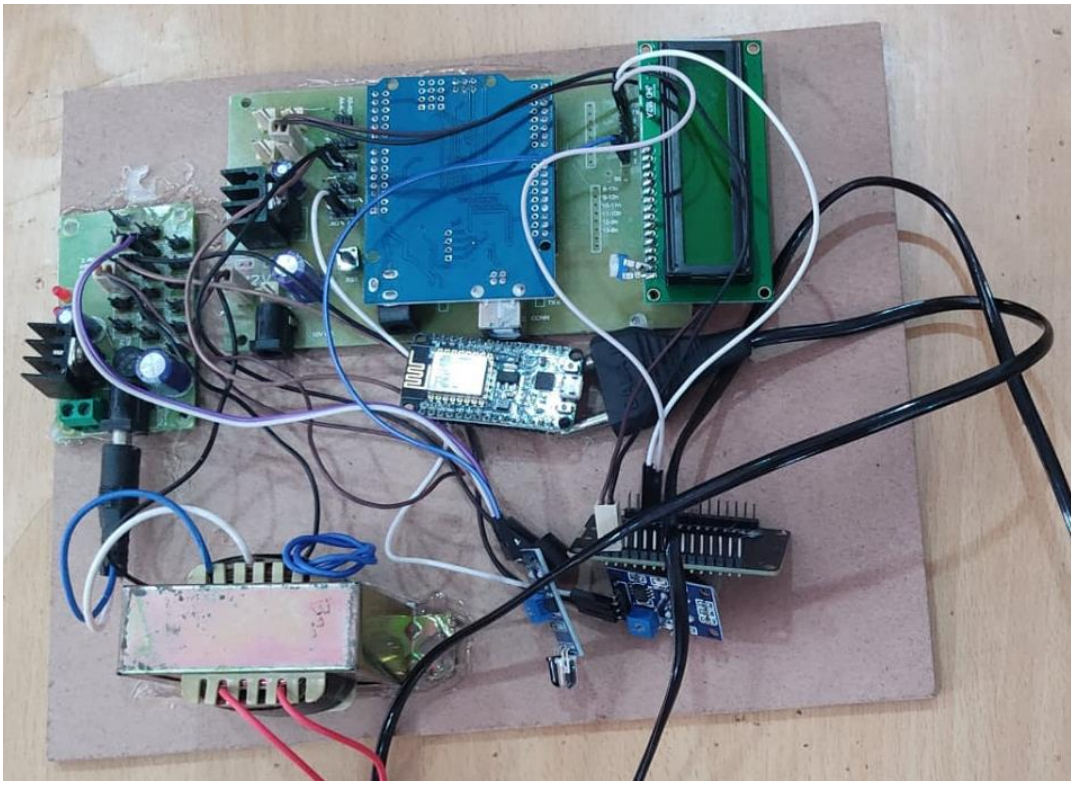
# IOT BASED HOME AUTOMATION



## Developed by:

1. **P SHABANA, ASST. PROFESSOR, DEPT. OF CSE, ALTS**
2. **K SWATHI, ASST. PROFESSOR, DEPT. OF CSE, ALTS**

# ABSTRACT:

This project presents the overall design of Home Automation System (HAS) with low cost and wireless system. It specifically focuses on the development of an IOT based home automation system that is able to control various components via internet or be automatically programmed to operate from ambient conditions. In this project, we design the development of a firmware for smart control which can successfully be automated minimizing human interaction to preserve the integrity within whole electrical devices in the home. We used Node MCU, a popular open source IOT platform, to execute the process of automation. Different components of the system will use different transmission mode that will be implemented to communicate the control of the devices by the user through Node MCU to the actual appliance. The main control system implements wireless technology to provide remote access from smart phone. We are using a cloud server-based communication that would add to the practicality of the project by enabling unrestricted access of the appliances to the user irrespective of the distance factor. We provided a data transmission network to create a stronger automation. The system intended to control electrical appliances and devices in house with relatively low cost design, user-friendly interface and ease of installation. The status of the appliance would be available, along with the control on an android platform. This system is designed to assist and provide support in order to fulfil the needs of elderly and disabled in home. Also, the smart home concept in the system improves the standard living at home

# CHAPTER 1

# EMBEDDED SYSTEMS

## 1.1. INTRODUCTION TO EMBEDDED SYSTEMS

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, sometimes with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems have become very important today as they control many of the common devices we use.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, Handheld computers share some elements with embedded systems — such as the operating systems and microprocessors which power them — but are not truly embedded systems, because they allow different applications to be load and peripherals to be connected.

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines, and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. Embedded systems that are programmable are provided with a programming interface, and embedded systems programming is a specialized occupation. Certain operating systems or language platforms are

tailored for the embedded market, such as Embedded Java and Windows XP Embedded. However, some low-end consumer products use very inexpensive microprocessors and limited storage, with the application and operating system both part of a single program. The program is written permanently into the system's memory in this case, rather than being loaded into RAM (random access memory), as programs on a personal computer are.

## 1.2. CHARACTERISTIC OF EMBEDDED SYSTEM

- Speed (bytes/sec): Should be high speed
- Power (watts): Low power dissipation
- Size and weight: As far as possible small in size and low weight
- Accuracy (%error): Must be very accurate
- Adaptability: High adaptability and accessibility
- Reliability: Must be reliable over a long period of time

## 1.3. APPLICATIONS OF EMBEDDED SYSTEMS

We are living in the Embedded World. You are surrounded with many embedded products and your daily life largely depends on the proper functioning of these gadgets. Television, Radio, CD player of your living room, Washing Machine or Microwave Oven in your kitchen, Card readers, Access Controllers, Palm devices of your work space enable you to do many of your tasks very effectively. Apart from all these, many controllers embedded in your car take care of car operations between the bumpers and most of the times you tend to ignore all these controllers.

- **Robotics:** industrial robots, machine tools, Robocop soccer robots
- **Automotive:** cars, trucks, trains
- **Aviation:** airplanes, helicopters
- Home and Building Automation
- **Aerospace:** rockets, satellites
- **Energy systems:** windmills, nuclear plants
- **Medical systems:** prostheses, revalidation machine.

## 1.4. MICROCONTROLLER VERSUS MICROPROCESSOR

What is the difference between a Microprocessor and Microcontroller? By microprocessor is meant the general purpose Microprocessors such as Intel's X86 family (8086, 80286, 80386, 80486, and the Pentium) or Motorola's 680X0 family (68000, 68010, 68020, 68030, 68040, etc). These microprocessors contain no RAM, no ROM, and no I/O ports on the chip itself. For this reason, they are commonly referred to as general-purpose Microprocessors.

A system designer using a general-purpose microprocessor such as the Pentium or the 68040 must add RAM, ROM, I/O ports, and timers externally to make them functional. Although the addition of external RAM, ROM, and I/O ports makes these systems bulkier and much more expensive, they have the advantage of versatility such that the designer can decide on the amount of RAM, ROM and I/O ports needed to fit the task at hand. This is not the case with Microcontrollers.

A Microcontroller has a CPU (a microprocessor) in addition to a fixed amount of RAM, ROM, I/O ports, and a timer all on a single chip. In other words, the processor, the RAM, ROM, I/O ports and the timer are all embedded together on one chip; therefore, the designer cannot add any external memory, I/O ports, or timer to it. The fixed amount of on-chip ROM, RAM, and number of I/O ports in Microcontrollers makes them ideal for many applications in which cost and space are critical.

In many applications, for example a TV remote control, there is no need for the computing power of a 486 or even an 8086 microprocessor. These applications most often require some I/O operations to read signals and turn on and off certain bits

## 1.5. MICROCONTROLLERS FOR EMBEDDED SYSTEMS

In the Literature discussing microprocessors, we often see the term Embedded System. Microprocessors and Microcontrollers are widely used in embedded system products. An embedded system product uses a microprocessor (or Microcontroller) to do one task only. A printer is an example of embedded system since the processor inside it performs one task only; namely getting the data and printing it. Contrast this with a Pentium based PC. A PC can be used for any number of applications such as word processor, print-server, bank teller terminal, Video game, network server, or Internet terminal. Software for a variety of applications can be

loaded and run. Of course the reason a pc can perform myriad tasks is that it has RAM memory and an operating system that loads the application software into RAM memory and lets the CPU run it.

In this robot as the fire sensor senses the fire, it senses the signal to microcontroller. In an Embedded system, there is only one application software that is typically burned into ROM. An x86 PC contains or is connected to various embedded products such as keyboard, printer, modem, disk controller, sound card, CD-ROM drives, mouse, and so on. Each one of these peripherals has a Microcontroller inside it that performs only one task.

# CHAPTER 2

# INTRODUCTION TO PROJECT

## 2.1. INTRODUCTION

## BLOCK DIAGRAM

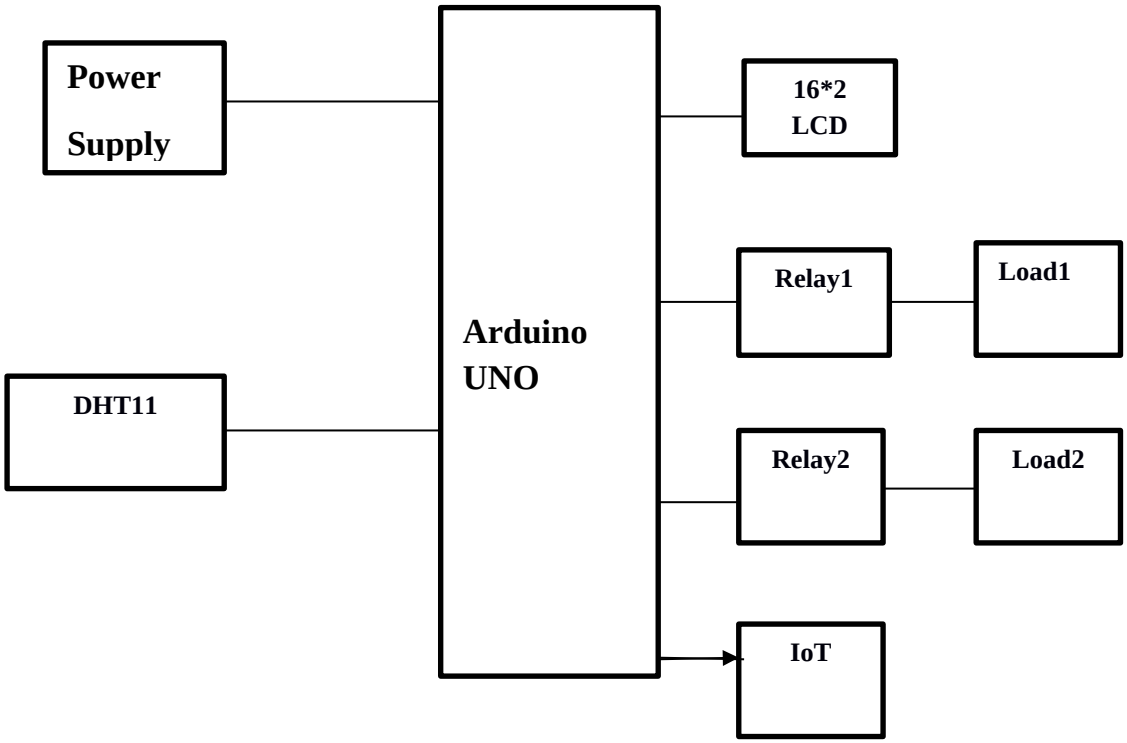



**Figure.2.1. Block Diagram**

# CHAPTER 3

# ARDUINO UNO

## 3.1 Microcontroller:

### 3.1.1 Introduction:

Microcontroller as the name suggest, a small controller. They are like single chip computers that are often embedded into other systems to function as processing/controlling unit. For example, the control you are using probably has microcontrollers inside that do decoding and other controlling functions. They are also used in automobiles, washing machines, microwaves ovens, toys….etc, where automation is needed.

### 3.1.2 Arduino Uno Microcontroller:

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digitalinput/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter. "Uno" means "One" in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5Vpin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V. The** regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3.3V.**A 3.3volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

**Memory:**

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the bootloader); It has also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

**Input and Output:**

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, arising or falling edge, or a change in value. See the attach Interrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.

- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which although provided by the underlying hardware, is not currently included in the Arduino language.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **I2C: 4 (SDA) and 5 (SCL).** Support I2C (TWI) communication using the Wire library. There are a couple of other pins on the board:
- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

**Communication:**

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USBCOM drivers, and no external driver is needed. However, on Windows, an *.inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus

### 3.1.3 ARDUINO UNO BOARD:

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains

everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

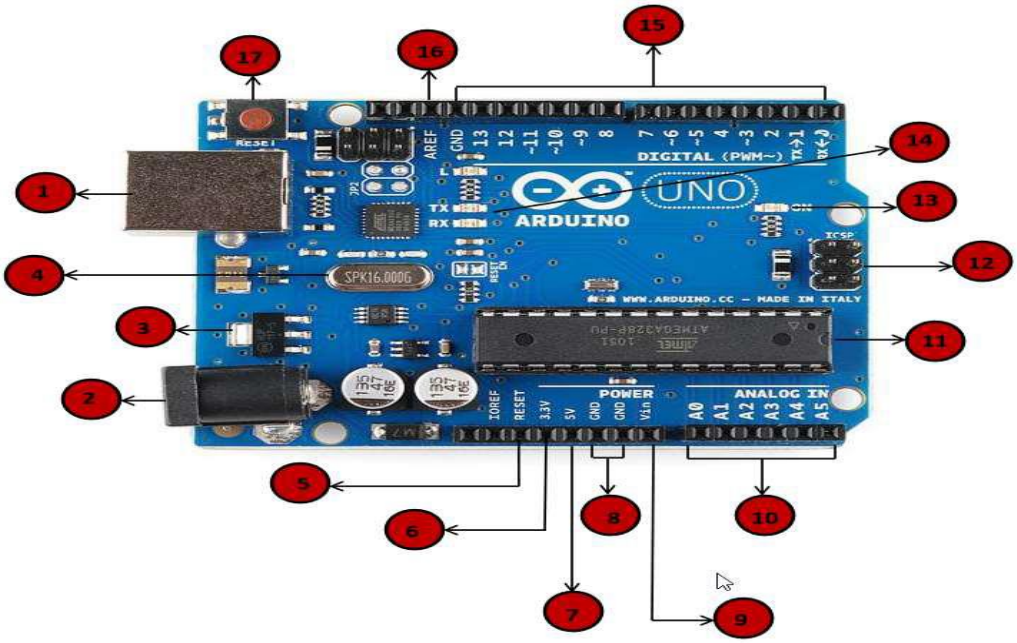



**Figure 3.1:** Arduino uno board

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converters.

### 5.1.3.1 Technical Specifications:

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

**Table 3.1:** Arduino uno specifications

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

**1. USB Interface:**

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

**2. External power supply:**

Arduino boards can be powered directly from the AC mains power supply by connecting it to the power supply (Barrel Jack)

**3. Voltage Regulator:**

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.

**4. Crystal Oscillator:**

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.

**5-17.Arduino Reset:**

It can reset your Arduino board, i.e., start your program from the beginning. It 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).

**6-9.Pins (3.3, 5, GND, Vin):**

- 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.

**10. Analog pins:**

The Arduino UNO board has five 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.

**11. Main microcontroller:**

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.

