCrystal lcd(4, 5, 0, 1, 2, 3);
byte degree_symbol[8] =
{
0b00111,
0b00101,
0b00111,
0b00000,
0b00000,
0b00000,
0b00000,
0b00000
};
int gate=11;
volatile unsigned long duration=0;
unsigned char i[5];
unsigned int j[40];
unsigned char value=0;
unsigned answer=0;
int z=0;
int b=1;
void setup()
{
lcd.begin(16, 2);
lcd.print("Temp = ");
```

```

lcd.setCursor(0,1);
lcd.print("Humidity = ");
lcd.createChar(1, degree_symbol);
lcd.setCursor(9,0);
lcd.write(1);
lcd.print("C");
lcd.setCursor(13,1);
lcd.print("%");
}
void loop()
{
delay(1000);
while(1)
{
delay(1000);
pinMode(gate,OUTPUT);
digitalWrite(gate,LOW);
delay(20);
digitalWrite(gate,HIGH);
pinMode(gate,INPUT_PULLUP);//by default it will become high due to internal
pull up
// delayMicroseconds(40);
duration=pulseIn(gate, LOW);
if(duration = 72)
{
while(1)
{
duration=pulseIn(gate, HIGH);
if(duration = 20){
value=0;}
else if(duration = 65){
value=1;}
else if(z==40){
break;}
i[z/8] |=value<<(7- (z%8));
j[z]=value;
z++;
}
}
answer=i[0]+i[1]+i[2]+i[3];

```

```
if(answer==i[4] && answer!=0)
{
  lcd.setCursor(7,0);
  lcd.print(i[2]);
  lcd.setCursor(11,1);
  lcd.print(i[0]);
}
z=0;
i[0]=i[1]=i[2]=i[3]=i[4]=0;
}
}
```

### Applications

- DHT11 Relative Humidity and Temperature Sensor can be used in many applications like:
- HVAC (Heating, Ventilation and Air Conditioning) Systems
- Weather Stations
- Medical Equipment for measuring humidity
- Home Automation Systems
- Automotive and other weather control applications

### How to Interface LCD (Liquid Crystal Display) Using An Arduino

In Arduino based embedded system design, the Liquid Crystal Display modules play a very important role. Hence it is very important to learn about [how to interface LCD](#) with an Arduino of 16×2 in embedded system design. The display units are very important in communication between the human world and the machine world. The display unit work on the same principle, it does not depend on the size of the display it may be big or the small. We are working with the simple displays like 16×1 and 16×2 units. The 16×1 display unit has the 16 characters which present in one line and 16×2 display units have 32 characters which are present in the 2 line. We should know that to display the each character there are 5×10 pixels. Thus to display one character all the 50 pixels should be together. In the display, there is a controller which is HD44780 it is used to control the pixels of characters to display

### What is a Liquid Crystal Display?

The [liquid crystal display](#) uses the property of light monitoring of liquid crystal and they do not emit the light directly. The Liquid crystal display is a flat panel display or the electronic visual display. With low information, content the LCD's are obtained in the fixed image or the arbitrary image which are displayed or

hidden like present words, digits, or [7 segment display](#). The arbitrary images are made up of large no of small pixels and the element has larger elements.



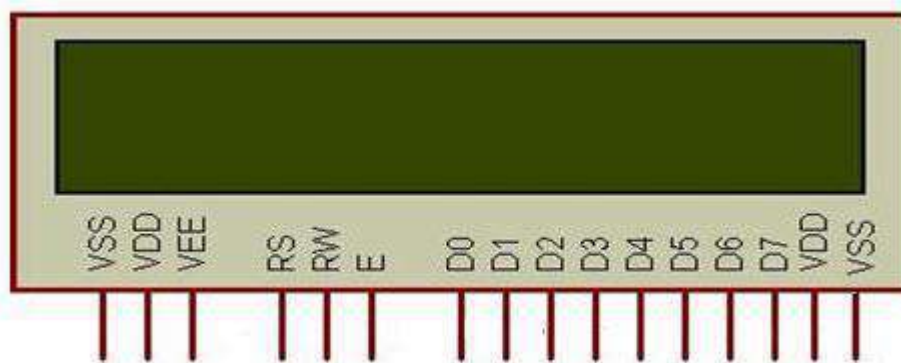