The Atmega8U2 programmed as a USB-to-serial converter. "Uno" means "One" in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards

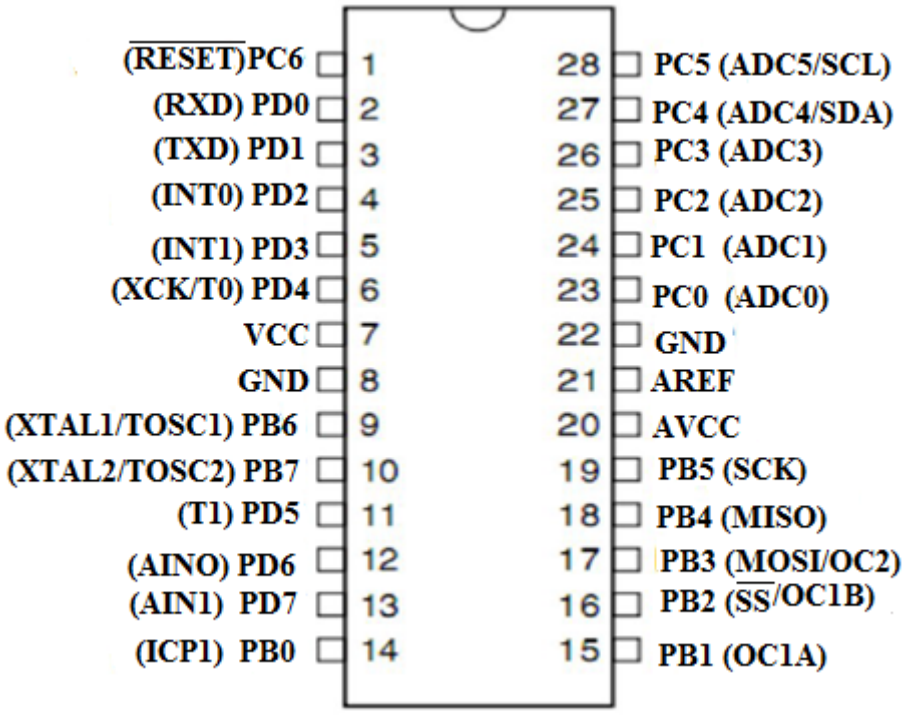



**Figure 3.2:** Pin diagram

### 3.1.3.2 Pin Description:

**VCC:** Digital supply voltage.

**GND:** Ground.

**Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2:**

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

**Port C (PC[5:0]):**

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs,

Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

**PC6/RESET:**

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

**Port D (PD[7:0]):**

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

**AVCC:** AVCC is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC[6:4] use digital supply voltage, VCC.

**AREF:** AREF is the analog reference pin for the A/D Converter.

**ADC [7:6] (TQFP and VFQFN Package Only):** In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

**12. ICSP pin:** 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.

**13. Power LED indicator:** 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.

**14. TX and RX LEDs:** 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.

**15. Digital I / O:** 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.

**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 working.

# CHAPTER 4

# HARDWARE COMPONENTS

## 4.1. POWER SUPPLY UNIT

The power supply for this system is shown below.

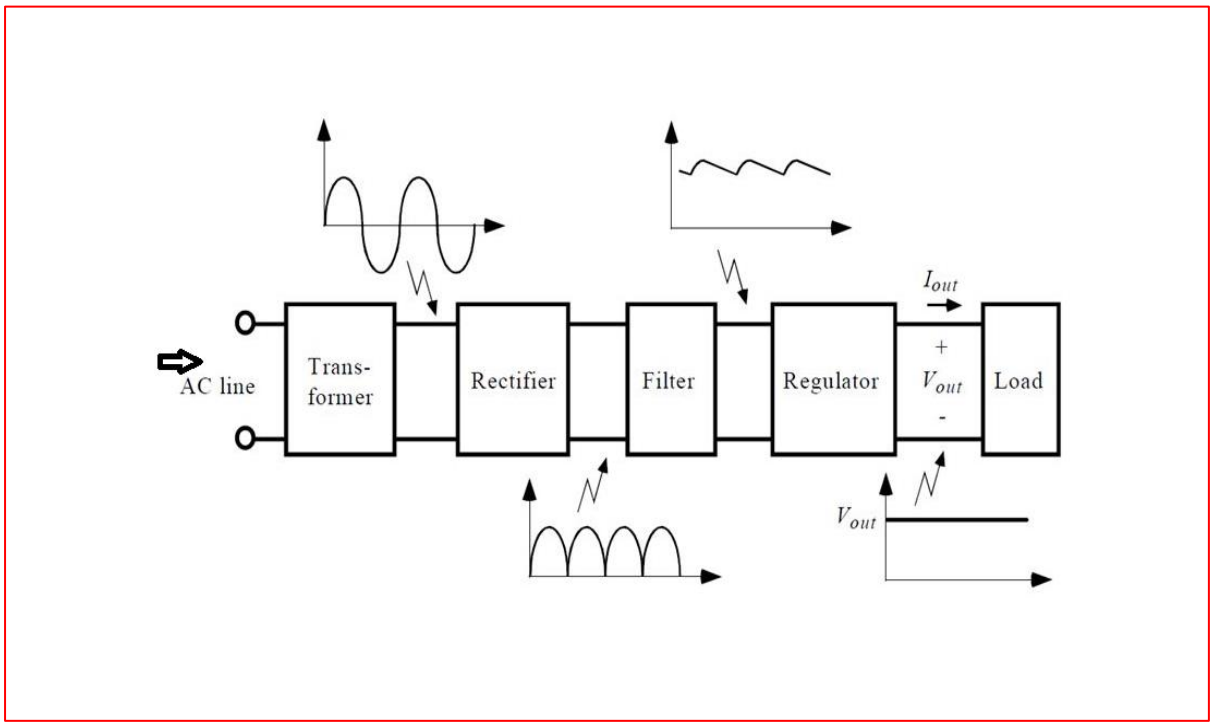



**Figure.4.1.power supply**

### 4.1.1. Transformer:

Transformer is a static device used to convert the voltage from one level to another level without change its frequency. There are two types of transformers

1. Step-up transformer
2. Step-down transformer

Step-up transformer converts low voltage level into high voltage level without change its frequency.

Step-down transformer converts high voltage level into low voltage level without change its frequency.

In this project we using step-down transformer which converts 230V AC to 12V AC [or] 230V AC to 5V as shown below.

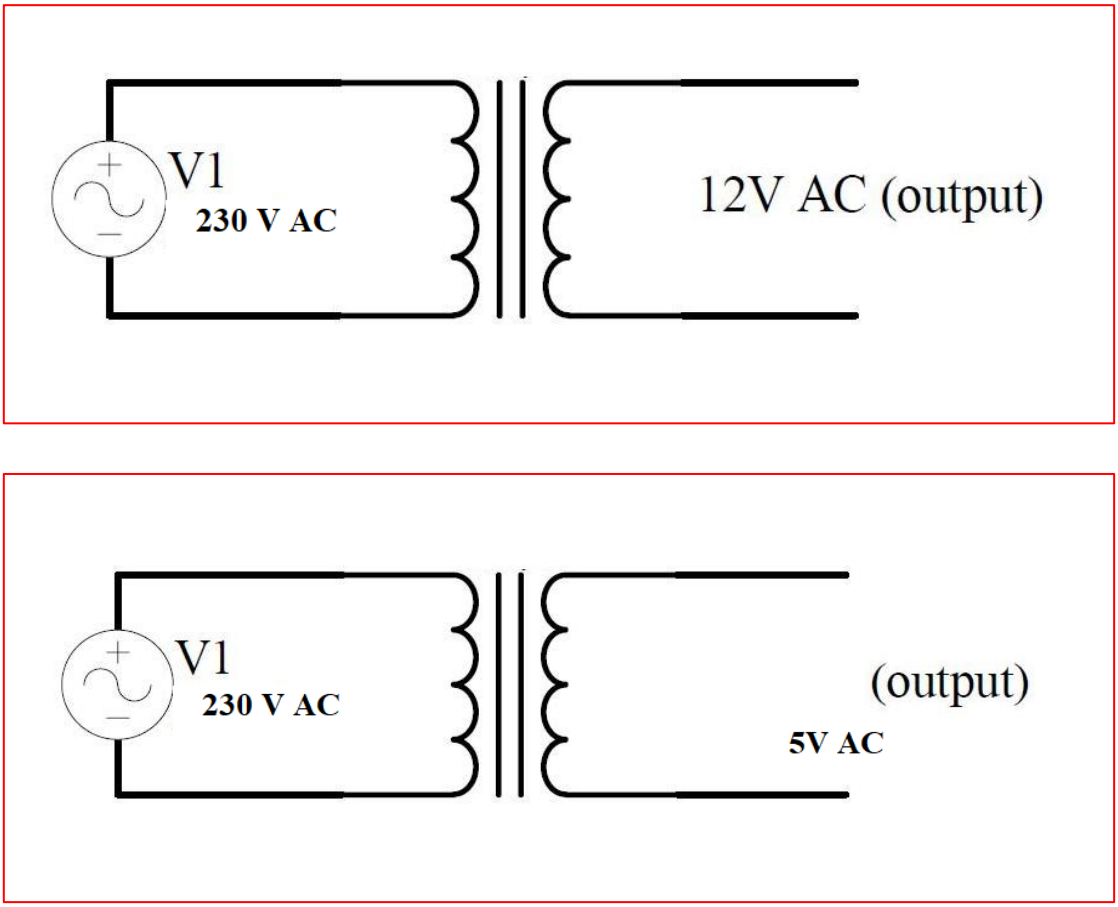



**Figure.4.2.Transformers**

### 4.1.2. Diodes:

Diodes allow electricity to flow in only one direction. The arrow of the circuit symbol shows the direction in which the current can flow. Diodes are the electrical version of a valve and early diodes were actually called valves.

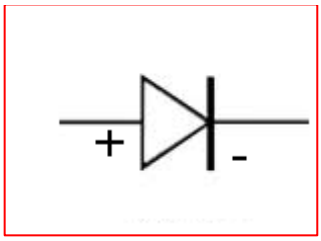



**Figure.4.3. Diode Symbol**

A **diode** is a device which only allows current to flow through it in one direction. In this direction, the diode is said to be 'forward-biased' and the only effect on the signal is that there

will be a voltage loss of around 0.7V. In the opposite direction, the diode is said to be 'reverse-biased' and no current will flow through it.

### 4.1.3. Rectifier

The purpose of a rectifier is to convert an AC waveform into a DC waveform (OR) Rectifier converts AC current or voltages into DC current or voltage. There are two different rectification circuits, known as **'half-wave'** and **'full-wave' rectifiers**. Both use components called **diodes** to convert **AC into DC**.

***The Half-wave Rectifier***

The half-wave rectifier is the simplest type of rectifier since it only uses one diode, as shown in figure.

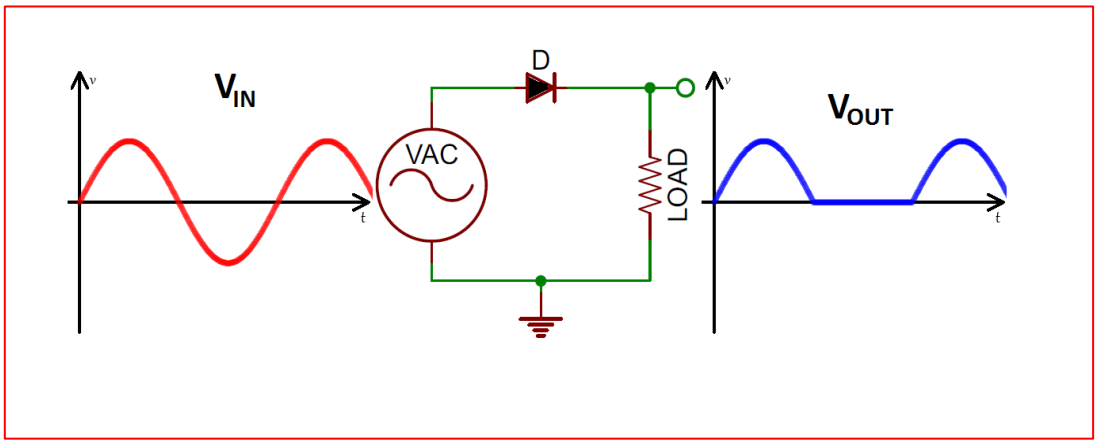



**Figure.4.4Half Wave Rectifier**

Figure 2 shows the AC input waveform to this circuit and the resulting output. As you can see, when the AC input is positive, the diode is forward-biased and lets the current through. When the AC input is negative, the diode is reverse-biased and the diode does not let any current through, meaning the output is 0V. Because there is a 0.7V voltage loss across the diode, the peak output voltage will be 0.7V less than Vs.

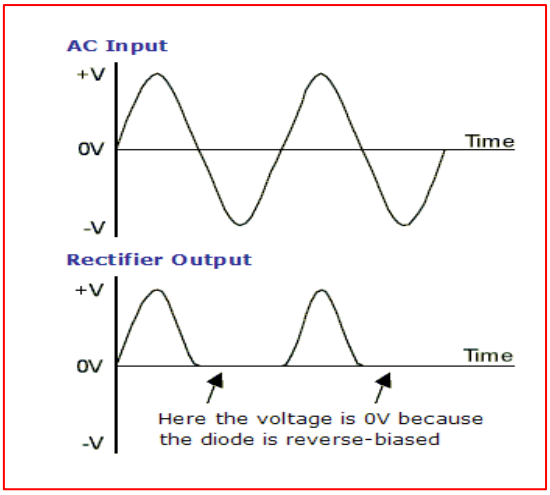



**Figure.4.5Half-Wave Rectification**

While the output of the half-wave rectifier is DC (it is all positive), it would not be suitable as a power supply for a circuit. Firstly, the output voltage continually varies between 0V and Vs-0.7V, and secondly, for half the time there is no output at all.

### *The Full-wave Bridge Rectifier*

The circuit in figure 3 addresses the second of these problems since at no time is the output voltage 0V. This time four diodes are arranged so that both the positive and negative parts of the AC waveform are converted to DC. The resulting waveform is shown in figure 4.

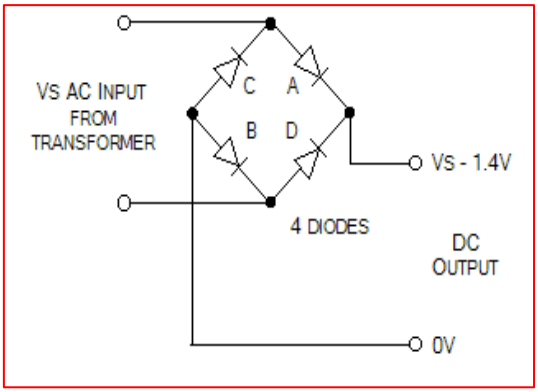



**Figure.4.6. Full-Wave Rectifier**

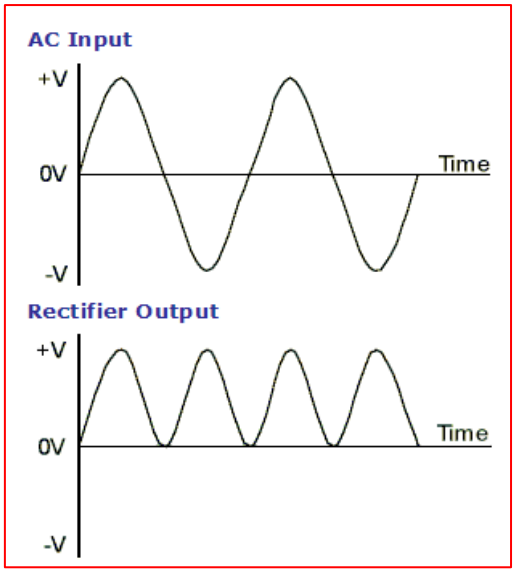



**Figure.4.7. Full-Wave Rectification**

When the AC input is positive, diodes A and B are forward-biased, while diodes C and D are reverse-biased. When the AC input is negative, the opposite is true - diodes C and D are forward-biased, while diodes A and B are reverse-biased.

While the full-wave rectifier is an improvement on the half-wave rectifier, its output still isn't suitable as a power supply for most circuits since the output voltage still varies between 0V and Vs-1.4V. So, if you put 12V AC in, you will 10.6V DC out.

### 4.1.4. Capacitor Filter

The **capacitor-input filter**, also called "Pi" filter due to its shape that looks like the Greek letterpi, is a type of electronic filter. Filter circuits are used to remove unwanted or undesired frequencies from a signal.

A typical capacitor input filter consists of a filter capacitor C1, connected across the rectifier output. The capacitor C1 offers low reactance to the AC component of the rectifier output while it offers infinite reactance to the DC component. As a result the AC components are going to ground. At that time DC components are feed to Regulator.

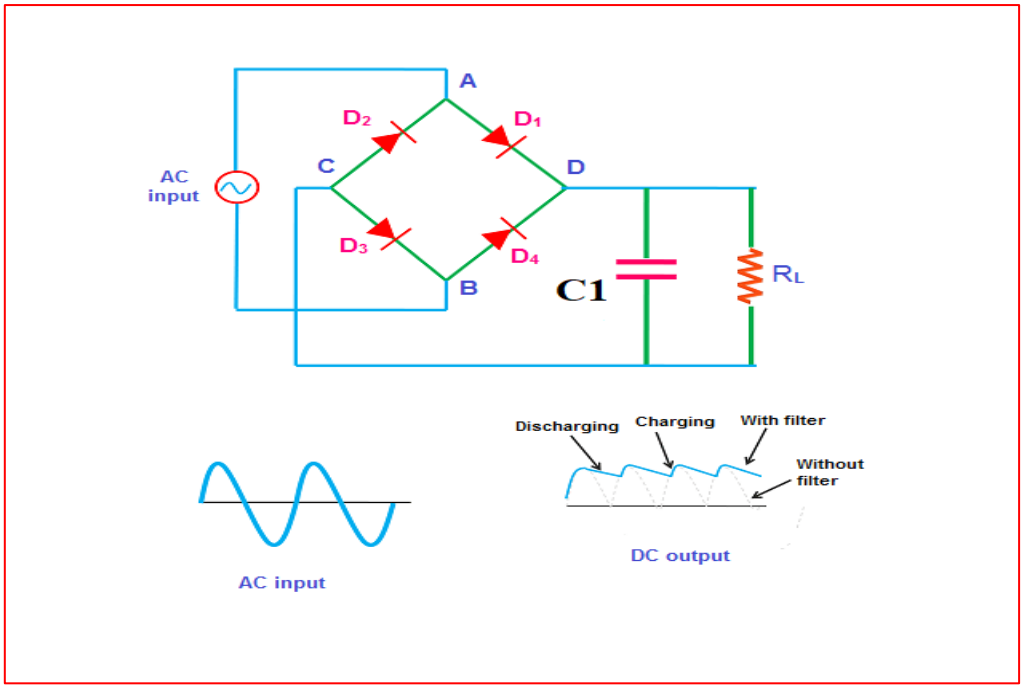



**Figure.4.8.Centered Tapped Full-Wave Rectifier with a Capacitor Filter**

### 4.1.5. Voltage Regulator:

A **voltage regulator** is an electrical regulator designed to automatically maintain a constant voltage level. It may use an electromechanical mechanism, or passive or active electronic components. Depending on the design, it may be used to regulate one or more AC or DC voltages. There are two types of regulator are they.

- Positive Voltage Series (78xx) and
- Negative Voltage Series (79xx)

**78xx:**'78' indicate the positive series and 'xx' indicates the voltage rating. Suppose 7805 produces the maximum 5V.'05'indicates the regulator output is 5V.

**79xx:**'78' indicate the negative series and 'xx' indicates the voltage rating. Suppose 7905 produces the maximum -5V.'05'indicates the regulator output is -5V.

These regulators consists the three pins there are

**Pin1:** It is used for input pin.

**Pin2:** This is ground pin for regulator

**Pin3:** It is used for output pin. Through this pin we get the output.

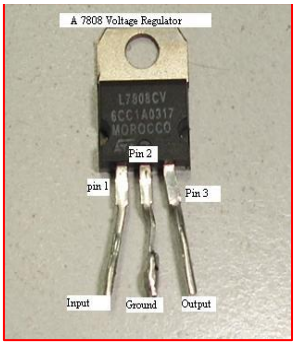



**Figure.4.9.Regulator**

## 4.2. LIQUID CRYSTAL DISPLAY

### 4.2.1. INTRODUCTION TO LCD

A liquid crystal display (LCD) is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. Each pixel consists of a column of liquid crystal molecules suspended between two transparent electrodes, and two polarizing filters, the axes of polarity of which are perpendicular to each other.

Without the liquid crystals between them, light passing through one would be blocked by the other. The liquid crystal twists the polarization of light entering one filter to allow it to pass through the other.

A program must interact with the outside world using input and output devices that communicate directly with a human being. One of the most common devices attached to an controller is an LCD display. Some of the most common LCDs connected to the controllers are 16X1, 16x2 and 20x2 displays. This means 16 characters per line by 1 line 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively.

Many microcontroller devices use 'smart LCD' displays to output visual information. LCD displays designed around LCD NT-C1611 module, are inexpensive, easy to use, and it is even possible to produce a readout using the 5X7 dots plus cursor of the display.

They have a standard ASCII set of characters and mathematical symbols. For an 8-bit data bus, the display requires a +5V supply plus 10 I/O lines (RS,RW,D7, D6,D5,D4,D3,D2,D1,D0).

For a 4-bit data bus it only requires the supply lines plus 6 extra lines(RS,RW,D7,D6,D5,D4). When the LCD display is not enabled, data lines are tri-state and they do not interfere with the operation of the microcontroller.

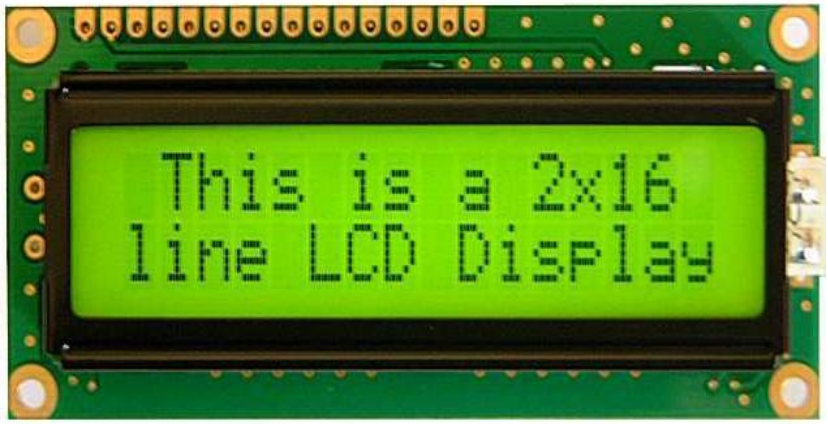



**Figure.4.10. 2x16 LCD Display**

### 4.2.2. USES

The LCD s used exclusively in watches, calculators and measuring instruments is the simple seven-segment displays, having a limited amount of numeric data. The recent advances in technology have resulted in better legibility, more information displaying capability and a wider temperature range. These have resulted in the LCD s being extensively used in telecommunications and entertainment electronics. The LCD s has even started replacing the cathode ray tubes (CRTs) used for the display of text and graphics, and also in small TV applications

### 4.2.3. LCD PIN DIAGRAM

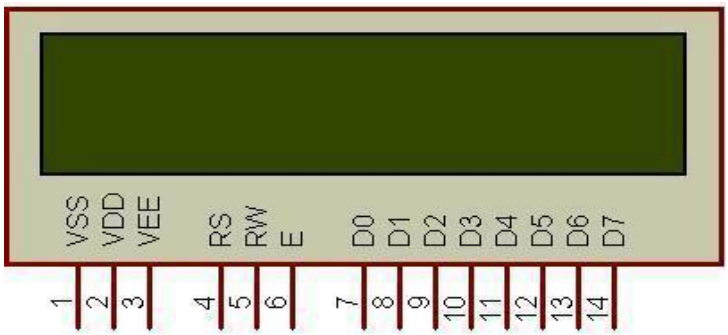



**Figure.4.11.Pin Diagram of LCD**

| PIN | SYMBOL | FUNCTION |
|---|---|---|
| | | |
| 1 | $V_{SS}$ | Power Supply(GND) |
| | | |
| 2 | $V_{DD}$ | Power Supply(+5V) |
| | | |
| 3 | $V_O$ | Contrast Adjust |
| | | |
| 4 | RS | Instruction/data register select |
| | | |
| 5 | R/W | Data Bus Line |
| | | |
| 6 | E | Enable Signal |
| | | |
| 7-14 | DB0-DB7 | Data Bus Line |
| | | |
| 15 | A | Power supply for LED B/L(+) |
| | | |
| 16 | K | Power Supply for LED B/L(-) |
| | | |

**Table.4.1. Functions of control lines**

### 4.2.4. CONTROL LINES

**EN**:

Line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely

ready, bring EN high (1) and wait for the minimum amount of time required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

**RS**:

Line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.).When RS is high (1), the data being sent is text data which should be displayed on the screen.

**RW**:

Line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands, so RW will almost always be low.

- **LOGIC STATUS ON CONTROL LINES**

- E - 0 Access to LCD disabled.
  1 Access to LCD enabled.
- R/W - 0 Writing data to LCD.
  1 Reading data from LCD
- RS - 0 Instructions..
  1 Character

- **CONTRAST CONTROL**

To have a clear view of the characters on the LCD, contrast should be adjusted. To adjust the contrast, the voltage should be varied. For this, a preset is used which can behave like a variable voltage device. As the voltage of this preset is varied, the contrast of the LCD can be adjusted.

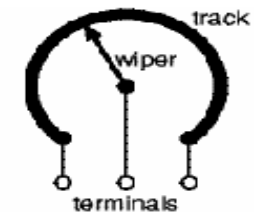



**Figure.4.12. variable resistor**

**Potentiometer**

Variable resistors used as potentiometers have all **three terminals** connected. This arrangement is normally used to **vary voltage**, for example to set the switching point of a circuit with a sensor, or control the volume (loudness) in an amplifier circuit. If the terminals at the ends of the track are connected across the power supply, then the wiper terminal will provide a voltage which can be varied from zero up to the maximum of the supply.

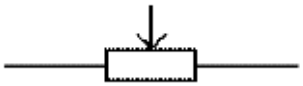


**Figure 4.13.potentiometer symbol**

**Presets**

These are miniature versions of the standard variable resistor. They are designed to be mounted directly onto the circuit board and adjusted only when the circuit is built. For example, to set the frequency of an alarm tone or the sensitivity of a light-sensitive circuit, a small screwdriver or similar tool is required to adjust presets. Presets are much cheaper than standard variable resistors so they are sometimes used in projects where a standard variable resistor would normally be used.

**Multiturn presets** are used where very precise adjustments must be made. The screw must be turned many times (10+) to move the slider from one end of the track to the other, giving very fine control.

Preset Symbol

**Figure.4.14. preset symbol**

### 4.3 ESP8266 Wi-Fi Module

This is the heart of the project. Since the project is based on WIFI control of appliances, the module forms the important component of the project.

The ESP8266 Arduino compatible module is a low-cost Wi-Fi chip with full TCP/IP capability, and the amazing thing is that this little board has a MCU (Micro Controller Unit) integrated which gives the possibility to control I/O digital pins via simple and almost pseudo-code like programming language. This device is produced by Shanghai-based Chinese manufacturer, Es press if Systems.

This chip was first time seen in August 2014, in ESP-01 version module, made by AIThinker, a third-party manufacturer. This little module allows the MCU to connect to WiFi network and create simple TCP/IP connections. His His very low price (1.7$ – 3.5$) and the incredible small size attracted many geeks and hackers to explore it and use it in a large variety of projects. Being a true success, Espressif produces now many versions having different dimensions and technical specifications. One of the successors is ESP32. You can find over the internet hundreds of projects and various implementations like home automation, data logging solutions, robotics, controlling things over the internet, even drones or copters.

4.3.1 ESP8266-01Technical specifications

- 32-bit RISC CPU: Ten silica Xtensa LX106 running at 80 MHz **
- 64 KiB of instruction RAM, 96 KiB of data RAM
- External QSPI flash – 512 KiB to 4 MiB* (up to 16 MiB is supported)
- IEEE 802.11 b/g/n Wi-Fi
- Integrated TR switch, balun, LNA, power amplifier and matching network
- WEP or WPA/WPA2 authentication, or open networks
- 16 GPIO pins **
- SPI, I²C,
- I²S interfaces with DMA (sharing pins with GPIO)
- UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO2
- 1 10-bit ADC

** CPU and flash clock speeds can be raised via over clocking on some devices and the 16 I/O are not available in all versions.

xxiv

ESP8266 Arduino Module comes with PCB trace antenna which seems to have a very good

coverage (I saw a demonstration with more than 1km range!!!). Other version can have onboard ceramic antenna or an external connector which allows you to attach external Wi-Fi antennas modules. ESP-01 has only 6 active pins, although the MCU can support up to 16 I/O. Board dimensions are 14.3 x 24.8 mm.


**Figure 4.7: ESP8266 Wi-Fi Module**

Over the internet i found that ESP8266 Arduino module, version 01, is sold in two or more versions, which at first glance seem quite the same. After buying both of them i saw that there is a difference in size of the flash memory. You may encounter issues while flashing if you don't make the proper settings according to board specifications.

Although the board default has 2 available GPIOs, you can do some workarounds and use other MCU available pins if you have the proper soldering tools. I managed to use GPIO 16 in order to wake up the device after DEEP SLEEP mode (explained later in SLEEP MODES). xxv

### 4.3.2 Module pin description (pin out)

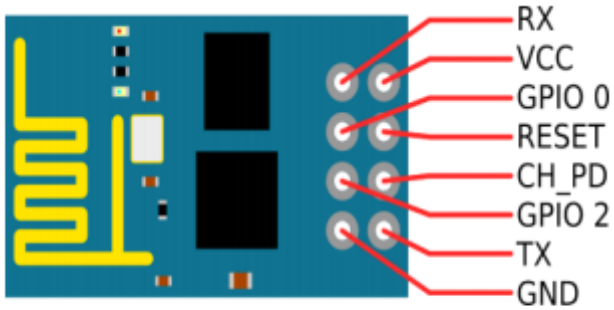



**Figure 4.8: ESP8266 -01 pin diagram**

Pins are arranged in two rows, having 4 on each row. Some models have pin description on the PCB, which make it simple. On the top row you can find following pins from the left to

the right:

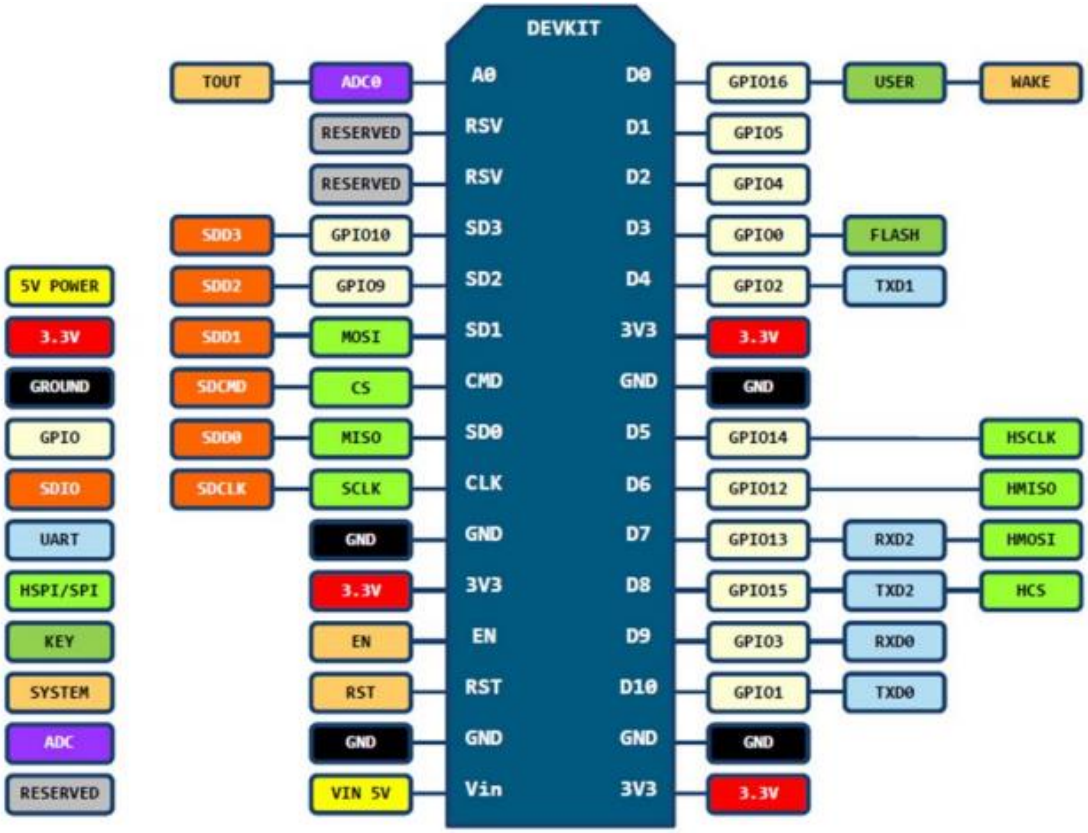
1. GND (Ground from power supply)
2. GPIO2 (Digital I/O programmable)
3. GPIO0 (Digital I/O programmable, also used for BOOT modes)
4. RX – UART Receiving channel

On the bottom (second row) you can find:

1. TX – UART Transmitting channel
2. CH_PD (enable/power down, must be pulled to 3.3v directly or via resistor)
3. REST – reset, must be pulled to 3.3v)
4. VCC -3.3v power supply

Power supply and current consumption

All esp8266 Arduino compatible modules must be powered with DC current from any kind of source that can deliver stable 3.3V and at least 250mA. Also logic signal is rated at 3.3v and the RX channel should be protected by a 3.3v divisor step-down. You should be careful when using this module with Arduino or other boards which supplies 5v, because this module usually do not come with overpower protection and can be easily destroyed.