### Liquid Crystal Display of 16x2

The 16x2 liquid crystal display contains two horizontal lines and they are used for compressing the space of 16 display characters. Inbuilt, the LCD has two registers which are described below.

- Command Register
- Data Register

**Command Register:** This register is used to insert a special command in the LCD. The command is a special set of data and it is used to give the internal command to the liquid crystal display like clear screen, move to line 1 character 1, setting the curser and etc.

**Data Register:** The data registers are used to enter the line in the LCD



Pin diagram and description of each pin have explained in the following table.

Pin No	Pin Name	Pin Description
Pin 1	GND	This pin is a ground pin and the LCD is connected to the Ground
Pin 2	VCC	The VCC pin is used to supply the power to the LCD
Pin 3	VEE	This pin is used for adjusting the contrast of the LCD by connecting the variable resistor in between the VCC &

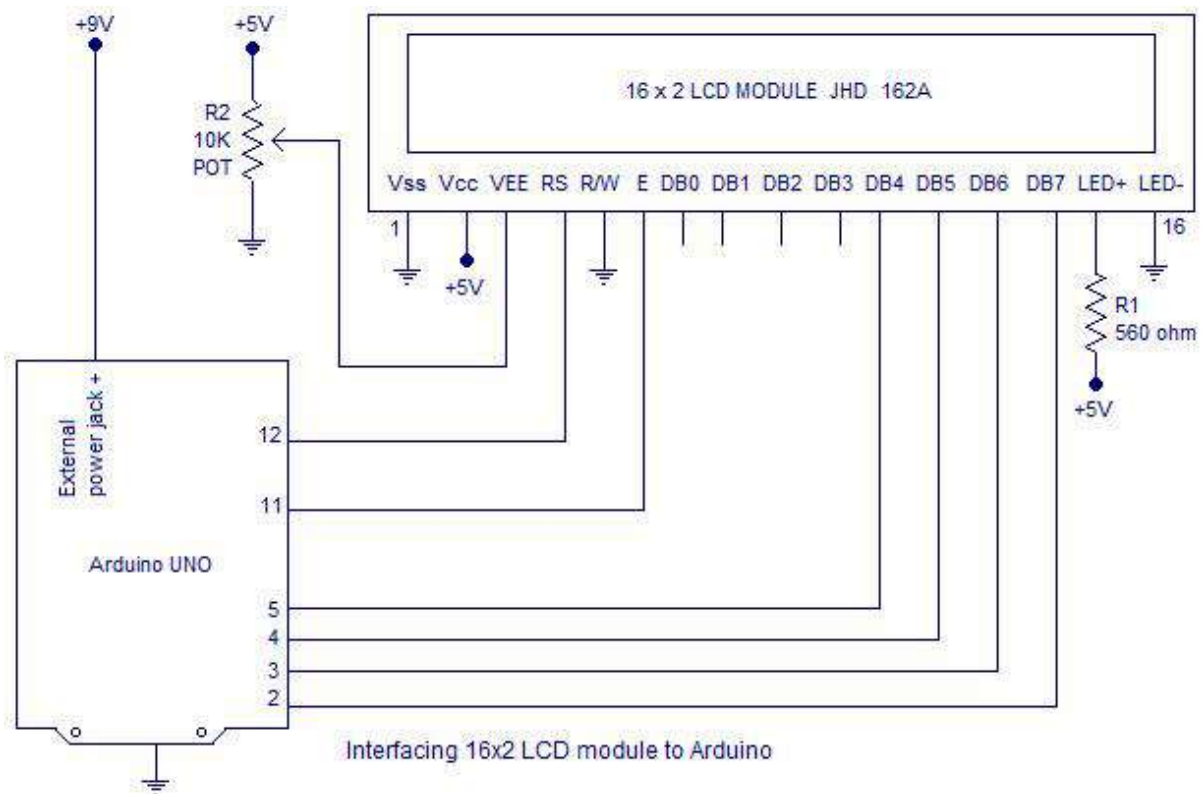
		Ground.
Pin 4	RS	The RS is known as register select and it selects the Command/Data register. To select the command register the RS should be equal to zero. To select the Data register the RS should be equal to one.
Pin 5	R/W	This pin is used to select the operations of Read/Write. To perform the write operations the R/W should be equal to zero. To perform the read operations the R/W should be equal to one.
Pin 6	EN	This is a enable signal pin if the positive pulses are passing through a pin, then the pin function as a read/write pin.
Pin 7	DB0 to DB7	The pin 7 contains total 8 pins which are used as a Data pin of LCD.
Pin 15	LED +	This pin is connected to VCC and it is used for the pin 16 to set up the glow of backlight of LCD.
Pin 16	LED –	This pin is connected to Ground and it is used for the pin 15 to set up the glow of backlight of the LCD.

### LCD Interfacing with the Arduino Module

The following circuit diagram shows the liquid crystal display with the [Arduino module](#). From the circuit diagram, we can observe that the RS pin of the LCD is connected to the pin 12 of the Arduino. The LCD of R/W pin is connected to the ground. The pin 11 of the Arduino is connected to the enable signal pin of LCD module. The LCD module & Arduino module are interfaced with the 4-bit mode in this project. Hence there are four input lines which are DB4 to DB7 of the LCD. This process very simple, it requires fewer connection cables and also we can utilize the most potential of the LCD module.

The digital input lines (DB4-DB7) are interfaced with the Arduino pins from 5-2. To adjust the contrast of the display here we are using a 10K potentiometer. The current through the back LED light is from the 560-ohm resistor. The external power jack is provided by the board to the Arduino. Using the PC through the USB port the Arduino can power. Some parts of the circuit can require the +5V power supply it is taken from the 5V source on the Arduino board.

The following schematic diagram shows the LCD module interfacing with the Arduino.



### Program – Arduino to LCD