The most basic way to use the ESP8266 module is to use serial commands, as the chip is basically a WiFi/Serial

transceiver. However, this is not convenient. What we recommend is using the very cool Arduino ESP8266 project,

which is a modified version of the Arduino IDE that you need to install on your computer. This makes it very

convenient to use the ESP8266 chip as we will be using the well-known Arduino IDE. Following the below step to

install ESP8266 library to work in Arduino IDE environment.

3.1 Install the Arduino IDE 1.6.4 or greater

Download Arduino IDE from Arduino.cc (1.6.4 or greater) - don't use 1.6.2 or lower version! You can use your

existing IDE if you have already installed it.

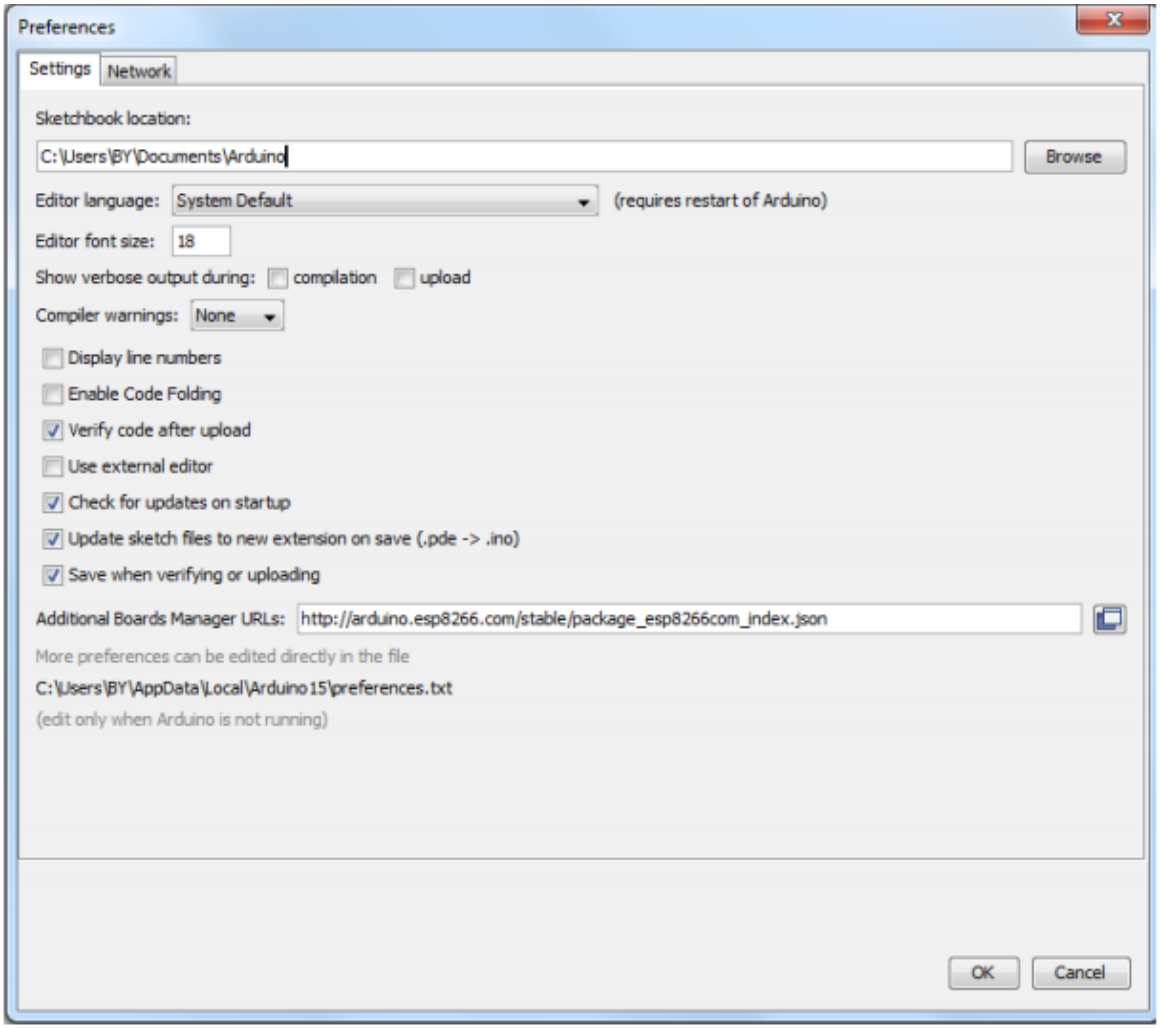
You can also try downloading the ready-to-go package from the ESP8266-Arduino project, if the proxy is giving you

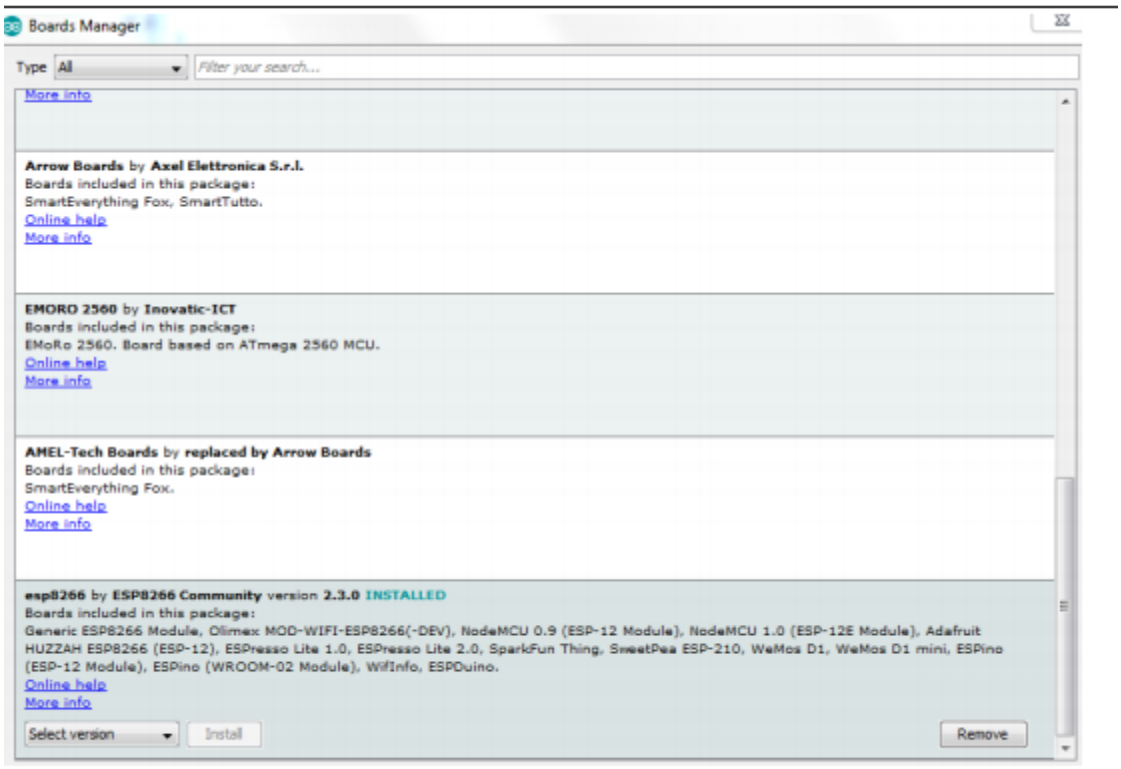
problems.

3.2 Install the ESP8266 Board Package

Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into Additional Board Manager URLs

field in the Arduino v1.6.4+ preferences.





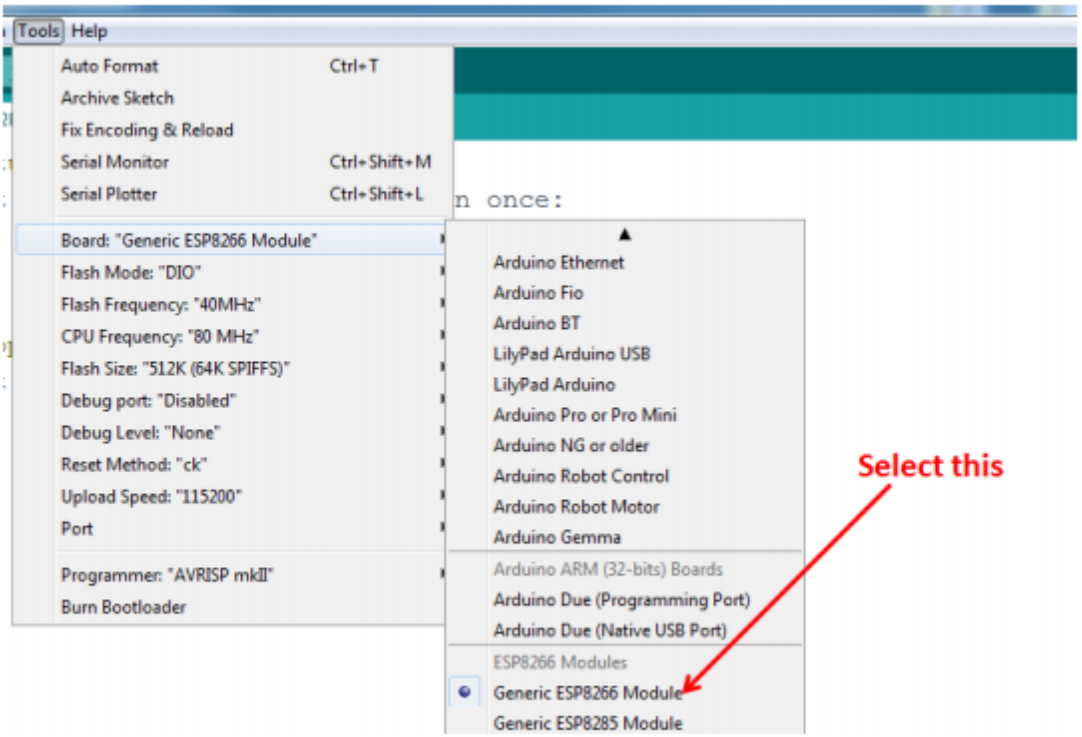

Click 'Tools' -> 'Board:' -> 'Board Manager...' to access this panel.

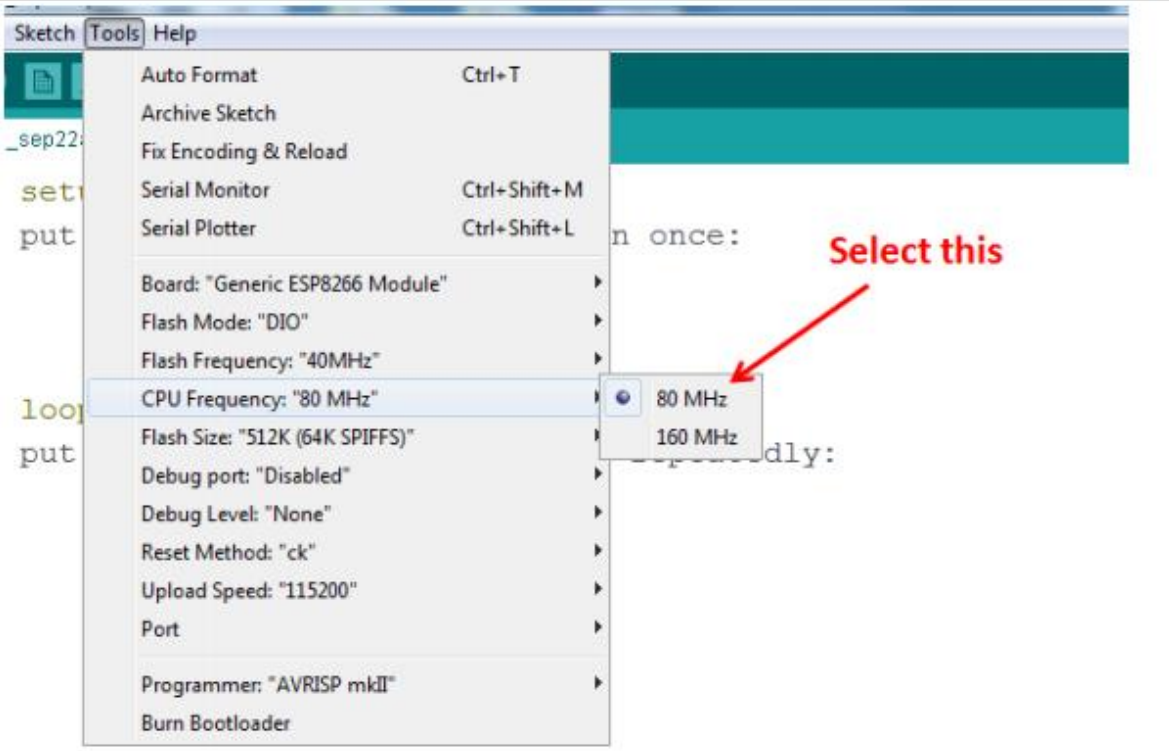
Scroll down to ' esp8266 by ESP8266 Community ' and click "Install" button to install the ESP8266 library package.

Once installation completed, close and re-open Arduino IDE for ESP8266 library to take effect.

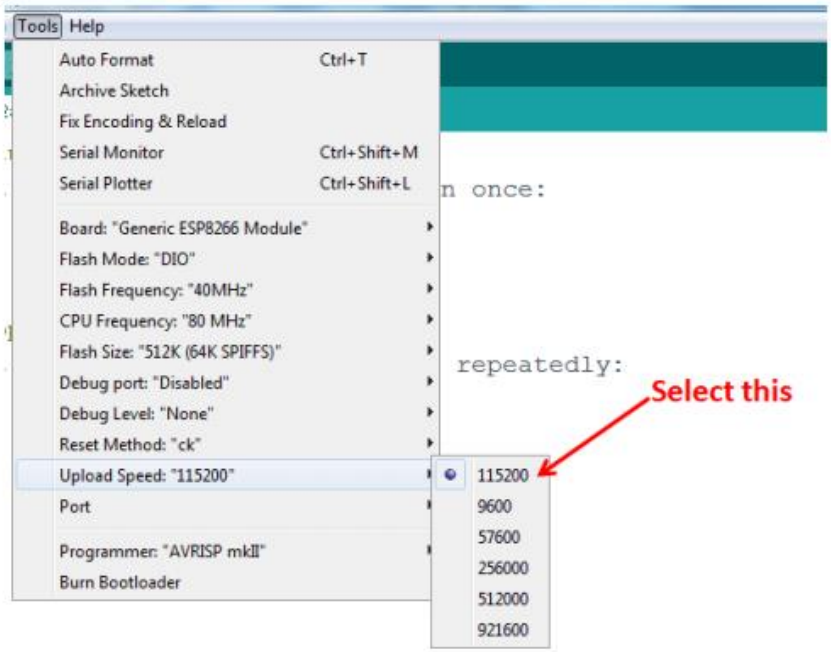
3.3 Setup ESP8266 Support

When you've restarted Arduino IDE, select 'Generic ESP8266 Module' from the 'Tools' -> 'Board:' dropdown menu.