```
#include<LiquidCrystal.h>
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // sets the interfacing pins
```

```
void setup()
```

```
{
  lcd.begin(16, 2); // initializes the 16x2 LCD
}
```

```
void loop()
```

```
{
  lcd.setCursor(0,0); //sets the cursor at row 0 column 0
  lcd.print("16x2 LCD MODULE"); // prints 16x2 LCD MODULE
  lcd.setCursor(2,1); //sets the cursor at row 1 column 2
  lcd.print("HELLO WORLD"); // prints HELLO WORLD
}
```

- Interfacing Air Quality Sensor-pollution (e.g. MQ135) – display data on LCD , switch on LED when data sensed is higher than specified value.

In this project we are going to make an **IoT Based Air Pollution Monitoring System** in which we will **monitor the Air Quality over a webserver using internet** and will trigger a alarm when the air quality goes down beyond a certain level, means when there are sufficient amount of harmful gases are present in the air like CO<sub>2</sub>, smoke, alcohol, benzene and NH<sub>3</sub>. It will show the air quality in PPM on the LCD and as well as on webpage so that we can monitor it very easily.

Previously we have built the [LPG detector using MQ6 sensor](#) and [Smoke detector using MQ2 sensor](#) but this time we have used MQ135 sensor as the air quality sensor which is the best choice for monitoring Air Quality as it can detects most harmful gases and can measure their amount accurately. In this [IOT project](#), you can monitor the pollution level from anywhere using your computer or mobile. We can install this system anywhere and can also trigger some device when pollution goes beyond some level, like we can switch on the Exhaust fan or can send alert SMS/mail to the user.

**Required Components:**

- MQ135 Gas sensor
- Arduino Uno
- Wi-Fi module ESP8266
- 16X2 LCD
- Breadboard
- 10K potentiometer
- 1K ohm resistors
- 220 ohm resistor
- Buzzer

**Circuit Diagram and Explanation:**

First of all we will connect the **ESP8266 with the Arduino**. ESP8266 runs on 3.3V and if you will give it 5V from the Arduino then it won't work properly and it may get damage. Connect the VCC and the CH\_PD to the 3.3V pin of Arduino. The RX pin of ESP8266 works on 3.3V and it will not communicate with the Arduino when we will connect it directly to the Arduino. So, we will have to make a voltage divider for it which will convert the 5V into 3.3V. This can be done by connecting three resistors in series like we did in the circuit. Connect the TX pin of the ESP8266 to the pin 10 of the Arduino and the RX pin of the esp8266 to the pin 9 of Arduino through the resistors.

ESP8266 Wi-Fi module gives your projects **access to Wi-Fi or internet**. It is a very cheap device and make your projects very powerful. It can communicate

with any microcontroller and it is the most leading devices in the [IOT platform](#). Learn more about [using ESP8266 with Arduino here](#).

Then we will connect the **MQ135 sensor with the Arduino**. Connect the VCC and the ground pin of the sensor to the 5V and ground of the Arduino and the Analog pin of sensor to the A0 of the Arduino.

Connect a buzzer to the pin 8 of the Arduino which will start to beep when the condition becomes true.

In last, we will [connect LCD with the Arduino](#). The connections of the LCD are as follows

- Connect pin 1 (VEE) to the ground.
- Connect pin 2 (VDD or VCC) to the 5V.
- Connect pin 3 (VO) to the middle pin of the 10K potentiometer and connect the other two ends of the potentiometer to the VCC and the GND. The potentiometer is used to control the screen contrast of the LCD. Potentiometer of values other than 10K will work too.
- Connect pin 4 (RS) to the pin 12 of the Arduino.
- Connect pin 5 (Read/Write) to the ground of Arduino. This pin is not often used so we will connect it to the ground.
- Connect pin 6 (E) to the pin 11 of the Arduino. The RS and E pin are the control pins which are used to send data and characters.
- The following four pins are data pins which are used to communicate with the Arduino.

Connect pin 11 (D4) to pin 5 of Arduino.

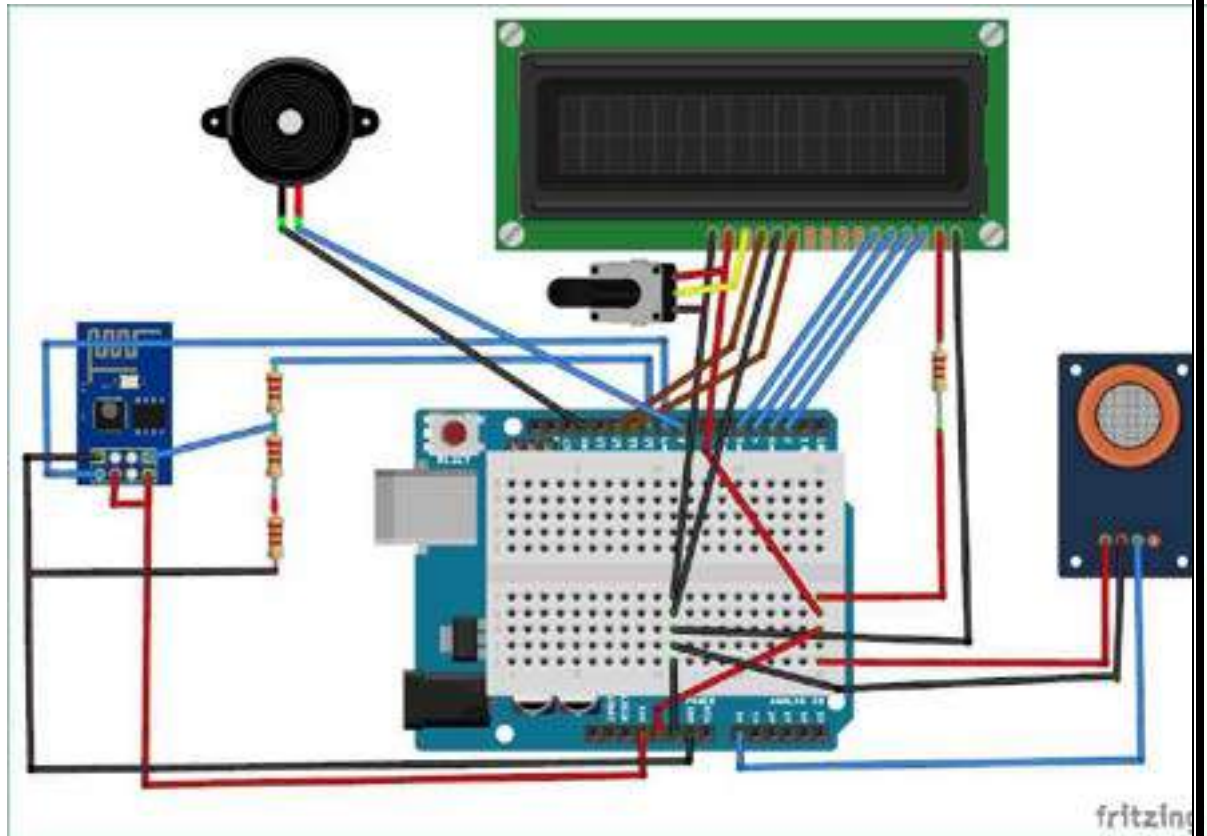
Connect pin 12 (D5) to pin 4 of Arduino.

Connect pin 13 (D6) to pin 3 of Arduino.

Connect pin 14 (D7) to pin 2 of Arduino.

- Connect pin 15 to the VCC through the 220 ohm resistor. The resistor will be used to set the back light brightness. Larger values will make the back light much more darker.
- Connect pin 16 to the Ground.





#### Working Explanation:

The MQ135 sensor can sense NH<sub>3</sub>, NO<sub>x</sub>, alcohol, Benzene, smoke, CO<sub>2</sub> and some other gases, so it is perfect gas sensor for our **Air Quality Monitoring Project**. When we will connect it to Arduino then it will sense the gases, and we will get the Pollution level in PPM (parts per million). MQ135 gas sensor gives the output in form of voltage levels and we need to convert it into PPM. So for converting the output in PPM, here we have used a library for MQ135 sensor, it is explained in detail in “Code Explanation” section below.

Sensor was giving us value of 90 when there was no gas near it and the safe level of air quality is 350 PPM and it should not exceed 1000 PPM. When it exceeds the limit of 1000 PPM, then it starts cause Headaches, sleepiness and stagnant, stale, stuffy air and if exceeds beyond 2000 PPM then it can cause increased heart rate and many other diseases.

When the value will be less than 1000 PPM, then the LCD and webpage will display “Fresh Air”. Whenever the value will increase 1000 PPM, then the buzzer will start beeping and the LCD and webpage will display “Poor Air, Open Windows”. If it will increase 2000 then the buzzer will keep beeping and the LCD and webpage will display “Danger! Move to fresh Air”.

#### Code Explanation:

Using library of MQ135 you can directly get the PPM values, by just using the below two lines:

```
MQ135 gasSensor = MQ135(A0);
```

```
float air_quality = gasSensor.getPPM();
```

But before that we need to **calibrate the MQ135 sensor**, for calibrating the sensor upload the below given code and let it run for 12 to 24 hours and then get the *RZERO* value.