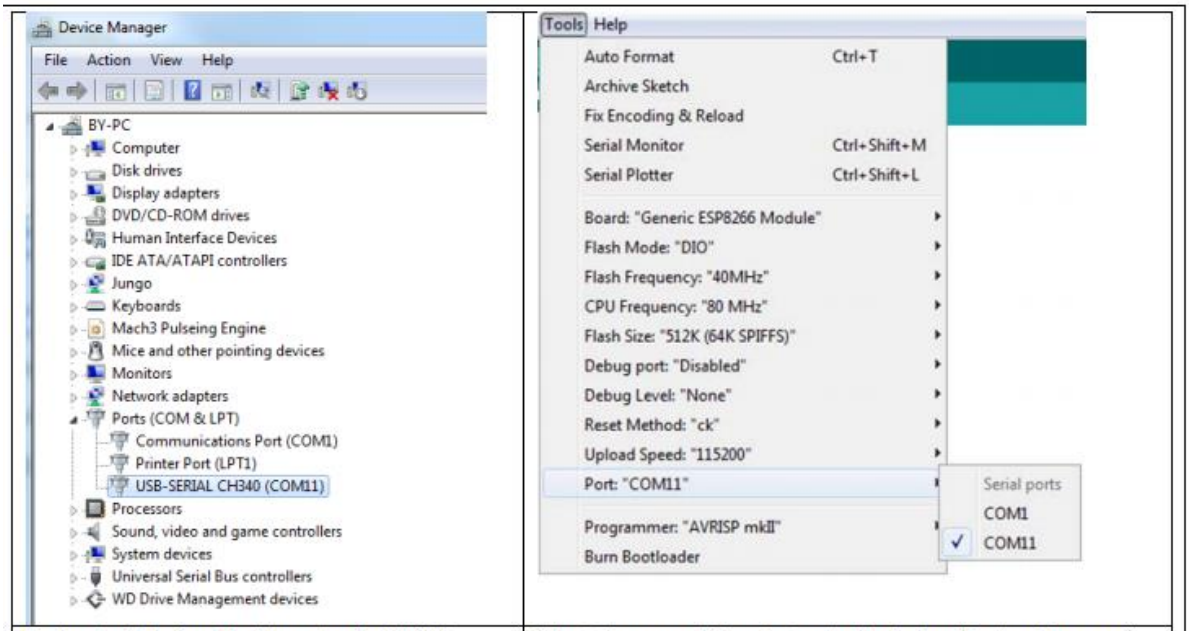

Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port
Programmer: "AVRISP mkII"
Burn Bootloader
Arduino Ethernet
Arduino Fio
Arduino BT
LilyPad Arduino USB
LilyPad Arduino
Arduino Pro or Pro Mini
Arduino NG or older
Arduino Robot Control
Arduino Robot Motor
Arduino Gemma
Arduino ARM (32-bits) Boards
Arduino Due (Programming Port)
Arduino Due (Native USB Port)
ESP8266 Modules
Generic ESP8266 Module
Generic ESP8285 Module
Select this


Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port
Programmer: "AVRISP mkII"
Burn Bootloader
80 MHz
160 MHz
Select this

Select '115200' baud upload speed is a good place to start - later on you can try higher speeds but 115200 is a good safe place to start.




Go to your Windows 'Device Manager' to find out which Com Port 'USB-Serial CH340' is assigned to. Select the matching COM/serial port for your CH340 USB-Serial interface.




| Find out which Com Port is assign for CH340 | Select the correct Com Port as indicated on 'Device Manager" |
|---|---|

***Note: if this is your first time using CH340 " USB-to-Serial " interface, please install the driver first before proceed the above Com Port setting. The CH340 driver can be download from the below site:***

***https://github.com/nodemcu/nodemcu-devkit/tree/master/Drivers***

3.4 Blink Test

https://github.com/nodemcu/nodemcu-devkit/tree/master/Drivers

We'll begin with the simple blink test.

Enter this into the sketch window (and save since you'll have to). Connect a LED as shown in Figure3-1.

```
void setup() {
 pinMode(5, OUTPUT); // GPIO05, Digital Pin D1
}
void loop() {
 digitalWrite(5, HIGH);
 delay(900);
 digitalWrite(5, LOW);
 delay(500);
}
```

Now you'll need to put the board into bootload mode. You'll have to do this before each upload. There is no timeout

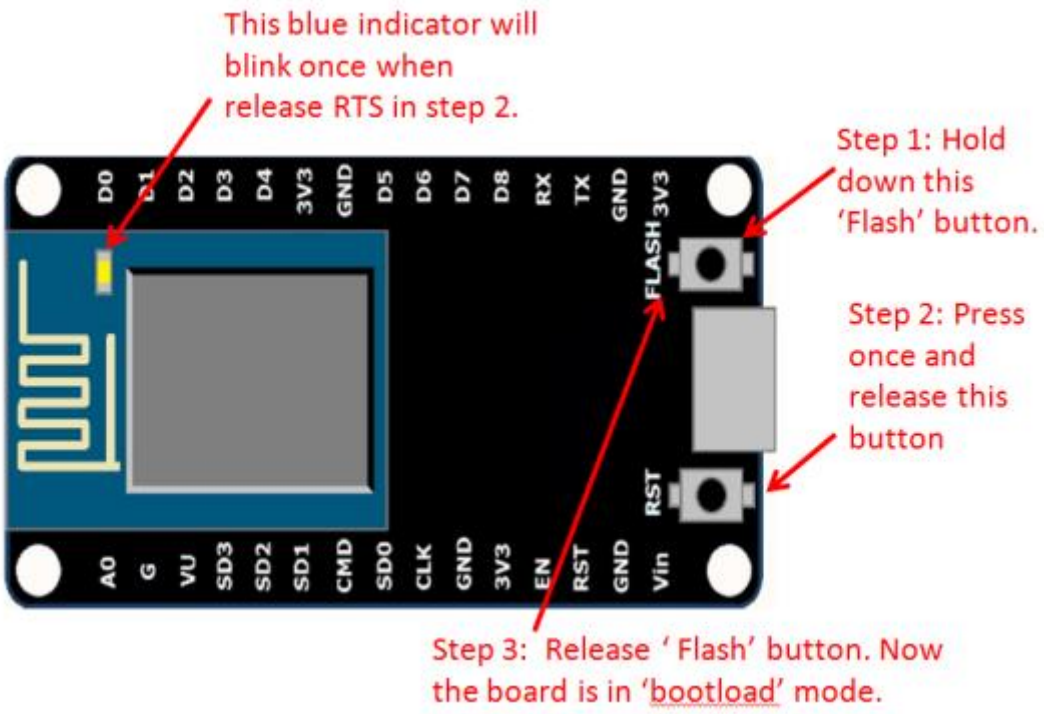
for bootload mode, so you don't have to rush!

• Hold down the ‘Flash’ button.

• While holding down ‘ Flash’, press the ‘RST’ button.

• Release ‘RST’, then release ‘Flash’

When you release the ‘RST’ button, the blue indication will blink once, this means its ready to bootload.




Once the ESP board is in bootload mode, upload the sketch via the IDE, Figure 3-2.

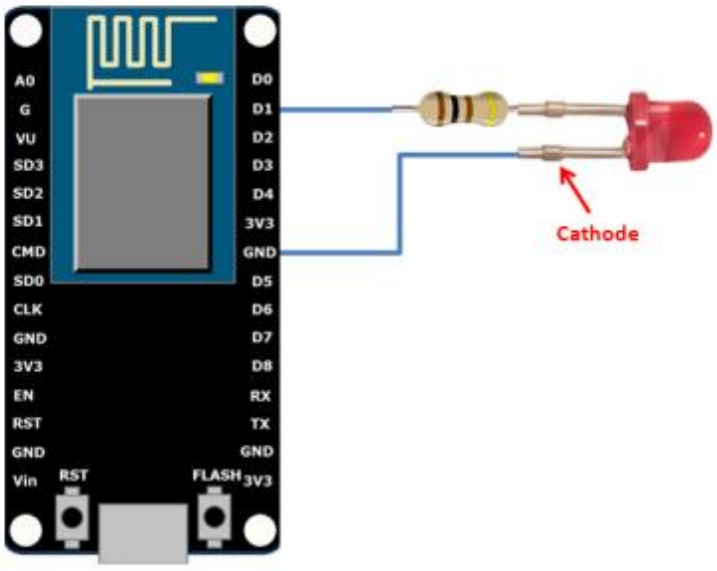
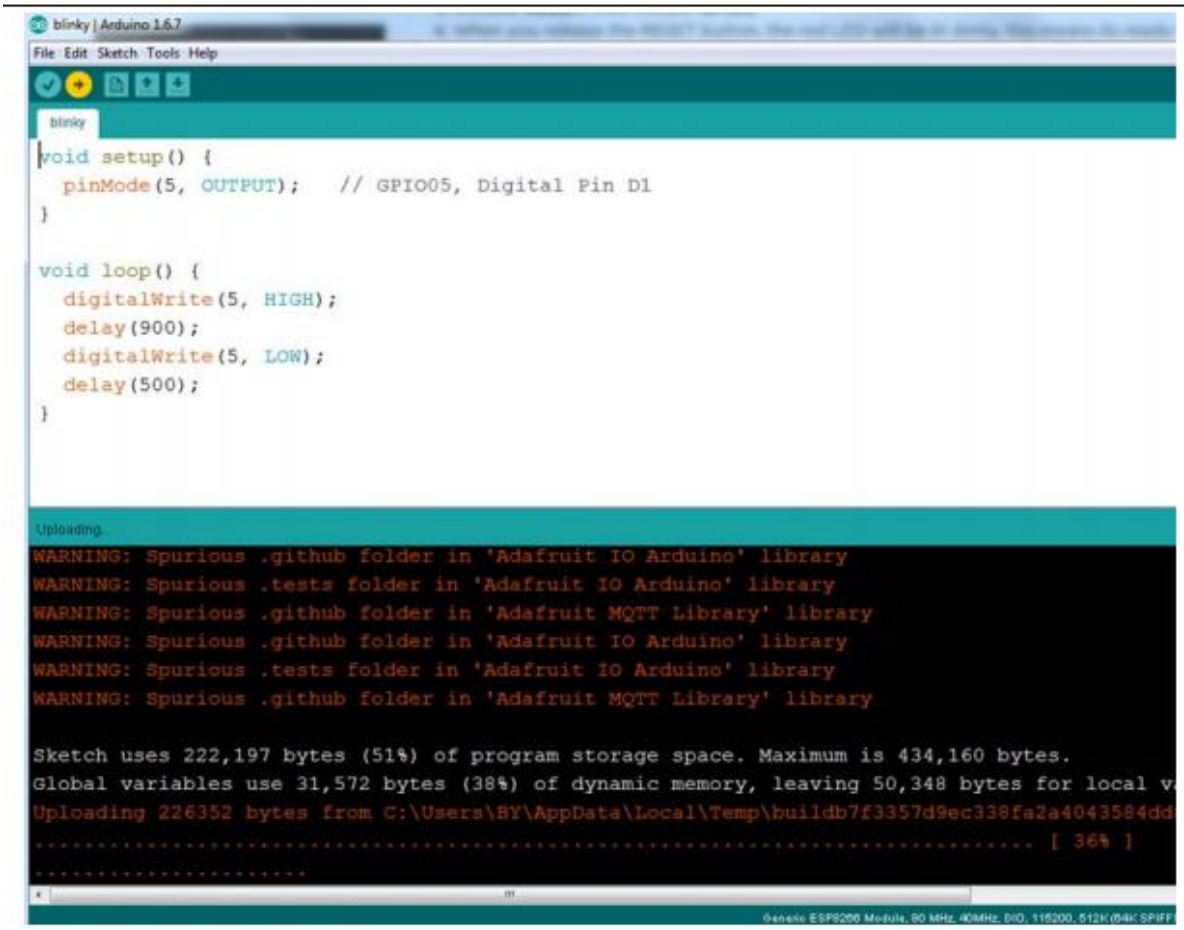



Figure3-1: Connection diagram for the blinking test




The sketch will start immediately - you'll see the LED blinking. Hooray!

3.5 Connecting via WiFi

OK once you've got the LED blinking, let’s go straight to the fun part, connecting to a webserver. Create a new sketch

with this code:

Don’t forget to update:

const char* ssid = "yourssid";

const char* password = "yourpassword";

to your WiFi access point and password, then upload the same way: get into bootload mode, then upload code via

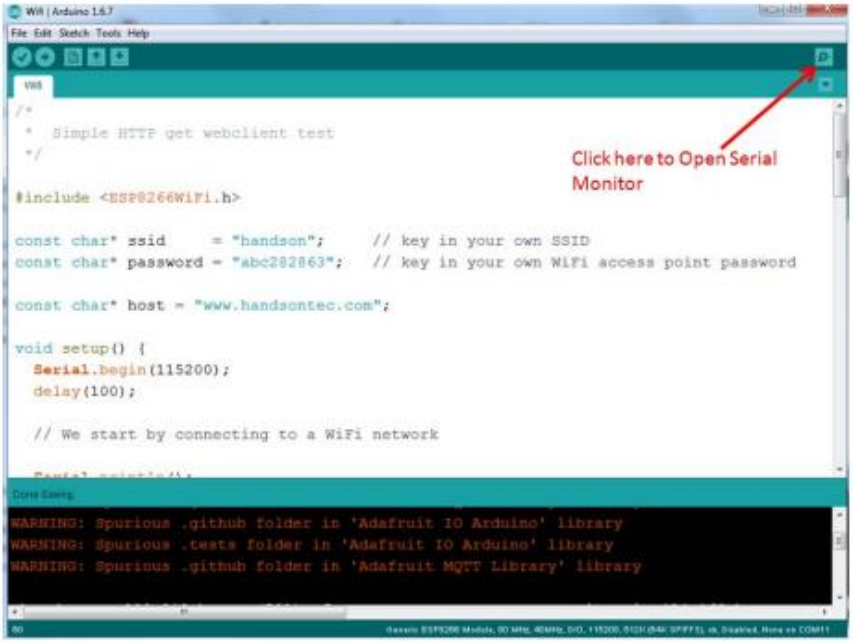
IDE.

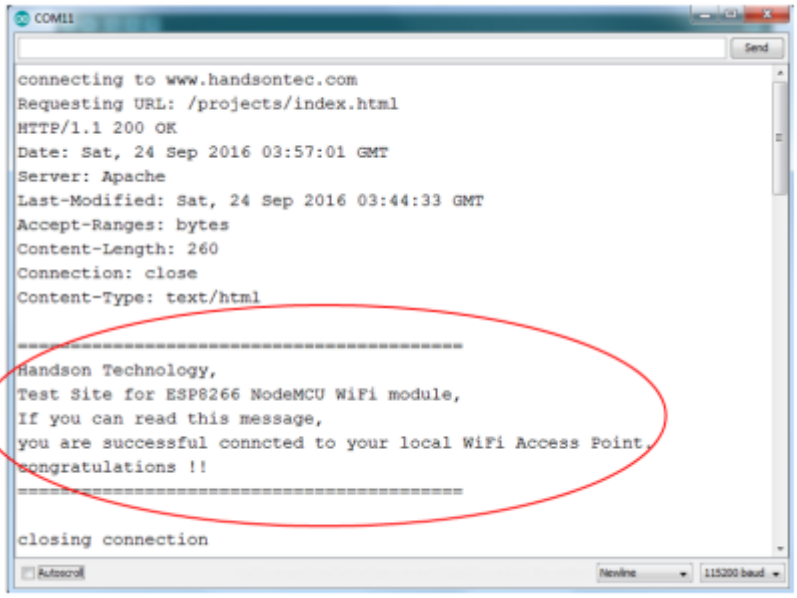
```
/*
 *  Simple HTTP get webclient test
 */

#include <ESP8266WiFi.h>

const char* ssid     = "handson";      // key in your own SSID
const char* password = "abc1234";      // key in your own WiFi access point
password
```

Open up the IDE serial console at 115200 baud to see the connection and webpage printout!



4. Flashing NodeMCU Firmware on the ESP8266 using Windows

Why flashing your ESP8266 module with NodeMCU?

NodeMCU is a firmware that allows you to program the ESP8266 modules with LUA script. And you'll find it very

similar to the way you program your Arduino. With just a few lines of code you can establish a WiFi connection,

control the ESP8266 GPIOs, turning your ESP8266 into a web server and a lot more.