```
#include "MQ135.h"
void setup (){
  Serial.begin (9600);
}
void loop() {
  MQ135 gasSensor = MQ135(A0); // Attach sensor to pin A0
  float rzero = gasSensor.getRZero();
  Serial.println (rzero);
  delay(1000);
}
```

After getting the *RZERO* value. **Put the RZERO value in the library file** you downloaded "MQ135.h": `#define RZERO 494.63`

Now we can begin the actual code for our Air quality monitoring project.

In the code, first of all we have defined the libraries and the variables for the Gas sensor and the LCD. By using the Software Serial Library, we can make any digital pin as TX and RX pin. In this code, we have made Pin 9 as the RX pin and the pin 10 as the TX pin for the ESP8266. Then we have included the library for the LCD and have defined the pins for the same. We have also defined two more variables: one for the sensor analog pin and other for storing *air\_quality* value.

```
#include <SoftwareSerial.h>
#define DEBUG true
SoftwareSerial esp8266(9,10);
#include <LiquidCrystal.h>
LiquidCrystallcd(12,11, 5, 4, 3, 2);
const int sensorPin= 0;
int air_quality;
```

Then we will declare the pin 8 as the output pin where we have connected the buzzer. `lcd.begin(16,2)` command will start the LCD to receive data and then we will set the cursor to first line and will print the '*circuitdigest*'. Then we will set the cursor on the second line and will print '*Sensor Warming*'.

```
pinMode(8, OUTPUT);
lcd.begin(16,2);
```

```
lcd.setCursor (0,0);  
lcd.print ("circuitdigest ");  
lcd.setCursor (0,1);  
lcd.print ("Sensor Warming ");  
delay(1000);
```

Then we will set the baud rate for the serial communication. Different ESP's have different baud rates so write it according to your ESP's baud rate. Then we will send the commands to set the ESP to communicate with the Arduino and show the IP address on the serial monitor.

```
Serial.begin(115200);  
esp8266.begin(115200);  
sendData("AT+RST\r\n",2000,DEBUG);  
sendData("AT+CWMODE=2\r\n",1000,DEBUG);  
sendData("AT+CIFSR\r\n",1000,DEBUG);  
sendData("AT+CIPMUair_quality=1\r\n",1000,DEBUG);  
sendData("AT+CIPSERVER=1,80\r\n",1000,DEBUG);  
pinMode(sensorPin, INPUT);  
lcd.clear();
```

For [printing the output on the webpage](#) in web browser, we will have to use **HTML programming**. So, we have created a string named *webpage* and stored the output in it. We are subtracting 48 from the output because the *read()* function returns the ASCII decimal value and the first decimal number which is 0 starts at 48.

```
if(esp8266.available())  
{  
if(esp8266.find("+IPD,"))  
{  
delay(1000);  
int connectionId = esp8266.read()-48;  
String webpage = "<h1>IOT Air Pollution Monitoring System</h1>";  
webpage += "<p><h2>";  
webpage+= " Air Quality is ";  
webpage+= air_quality;  
webpage+=" PPM";  
webpage += "<p>";
```

The following code will call a function named *sendData* and will send the data & message strings to the webpage to show.

```
sendData(cipSend,1000,DEBUG);
sendData(webpage,1000,DEBUG);
```

```
cipSend = "AT+CIPSEND=";
cipSend += connectionId;
cipSend += ",";
cipSend +=webpage.length();
cipSend += "\r\n";
```

The following code will print the data on the LCD. We have applied various conditions for checking air quality, and LCD will print the messages according to conditions and buzzer will also beep if the pollution goes beyond 1000 PPM.

```
lcd.setCursor (0, 0);
lcd.print ("Air Quality is ");
lcd.print (air_quality);
lcd.print (" PPM ");
lcd.setCursor (0,1);
if (air_quality<=1000)
{
lcd.print("Fresh Air");
digitalWrite(8, LOW);
```

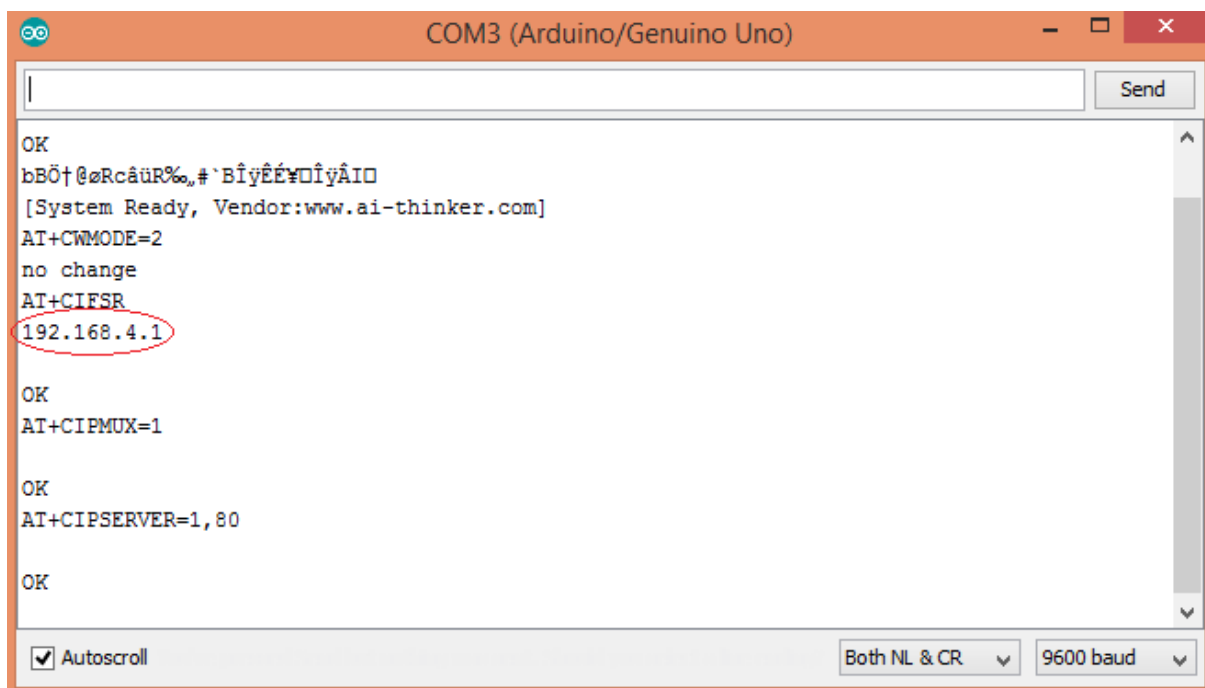
Finally the below function will send and show the data on the webpage. The data we stored in string named '*webpage*' will be saved in string named '*command*'. The ESP will then read the character one by one from the '*command*' and will print it on the webpage.

```
String sendData(String command, const int timeout, boolean debug)
{
  String response = "";
  esp8266.print(command); // send the read character to the esp8266
  long int time = millis();
  while( (time+timeout) >millis())
  {
  while(esp8266.available())
  {
    // The esp has data so display its output to the serial window
    char c = esp8266.read(); // read the next character.
    response+=c;
  }
}
}
```

```
if(debug)
{
Serial.print(response);
}
return response;
}
```

### Testing and Output of the Project:

Before uploading the code, make sure that you are connected to the Wi-Fi of your ESP8266 device. After uploading, open the serial monitor and it will show the IP address like shown below.



The screenshot shows the Serial Monitor window for COM3 (Arduino/Genuino Uno). The output text is as follows:

```
OK
bBÖ†@øRcâüR%,#`BiyÊÉ¥DiyÂID
[System Ready, Vendor:www.ai-thinker.com]
AT+CWMODE=2
no change
AT+CIFSR
192.168.4.1
OK
AT+CIPMUX=1
OK
AT+CIPSERVER=1,80
OK
```

The IP address `192.168.4.1` is circled in red. At the bottom of the window, the 'Autoscroll' checkbox is checked, and the baud rate is set to 9600.

Type this IP address in your browser, it will show you the output as shown below. You will have to refresh the page again if you want to see the current Air Quality Value in PPM.



# IOT Air Pollution Monitoring System

**Air Quality is 977 PPM**

**Good Air**

We have setup a local server to demonstrate its working, you can check the **Video** below. But to monitor the air quality from anywhere in the world, you need to **forward the port 80 (used for HTTP or internet) to your local or private IP address (192.168\*)** of you device. After port forwarding all the incoming connections will be forwarded to this local address and you can open above shown webpage by just entering the public IP address of your internet from anywhere. You can forward the port by logging into your router (192.168.1.1) and find the option to setup the port forwarding.

## Code (Merged)

```
#include "MQ135.h"
#include <SoftwareSerial.h>
#define DEBUG true
SoftwareSerial esp8266(9,10); // This makes pin 9 of Arduino as RX pin and pin 10 of Arduino as the
TX pin
const int sensorPin= 0;
int air_quality;
#include <LiquidCrystal.h>
LiquidCrystallcd(12,11, 5, 4, 3, 2);
void setup() {
pinMode(8, OUTPUT);
lcd.begin(16,2);
lcd.setCursor (0,0);
lcd.print ("circuitdigest ");
lcd.setCursor (0,1);
lcd.print ("Sensor Warming ");
```

```

delay(1000);
Serial.begin(115200);
esp8266.begin(115200); // your esp's baud rate might be different
sendData("AT+RST\r\n",2000,DEBUG); // reset module
sendData("AT+CWMODE=2\r\n",1000,DEBUG); // configure as access point
sendData("AT+CIFSR\r\n",1000,DEBUG); // get ip address
sendData("AT+CIPMUair_quality=1\r\n",1000,DEBUG); // configure for multiple connections
sendData("AT+CIPSERVER=1,80\r\n",1000,DEBUG); // turn on server on port 80
pinMode(sensorPin, INPUT); //Gas sensor will be an input to the arduino
lcd.clear();
}
void loop() {
MQ135 gasSensor = MQ135(A0);
float air_quality = gasSensor.getPPM();
if(esp8266.available()) // check if the esp is sending a message
{
  if(esp8266.find("+IPD,"))
  {
    delay(1000);
    int connectionId = esp8266.read()-48; /* We are subtracting 48 from the output because the
read() function returns the ASCII decimal value and the first decimal number which is 0 starts at
48*/
    String webpage = "<h1>IOT Air Pollution Monitoring System</h1>";
    webpage += "<p><h2>";
    webpage+= " Air Quality is ";
    webpage+= air_quality;
    webpage+=" PPM";
    webpage += "<p>";
    if (air_quality<=1000)
    {
      webpage+= "Fresh Air";
    }
    else if(air_quality<=2000 &&air_quality>=1000)
    {
      webpage+= "Poor Air";
    }
    else if (air_quality>=2000 )
    {
      webpage+= "Danger! Move to Fresh Air";
    }
    webpage += "</h2></p></body>";
    String cipSend = "AT+CIPSEND=";
    cipSend += connectionId;
    cipSend += ",";
    cipSend +=webpage.length();
    cipSend += "\r\n";

    sendData(cipSend,1000,DEBUG);
    sendData(webpage,1000,DEBUG);

    cipSend = "AT+CIPSEND=";

```

```

cipSend += connectionId;
cipSend += ",";
cipSend += webpage.length();
cipSend += "\r\n";

String closeCommand = "AT+CIPCLOSE=";
closeCommand+=connectionId; // append connection id
closeCommand+="\r\n";

sendData(closeCommand,3000,DEBUG);
}
}
lcd.setCursor (0, 0);
lcd.print ("Air Quality is ");
lcd.print (air_quality);
lcd.print (" PPM ");
lcd.setCursor (0,1);
if (air_quality<=1000)
{
lcd.print("Fresh Air");
digitalWrite(8, LOW);
}
else if( air_quality>=1000 &&air_quality<=2000 )
{
lcd.print("Poor Air, Open Windows");
digitalWrite(8, HIGH );
}
else if (air_quality>=2000 )
{
lcd.print("Danger! Move to Fresh Air");
digitalWrite(8, HIGH); // turn the LED on
}
lcd.scrollDisplayLeft();
delay(1000);
}
String sendData(String command, const int timeout, boolean debug)
{
String response = "";
esp8266.print(command); // send the read character to the esp8266
long int time = millis();
while( (time+timeout) >millis())
{
while(esp8266.available())
{
// The esp has data so display its output to the serial window
char c = esp8266.read(); // read the next character.
response+=c;
}
}
if(debug)
{

```