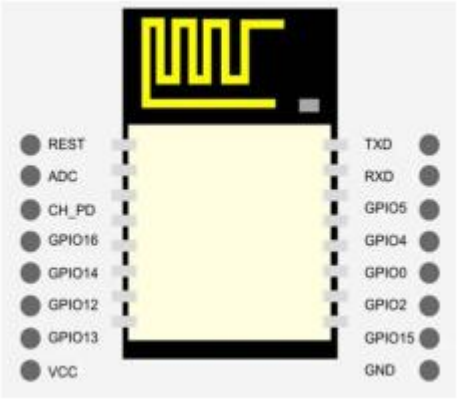
In this tutorial we are going to use another ESP8266 module with pin header adapter board which is breadboard

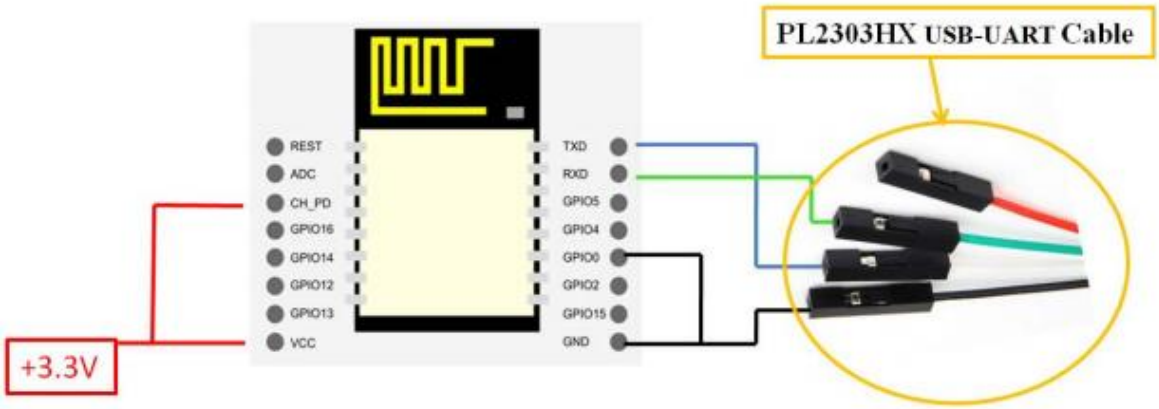
friendly.

ESP8266 Module Breadboard Friendly with Header Connector

4.1 Parts Required:

• ESP8266 Module Breadboard Friendly




| ESP8266 Pin | Description |
|---|---|
| CH_PD | Pull high, connect to Vcc +3.3V |
| Vcc | Power Supply +3.3V |
| TXD | Connect to RXD (white) of PL2303HX USB-Serial converter cable |
| RXD | Connect to TXD (Green) of PL2303HX USB-Serial converter cable |
| GPIO0 | Pull low, connect to GND pin |
| GND | Power Supply ground |

4.4 Downloading NodeMCU Flasher for Windows

After wiring your circuit, you have to download the NodeMCU flasher. This is a .exe file that you can download using

one of the following links:

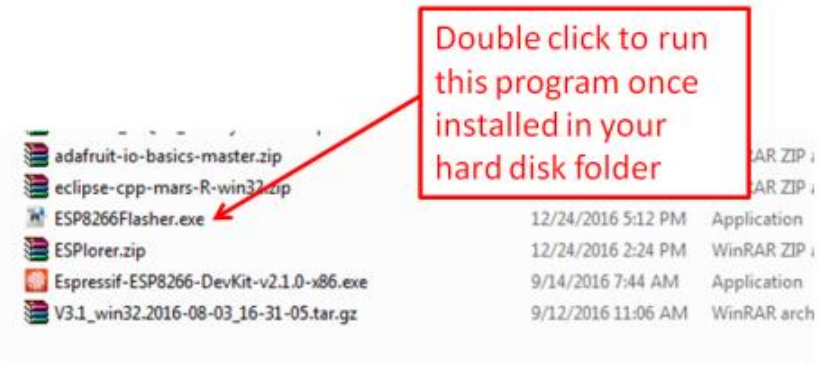
• Win32 Windows Flasher

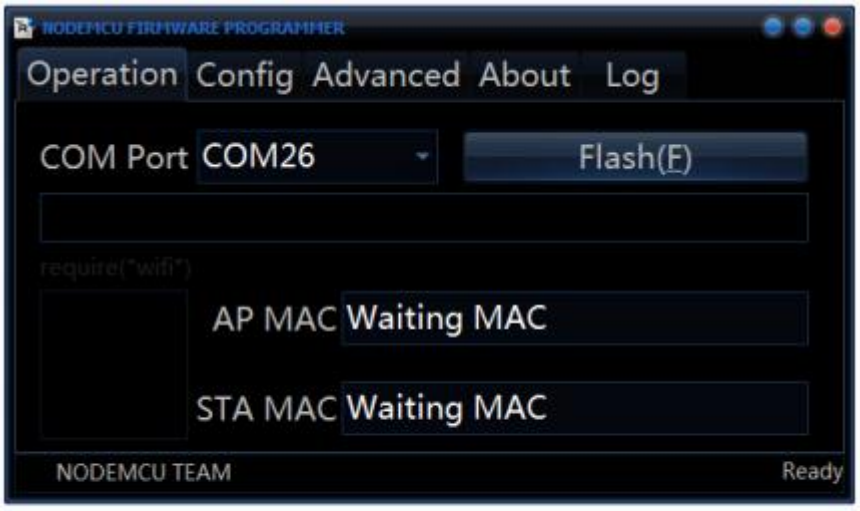
• Win64 Windows Flasher

You can find all the information about NodeMCU flasher here.

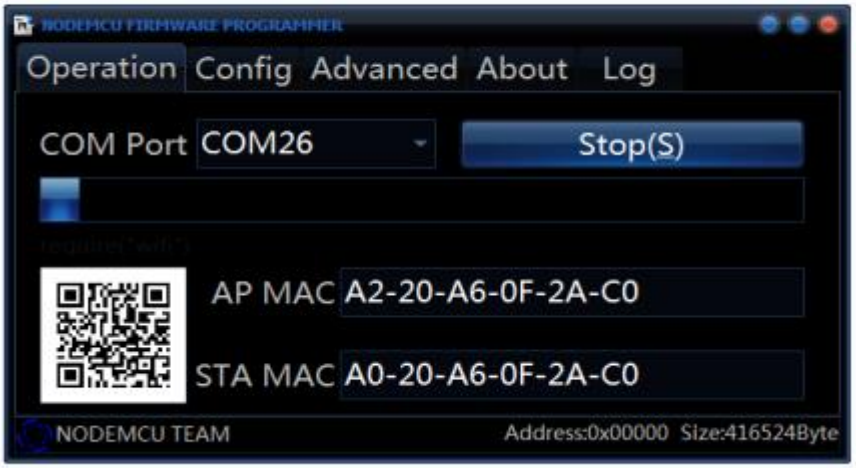
## 4.5 Flashing your ESP8266 using Windows

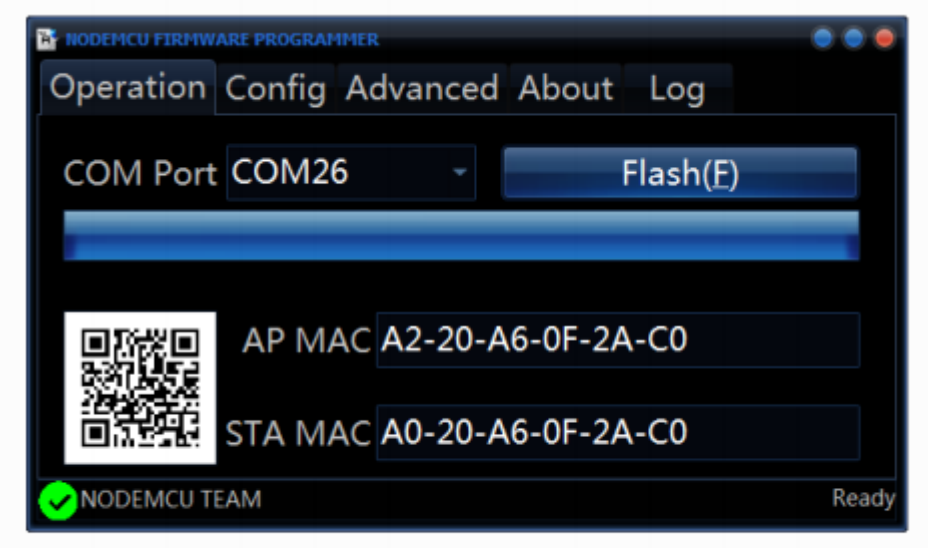
Open the flasher that you just downloaded and a window should appear (as shown in the following figure).




"Flash" and it should start the flashing process immediately, showing the Modu
cted.






Your ESP8266 module is now loaded with NodeMCU firmware.

Getting Started with the ESPlorer IDE

ESPlorer is an IDE (Integrated Development Environment) for ESP8266 devices. It's a multi platform IDE, can be

used in any OS environment, this simply means that it runs on Windows, Mac OS X or Linux.

Supported platforms:

• Windows(x86, x86-64)

• Linux(x86, x86-64, ARM soft & hard float)

• Solaris(x86, x86-64)

• Mac OS X(x86, x86-64, PPC, PPC64)

This software allows you to establish a serial communications with your ESP8266 module, send commands, and

upload code and much more.

Requirements:

• You need to have JAVA installed in your computer. If you don't have, go to this

website: http://java.com/download, download and install the latest version. It requires JAVA (SE version 7

and above) installed.

• In order to complete the sample project presented in this Guide you need to flash your ESP8266 with

NodeMCU firmware. Refer to chapter-4 in this guide on how to flash the NodeMCU firmware.

Main Resources:

• ESPlorer Homepage: http://esp8266.ru/esplorer/

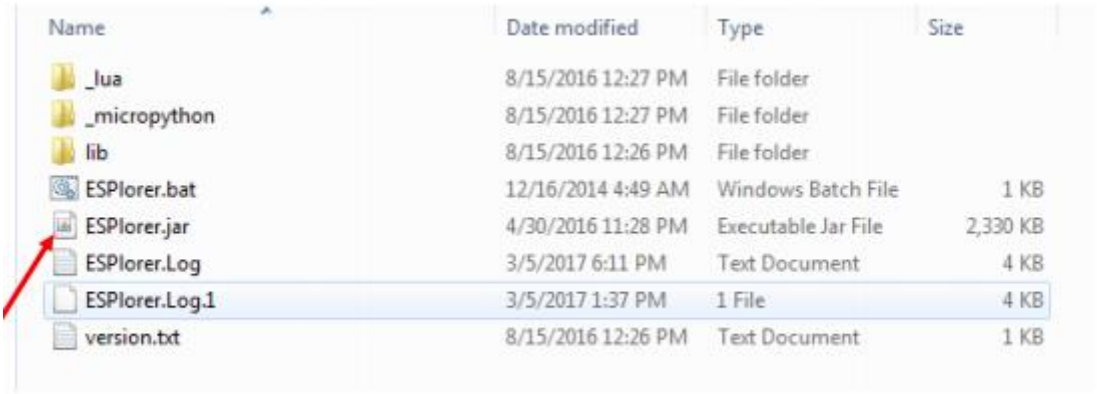
• GitHub Repository: https://github.com/4refr0nt/ESPlorer

5.1 Installing ESPlorer

Now let's download the ESPlorer IDE, visit the following URL: http://esp8266.ru/esplorer/#download

Grab the folder that you just downloaded. It should be named "ESPlorer.zip" and unzip it. Inside that folder you

should see the following files:




Execute the "ESPlorer.jar" file and the ESPlorer IDE should open after a few seconds (the "ESPlorer.jar" file is what

you need to open every time you want to work with the ESPlorer IDE).

Note: If you're on Mac OS X or Linux you simply use this command line in your terminal to run the ESPlorer: sudo

java –jar ESPlorer.jar.

When the ESPlorer first opens, that's what you should see:


Here's a rundown of the features the ESPlorer IDE includes:

• Syntax highlighting LUA and Python code.

• Code editor color themes: default, dark, Eclipse, IDEA, Visual Studio.

• Undo/Redo editors features.

• Code Autocomplete (Ctrl+Space).

• Smart send data to ESP8266 (without dumb send with fixed line delay), check correct answer from ESP8266

after every lines.

• Code snippets.

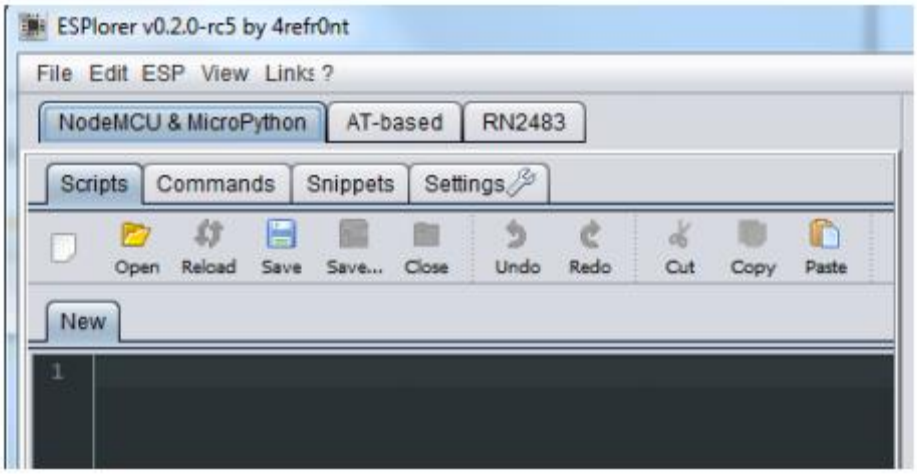
• Detailed logging.

• And a lot more…

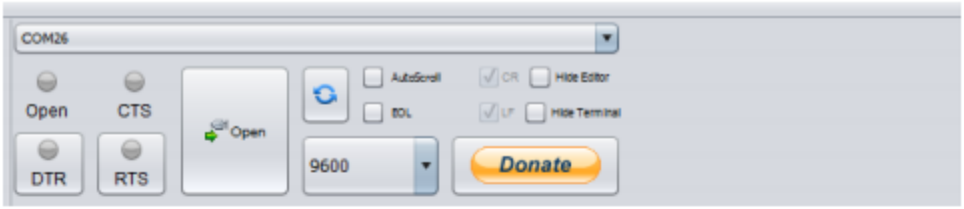
The ESPlorer IDE has a couple of main sections, let's break it down each one.

In the top left corner you can see all the regular options that you find in any software. Create a New file, Open a new

file, Save file, Save file as, Undo, Redo, etc.

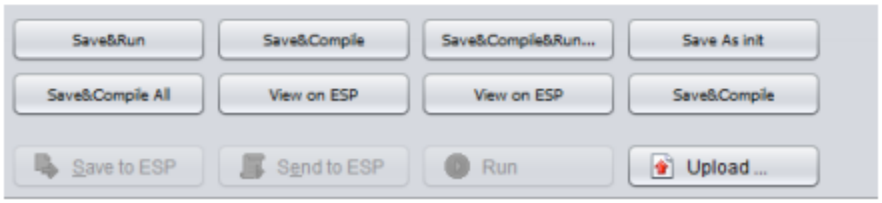



In the top right corner you have all the options you need to establish a serial communication (you're going to learn
how to use them later in this Guide).

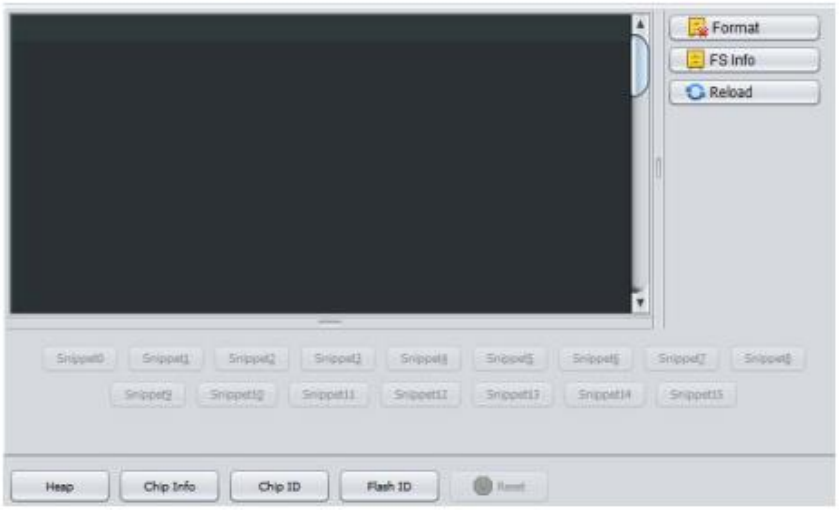



reenshot shows your Code Window, that's where you write your scripts (your scripts are hig
ntax).


ode Window, you have 12 buttons that offer you all the functions you could possible need t
66. Here's the ones you'll use most: "Save to ESP" and "Send to ESP".



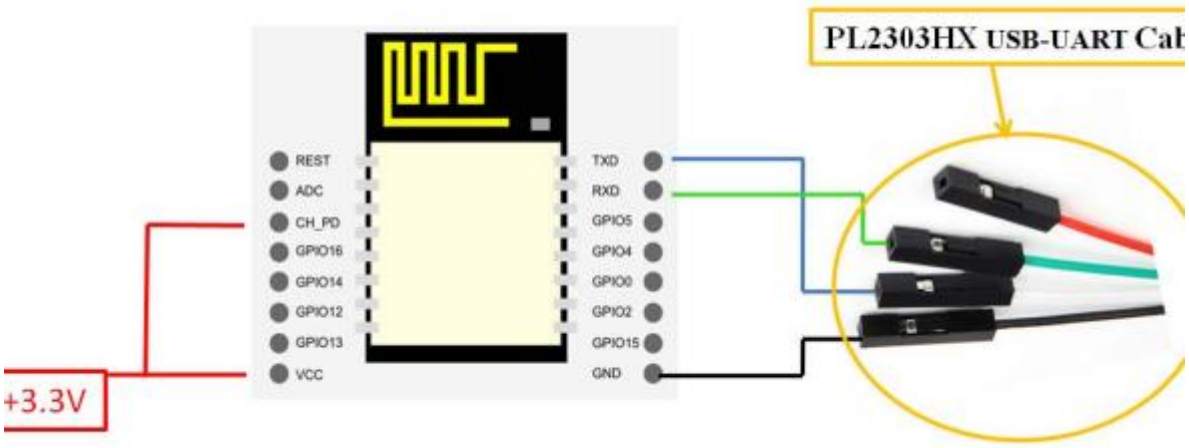

This screenshot shows the Output Window which tells you exactly what's going on in your ESP8266. You c
errors and use prints in your code to debug your projects.



5.2 Schematics

To upload code to your ESP8266, you should connect your ESP8266 to your PL2303HX USB-UART Programming Cable

like the figure below:




```
lighton=0
pin=4
gpio.mode(pin,gpio.OUTPUT)
tmr.alarm(1,2000,1,function()
    if lighton==0 then
        lighton=1
        gpio.write(pin,gpio.HIGH)
    else
        lighton=0
         gpio.write(pin,gpio.LOW)
    end
end)
```

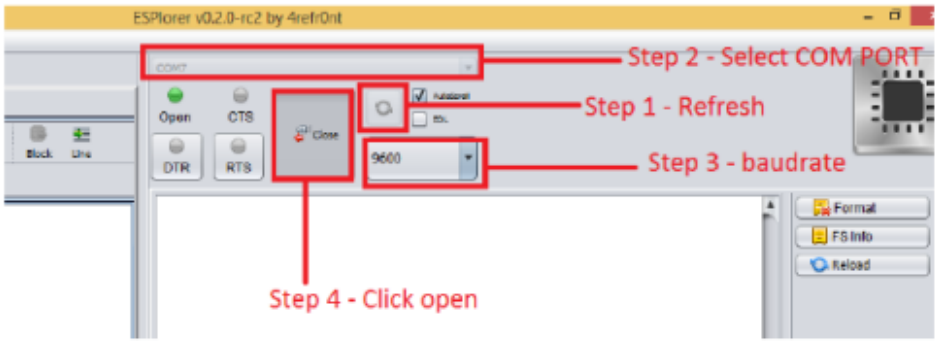
init.lua

```
lighton=0
pin=4
gpio.mode(pin,gpio.OUTPUT)
tmr.alarm(1,1000,1,function()
    if lighton==0 then
        lighton=1
        gpio.write(pin,gpio.HIGH)
    else
        lighton=0
         gpio.write(pin,gpio.LOW)
    end
end)
```

Having your ESP8266+PL2303HX Programmer connected to your computer, go to the ESPlorer IDE:

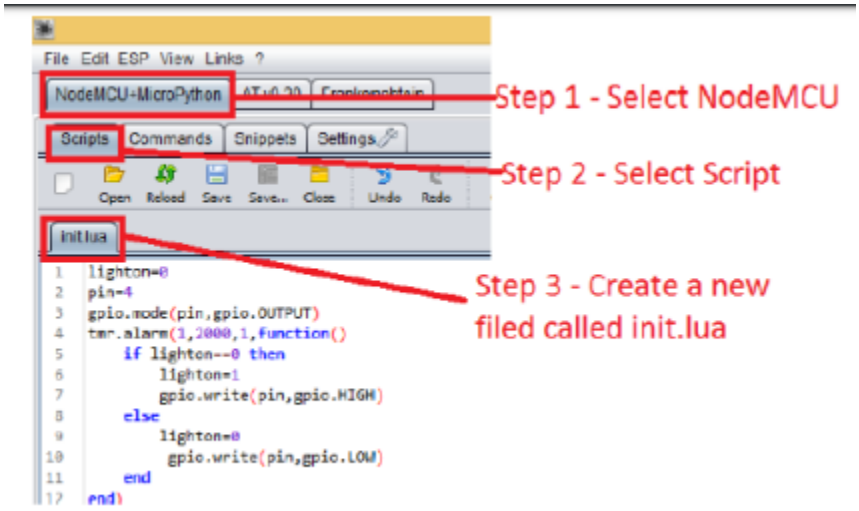

Look at the top right corner of your ESPlorer IDE and follow these instructions:

1. Press the Refresh button.
2. Select the COM port for your FTDI programmer.
3. Select your baudrate.
4. Click Open.

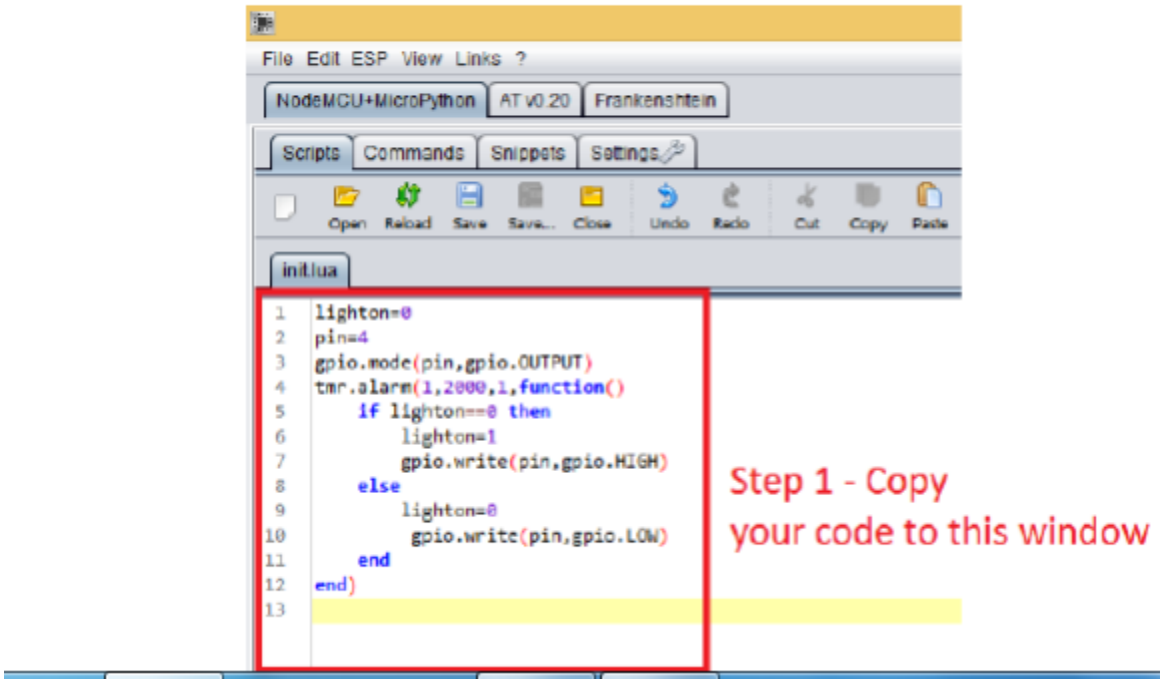



Then in the top left corner of your ESPlorer IDE, follow these instructions:

1. Select NodeMCU
2. Select Scripts
3. Create a new filled called “init.lua”




Copy your Lua script to the code window (as you can see in the Figure below):

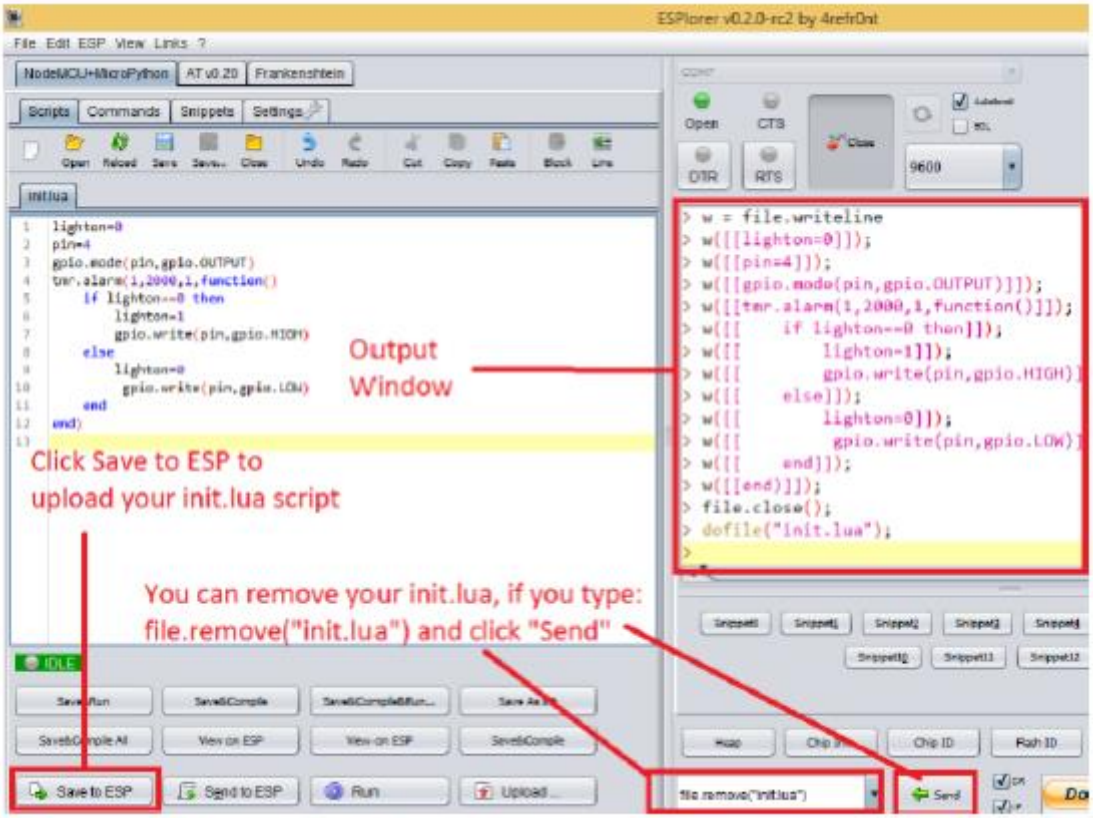



The next step is to save your code to your ESP8266!

At the left bottom corner click the button “Save to ESP”.

In your output window, it should start showing exactly which commands are being sent to your ESP8266 and it should

look similar to the Figure below.

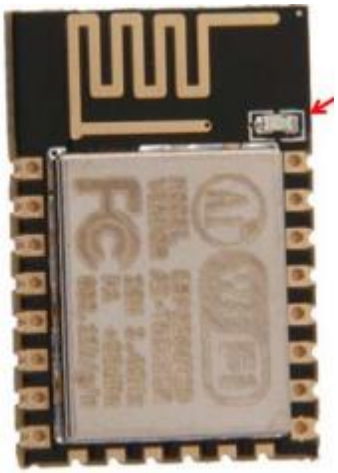



Note: If you want to delete your “init.lua” file, you can do that easily. Simply type file.remove(“init.lua”) and press

the button “Send” (see Figure above). Or you can type the command file.format() to remove all the files saved in your

ESP8266. You can type any commands and send them to your ESP8266 through that window. After uploading your code to your ESP8266, unplug your ESP8266 from your computer and power up the ESP8288

module.



6. NodeMCU GPIO for Lua

The GPIO(General Purpose Input/Output) allows us to access to pins of ESP8266 , all the pins of ESP8266 accessed
using the command GPIO, all the access is based on the I/O index number on the NoddMCU dev kits, not the internal
GPIO pin, for example, the pin 'D7' on the NodeMCU dev kit is mapped to the internal GPIO pin 13, if you want to
turn 'High' or 'Low' that particular pin you need to called the pin number '7', not the internal GPIO of the pin. When
you are programming with generic ESP8266 this confusion will arise which pin needs to be called during
programming, if you are using NodeMCU devkit, it has come prepared for working with Lua interpreter which can
easily program by looking the pin names associated on the Lua board. If you are using generic ESP8266 device or any
other vendor boards please refer to the table below to know which IO index is associated to the internal GPIO of
ESP8266.

| Nodemcu dev kit | ESP8266 Pin | Nodemcu dev kit | ESP8266 Pin |
|---|---|---|---|
| D0 | GPIO16 | D7 | GPIO13 |
| D1 | GPIO5 | D8 | GPIO15 |
| D2 | GPIO4 | D9 | GPIO3 |
| D3 | GPIO0 | D10 | GPIO1 |
| D4 | GPIO2 | D11 | GPIO9 |
| D5 | GPIO14 | D12 | GPIO10 |
| D6 | GPIO12 | | |

D0 or GPIO16 can be used only as a read and write pin, no other options like PWM/I2C are supported by
this pin.
In our example in chapter 5 on blinking the blue LED, the blue LED in connected to GPIO2, it is defined as
Pin4 (D4) in Lua script.

7. Web Resources:

• ESP8266 Lua Nodemcu WIFI Module

• ESP8266 Breadboard Friendly Module

• ESP8266 Remote Serial WIFI Module

• PL2303HX USB-UART Converter Cable

NodeMCU is an open source IoT platform. It includes firmware which runs or the ESP8266 from Espressif, and hardware which is based on the ESP12 module.The term "NodeMCU" by default refers to the firmware rather than the dev kits. The firmware uses the Lua scripting language. It is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson,and spiffs.

Figure 2: NODE MCU Development board

NodeMCU was created shortly after the ESP8266 came out. On December 30, 2013, Espressiv system began production of the ESP8266. The ESP8266 is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core,widely used in IoT applications

The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability produced by Shanghai-based Chinese manufacturer, Espressif Systems.

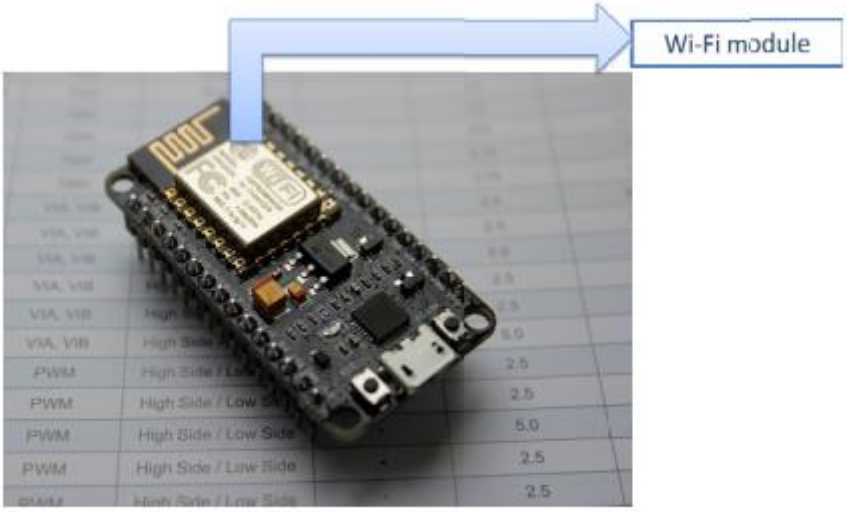



Figure 3: NODE MCU with inbuilt wifi module

This mod based on the D0 pi Please ref NODE M U is an 266 from E CU" by de cripting lan 266. It uses U was crea egan produc X106 core,w 8266 is a lo d by Shangh Pin confi dule provide the I/O ind n on the dev fer to the be MCU open so Espressif, a efault refer nguage. It is s many open ated shortly ction of the widely used ow-cost Wi hai-based Ch Figure 3 guration es access to dex number v kit is map elow GPIO Figure ource IoT pl and hardwa rs to the f s based on t n source pro y after the e ESP8266. d in IoT app -Fi microch hinese manu 3: NODE M of NODE o the GPIO on the Nod pped to the i pin maps fo e 2: NODE latform. I are which firmware ra the eLua pr ojects, such ESP8266 c The ESP82 plications hip with fu ufacturer, E MCU with in E MCU de (General P deMCU dev internal GP or the index MCU Deve It include is based ather than roject, and b as lua-cjson came out. O 266 is a Wi ull TCP/IP Espressif Sy nbuilt wifi m evelopme Purpose Inpu v kits, not t IO pin 16. x↔gpio map elopment bo es firmwa on the ES the dev k built on the n,and spiffs On Decemb i-Fi SoC int stack and m stems. module nt board ut/Output) the internal pping.