```
Serial.print(response);  
}  
return response;  
}
```

- Interfacing Bluetooth module (e.g. HC05)- receiving data from mobile phone on Arduino and display on LCD

Bluetooth is a one of the great example for wireless connectivity. It is used in many fields. Bluetooth consumes very small amount of energy. Do you know about Smartphone controlled robot or car. Commonly one of these two wireless technology is used in Smartphone controlled robot. One is WIFI and other is Bluetooth. And another commonly used wireless technology for controlling Robot car is RF. Which is the same remote and receiver used in drones. Here we are going to interface a Bluetooth Module (HC-05) with Arduino Uno. And describe each line of code. Then we control the builtin LED of Arduino Uno from smartphone via Bluetooth.

Before starting we must know about the HC-05

HC-05 Bluetooth Module



HC-05 is a Bluetooth module which can communicate in two way. Which means, It is full-duplex. We can use it with most micro controllers. Because it operates Serial Port Protocol (SSP). The module communicate with the help of USART (Universal Synchronous/Asynchronous Receiver/Transmitter ) at the baud rate of 9600. and it also support other baud rate. So we can interface this module with any microcontroller which supports USART. The HC-05 can operate in two modes. One is Data mode and other is AT command mode. When the enable pin is "LOW" the HC-05 is in Data Mode. If that pin set as "HIGH" the module is in AT command mode. Here we operate this module in Data Mode.

Technical Specifications

- Operating Voltage: 4V to 6V (Typically +5V)
- Operating Current: 30mA

- Range: <100m
- Works with Serial communication (USART) and TTL compatible
- Can be easily interfaced with Laptop or Mobile phones with Bluetooth

You can see the more about the module in the datasheet.

it's time to start.

Step - 1

First I am going to create a sketch for Arduino Uno to Interface the HC-05  
Open Arduino IDE.

Here we make this project without any library. First declare a variable named "inputByte" as char datatype. Alternatively you can use any variable name. Here we use the character to control the LED. And set the initial value of "inputByte" is "z" (it is lowercase). Why it is "z" ? See the String table. It is used to set the initial state of LED as "LOW" ( When turn on the Arduino, Set the LED is off).

```
char inputByte='z';
```

Step - 2

Next we need to code the setup part. HC-05 use the serial communication. So begin the serial communication by using the function "Serial.begin()". Set the baud rate as 9600. Then set the digital pin 13 as a "OUTPUT" pin. Because this is the pin which internally connected to the inbuilt LED.

```
void setup(){
  Serial.begin(9600);
  pinMode(13,OUTPUT);
}
```

Setup part is completed.

Step - 3

Next I am going to code the loop part. Use a while loop and the function "Serial.available()". This function returns the number of bytes available to read. The body part of while loop works only the "Serial.available()" is greater than 0. Then read the data available in serial port. For that, I use the function "Serial.read()". Then store it to "inputByte". Then use an "if" condition. Make the pin 13 "HIGH" if the "inputByte" is "Z" (upper case). Because the the App will send "z" when the button is in ON mode. This is for turn on the LED. Next I use a "else if" condition to turn off the LED. condition for turn off the LED is "inputByte==z"(lower case). Because the the App will send "Z" when the button is in OFF mode. For more see the String table.

```
void loop() {
  while(Serial.available(>)0){
    inputByte= Serial.read();
```

```
Serial.println(c);
if (inputByte=='Z'){
digitalWrite(13,HIGH);
}
elseif (inputByte=='z'){
digitalWrite(13,LOW);
}
}
}
```

The loop part is completed. You can see the complete code in the code section of this article. Then upload the code to Arduino Uno

Please make sure the Rx and Tx pin of HC-05 remove from Arduino Uno while uploading. Otherwise you may encounter with some problem to upload the code.

Step - 4

Connections

Arduino Uno HC-05

Rx - Tx

Tx - Rx

5v - +5v

GND - GND

Connection is completed. After turn on the Arduino Uno, The indicator LED in the HC-05 will start blinking continuously

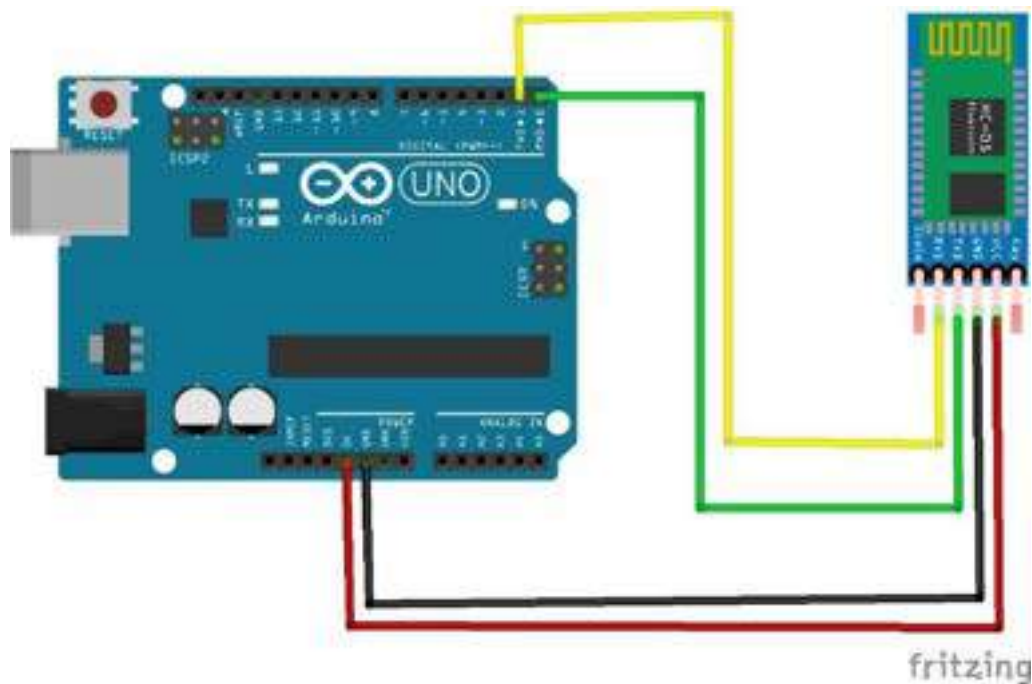
## CODE

```
//sketch created by Akshay Joseph
char inputByte;
void setup(){
Serial.begin(9600);
pinMode(13,OUTPUT);

}

void loop(){
while(Serial.available(>0){
inputByte=Serial.read();
Serial.println(inputByte);
if(inputByte=='Z'){
digitalWrite(13,HIGH);
}
elseif(inputByte=='z'){
digitalWrite(13,LOW);
}
}
}
```

## SCHEMATICS

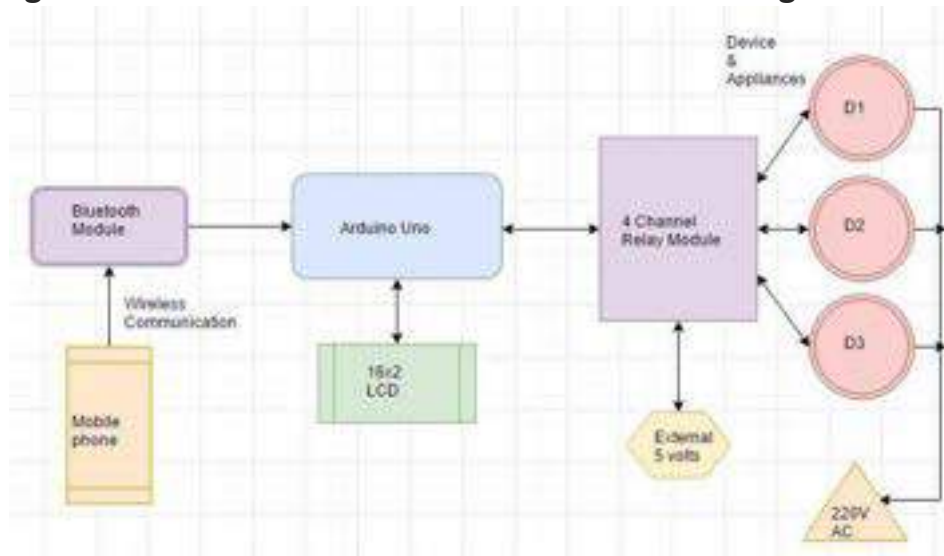


Interfacing Relay module to demonstrate Bluetooth based home automation application. (using Bluetooth and relay)

Bluetooth Based Home Automation project using Arduino

In this project, Bluetooth is used to communicate with Arduino using an Android Application know as **S2 Terminal**.

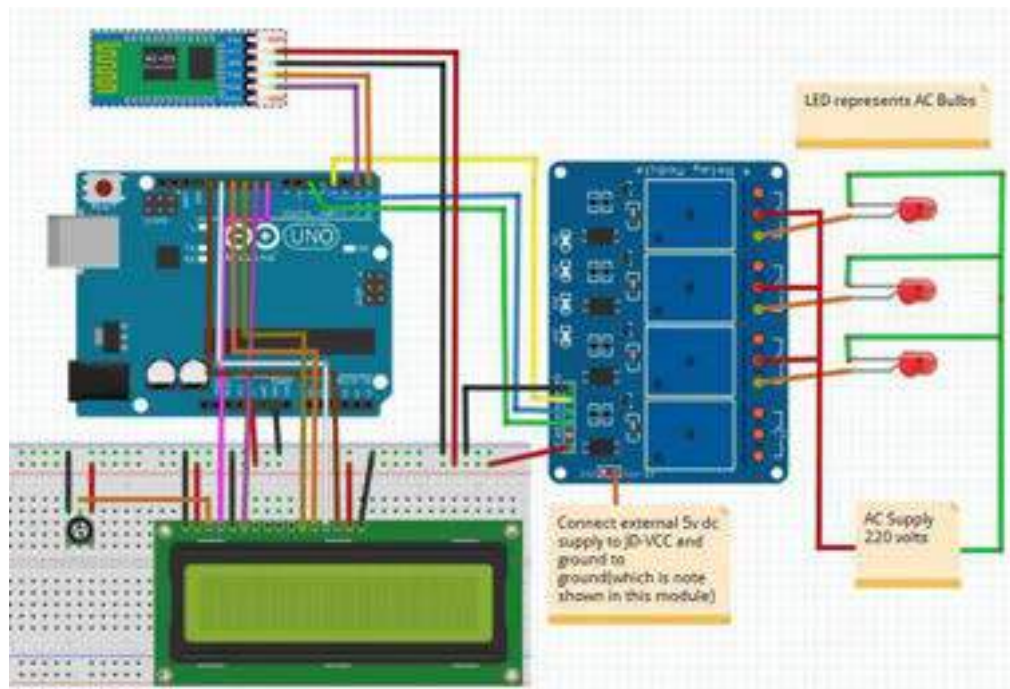
**Block Diagram of Bluetooth Based Home Automation using Arduino**



### **Components Required for Bluetooth Based Home Automation using Arduino**

- Arduino Uno: We use Arduino due to its simplicity and it also provides a much digital pin to interface with LCD and relay module at the same time. It is very useful when you prototyping a project.
- HC-05 Bluetooth Module: Bluetooth is very easy to interface with Arduino. If you are not familiar with it search it on our website.
- 4 Channel Relay Modules: The module we use in this project is HL-54S. It switches on and off using 5v logical signal from Arduino. It can bear up to 250VAC and 10A. These modules have 4 channels so we can control 4 AC devices at a time.
- 16×2 LCD: LCD is used to display project name, a list of commands which can be entered then it asks to give any command and show the status of the command which is entered. We use 16×2 LCD because it is easy to interface with Arduino and very cheap in price. 10k potentiometer is used to control the contrast of display
- AC bulbs with holders: AC bulbs are used to represent devices and appliances. Because it is easy to handle and very useful when you are prototyping any AC project. In final product just replace with AC socket to control.
- AC wire with plug: I advise you to use good quality wire when working with higher voltages. It is always good to use electrical tape to protect connections.
- External 5 volt supply: 5-volt dc supply is required to switch a relay on and off. Otherwise, it did not work. Also do not supply 5v from Arduino.

### **Circuit Diagram of Bluetooth Based Home Automation using Arduino**



Connections:

#### Bluetooth Module HC-05

- HC-05 Rx to Arduino Tx.
- HC-05 Tx to Arduino Rx.
- Vcc to 5v
- Ground to ground.

#### 16x2 LCD:

- $V_{SS}$  to a ground.
- $V_{DD}$  to supply voltage.
- $V_0$  to adjust pin of 10k potentiometer.
- RS to Pin 8.
- RW to a ground.
- Enable to Pin 9.
- LCD D4 to Pin 10.
- LCD D5 to Pin 11.
- LCD D6 to Pin 12.
- LCD D7 to Pin 13
- Ground one end of a potentiometer.
- 5v to other ends of a potentiometer.

#### 4 Channel Relay modules:

- External 5 volts to JD-VCC
- 
- Ground to ground.

- Ini1 to Pin 3.
- Ini2 to Pin 4.
- Ini3 to Pin5.
- Vcc to Arduino 5v.
- Connect one end of all bulbs to normally open terminal of relays.
- One end of 220VAC to all common terminals of a relay and another end with another terminal of bulbs.

### Working of Bluetooth Based Home Automation using Arduino

First of all download **S2 Terminal** app from Google play store. Open the application connects to the Bluetooth module. Write the specified commands and send it. Bluetooth module receives it and Arduino performs the describe operation, display status and send a message to mobile.

Commands:

Command sent by mobile	Message receives by mobile
<b>all on</b>	All ON
<b>all off</b>	All OFF
<b>white on</b>	White ON
<b>white off</b>	White OFF
<b>blue on</b>	Blue ON
<b>blue off</b>	Blue OFF
<b>green on</b>	Green ON
<b>green off</b>	Green OFF

Code for Bluetooth based home automation system

```
#include <LiquidCrystal.h>
LiquidCrystallcd(8, 9, 10, 11, 12, 13);
#define white 3
#define blue 4
#define green 5
int tx=1;
int rx=0;
char inSerial[15];
```

```
void setup(){
Serial.begin(9600);
pinMode(white, OUTPUT);
pinMode(blue, OUTPUT);
pinMode(green, OUTPUT);
pinMode(tx, OUTPUT);
pinMode(rx, INPUT);
digitalWrite(white, HIGH);
digitalWrite(blue, HIGH);
digitalWrite(green, HIGH);
lcd.begin(16, 2);
lcd.clear();
lcd.print("MICROCONTROLLERS ");
lcd.setCursor(0,1);
lcd.print(" LAB ");
delay(2000);
lcd.clear();
lcd.print("HOME AUTOMATION ");
lcd.setCursor(0,1);
lcd.print("USING BLUETOOTH");
delay(2000);
lcd.clear();
lcd.print("1. Bulb 1 WHITE");
lcd.setCursor(0,1);
lcd.print("2. Bulb 2 BLUE");
delay(2000);
lcd.clear();
lcd.print("3. Bulb 3 GREEN");
delay(2000);
lcd.clear();
lcd.print("Bulb 1 2 3 ");
lcd.setCursor(0,1);
lcd.print(" OFF OFF OFF");

}
```

```
void loop(){
int i=0;
int m=0;
delay(500);
if (Serial.available() > 0) {
while (Serial.available() > 0) {
inSerial[i]=Serial.read();
i++;
}
inSerial[i]='\0';
Check_Protocol(inSerial);
}}
```



```
void Check_Protocol(char inStr[]){  
int i=0;  
int m=0;  
Serial.println(inStr);
```

```
if(!strcmp(inStr,"all on")){  
digitalWrite(white, LOW);  
digitalWrite(blue, LOW);  
digitalWrite(green, LOW);  
Serial.println("ALL ON");  
lcd.setCursor(4,1);  
lcd.print("ON ");  
lcd.setCursor(8,1);  
lcd.print("ON ");  
lcd.setCursor(12,1);  
lcd.print("ON ");  
for(m=0;m<11;m++){  
inStr[m]=0;}  
i=0;}
```

```
if(!strcmp(inStr,"all off")){  
digitalWrite(white, HIGH);  
digitalWrite(blue, HIGH);  
digitalWrite(green, HIGH);  
Serial.println("ALL OFF");  
lcd.setCursor(4,1);  
lcd.print("OFF ");  
lcd.setCursor(8,1);  
lcd.print("OFF ");  
lcd.setCursor(12,1);  
lcd.print("OFF ");  
for(m=0;m<11;m++){  
inStr[m]=0;}  
i=0;}
```

```
if(!strcmp(inStr,"white on")){  
digitalWrite(white, LOW);  
Serial.println("White ON");  
lcd.setCursor(4,1);  
lcd.print("ON ");  
for(m=0;m<11;m++){  
inStr[m]=0;}  
i=0;}
```

```
if(!strcmp(inStr,"white off")){  
digitalWrite(white, HIGH);  
Serial.println("White OFF");  
lcd.setCursor(4,1);  
lcd.print("OFF ");
```

```
for(m=0;m<11;m++){
inStr[m]=0;}
i=0;}

if(!strcmp(inStr,"blue on")){

digitalWrite(blue, LOW);
Serial.println("Blue ON");
lcd.setCursor(8,1);
lcd.print("ON ");
for(m=0;m<11;m++){
inStr[m]=0;}
i=0;}

if(!strcmp(inStr,"blue off")){

digitalWrite(blue, HIGH);
Serial.println("Blue OFF");
lcd.setCursor(8,1);
lcd.print("OFF ");
for(m=0;m<11;m++){
inStr[m]=0;}
i=0;}
if(!strcmp(inStr,"green on")){

digitalWrite(green, LOW);
Serial.println("Green ON");
lcd.setCursor(12,1);
lcd.print("ON ");
for(m=0;m<11;m++){
inStr[m]=0;}
i=0;}

if(!strcmp(inStr,"green off")){

digitalWrite(green, HIGH);
Serial.println("Green OFF");
lcd.setCursor(12,1);
lcd.print("OFF ");
for(m=0;m<11;m++){
inStr[m]=0;}
i=0;}

else{
for(m=0;m<11;m++){
inStr[m]=0;
}
i=0;

}}
```