Table 1: Node MCU index↔gpio mapping

| IO index | ESP826 6 pin | IO index | ESP8266 pin |
|---|---|---|---|
| 0 [*] | GPIO16 | 7 | GPIO13 |
| 1 | GPIO5 | 8 | GPIO15 |
| 2 | GPIO4 | 9 | GPIO3 |
| 3 | GPIO0 | 10 | GPIO1 |
| 4 | GPIO2 | 11 | GPIO9 |
| 5 | GPIO14 | 12 | GPIO10 |
| 6 | GPIO12 | | |

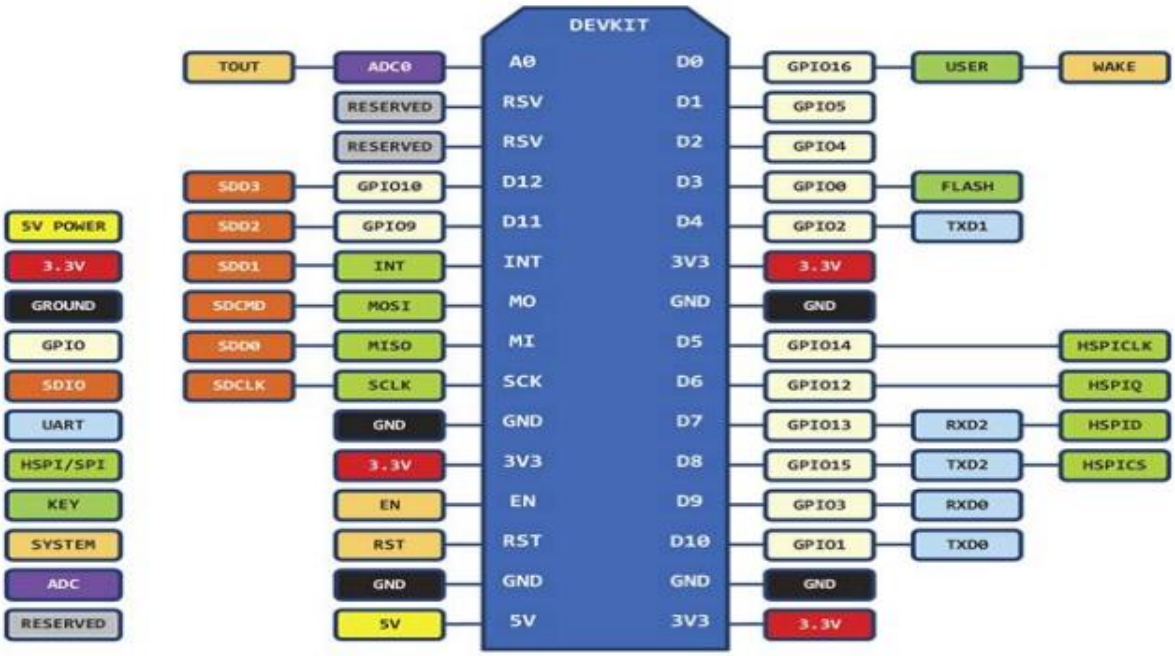



Figure 4: NODE MCU pin configuration

3.2.2 Installation of Node MCU & Coding

Mostly these days devices download and install drivers on their own, automatically. Windows doesn't know how to talk to the USB driver on the Node MCU so it can't figure out that the board is a Node MCU and proceed normally.


Figure 5: Snapshot of the installation process of NODE MCU

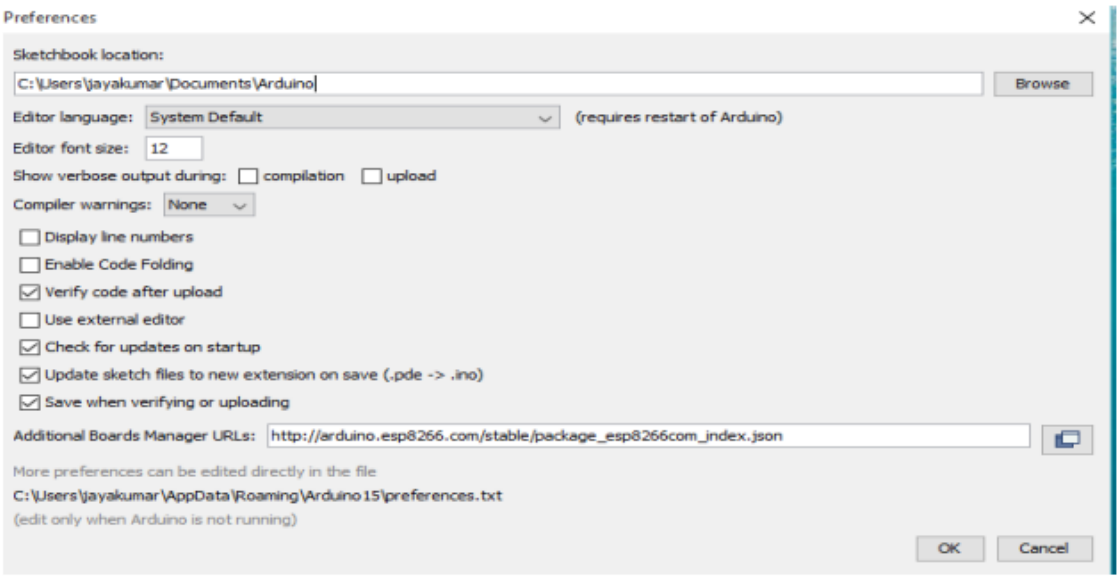



Figure 7: Arduino IDE preferences

copy the below code in the Additional boards Manager
http://arduino.esp8266.com/stable/package_esp8266com_index.json
click OK to close the preference Tab

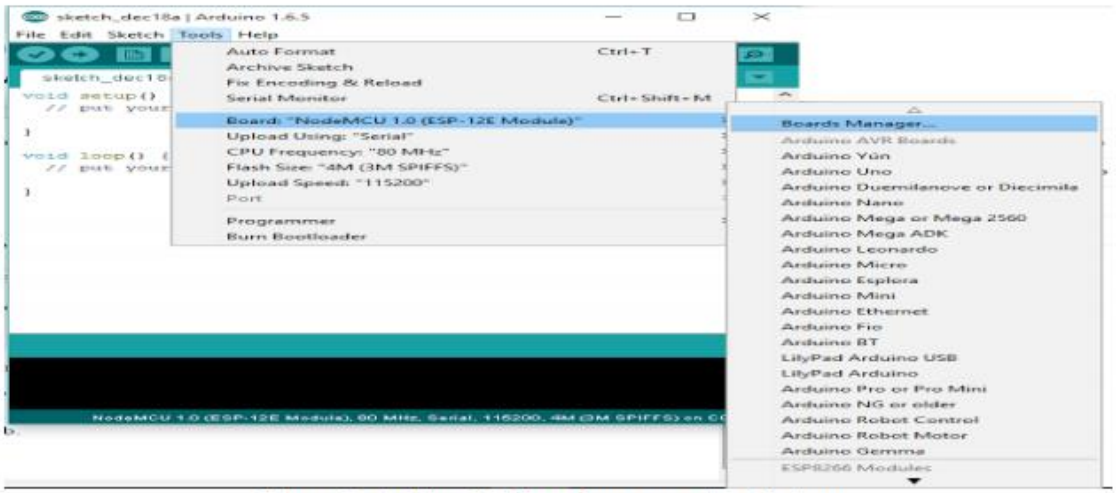



Figure 8: Arduino IDE board manager installation


Figure 9: ESP 8266 board installation in Arduino

Navigate to esp8266 by esp8266 community and install the software for Arduino. Once all the above process had been completed we are ready to program our esp8266 with Arduino IDE.


Figure 10: NODE MCU interfacing with Arduino

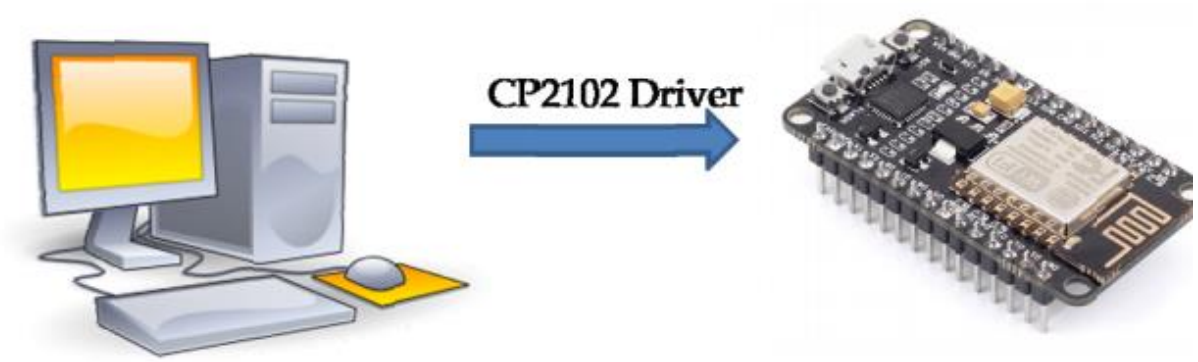
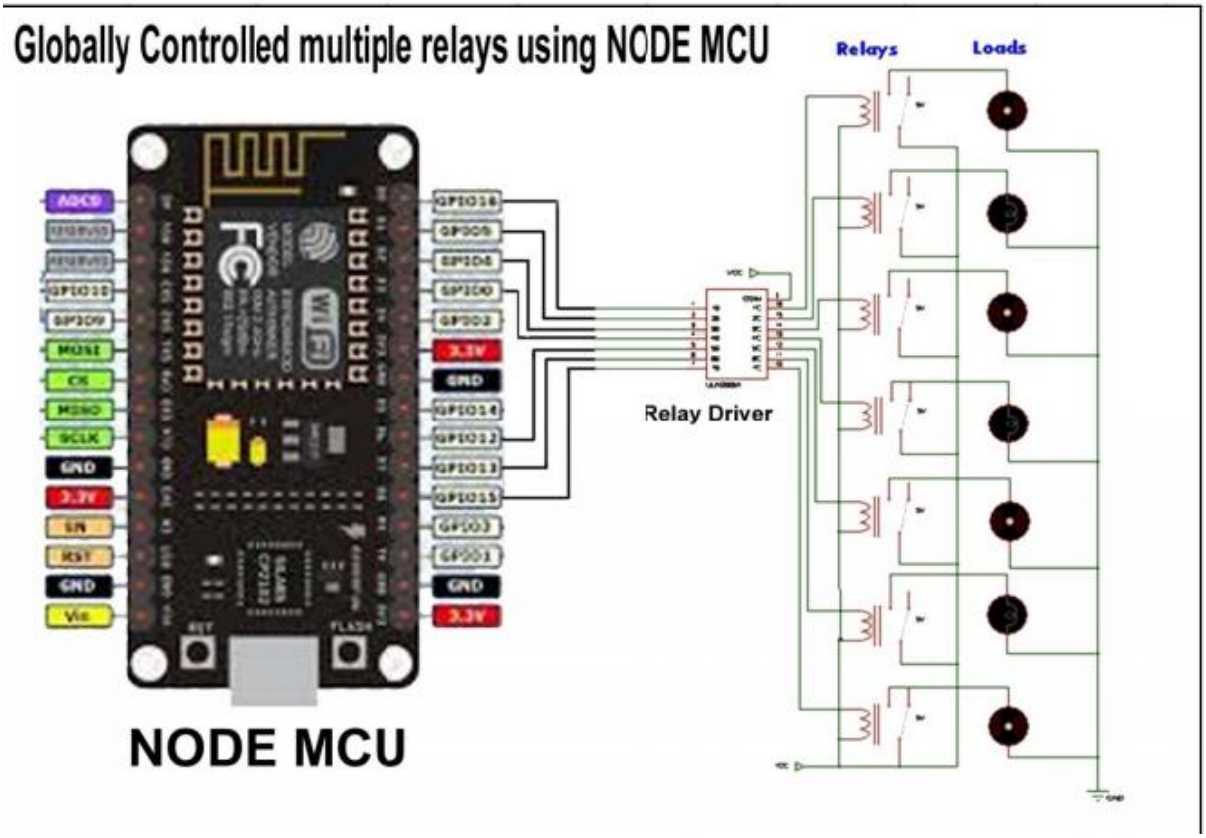



Figure 6: Driver Installation for NODE MCU

### 3.2.3 Interfacing of node mcu with arduino IDE

Firstly open the Arduino IDE. Go to files and click on the preference in the Arduino IDE

## 3.4 Circuit Diagram



# CHAPTER 5

# SOFTWARES

## 5.1 Introduction to Arduino IDE

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.

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

### 5.1.1 Arduino data types:

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

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

**Void:**

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

**Example:**

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

**Boolean:**

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

**Example:**

Boolean state= false ; // declaration of variable with type boolean and initialize it with false.

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

**Char:**A data type that takes up one byte of memory that stores a character value. Character literals are written in single quotes like this: 'A' and for multiple characters, strings use double quotes: "ABC".

However, characters are stored as numbers. You can see the specific encoding in the ASCII chart. This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used. For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

**Example:**

Char chr_a = ‘a’ ;//declaration of variable with type char and initialize it with character a.

Char chr_c = 97 ;//declaration of variable with type char and initialize it with character 97

**Unsigned char:**

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

**Example:**

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

**Byte:**

A byte stores an 8-bit unsigned number, from 0 to 255.

**Example:**

byte m = 25 ;//declaration of variable with type byte and initialize it with 25

**int:**

Integers are the primary data-type for number storage. **int** stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

The **int** size varies from board to board. On the Arduino Due, for example, an **int** stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

**Example:**

int counter = 32 ;// declaration of variable with type int and initialize it with 32.

**Unsigned int:**

Unsigned ints (unsigned integers) are the same as int in the way that they store a 2 byte value. Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). The Due stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 (2^32 - 1).

**Example:**

Unsigned int counter= 60 ; // declaration of variable with type unsigned int and initialize it with 60.

**Word:**

On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero, it stores a 32-bit unsigned number.

**Example**

word w = 1000 ;//declaration of variable with type word and initialize it with 1000.

**Long:**

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from 2,147,483,648 to 2,147,483,647.

**Example:**

Long velocity= 102346 ;//declaration of variable with type Long and initialize it with 102346

**Unsigned long:** Unsigned long variables are extended size variables for number storage and store 32 bits (4 bytes). Unlike standard longs, unsigned longs will not store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

**Example:**

Unsigned Long velocity = 101006 ;// declaration of variable with type Unsigned Long and initialize it with 101006.

**Short:**

A short is a 16-bit data-type. On all Arduinos (ATMega and ARM based), a short stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

**Example:**

short val= 13 ;//declaration of variable with type short and initialize it with 13

**Float:**

Data type for floating-point number is a number that has a decimal point. Floating-point numbers are often used to approximate the analog and continuous values because they have greater resolution than integers.

Floating-point numbers can be as large as 3.4028235E+38 and as low as 3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

**Example:**

float num = 1.352;//declaration of variable with type float and initialize it with 1.352.

**Double:**

On the Uno and other ATMEGA based boards, Double precision floating-point number occupies four bytes. That is, the double implementation is exactly the same as the float, with no gain in precision. On the Arduino Due, doubles have 8-byte (64 bit) precision.

**Example:**

double num = 45.352 ;// declaration of variable with type double and initialize it with 45.352.

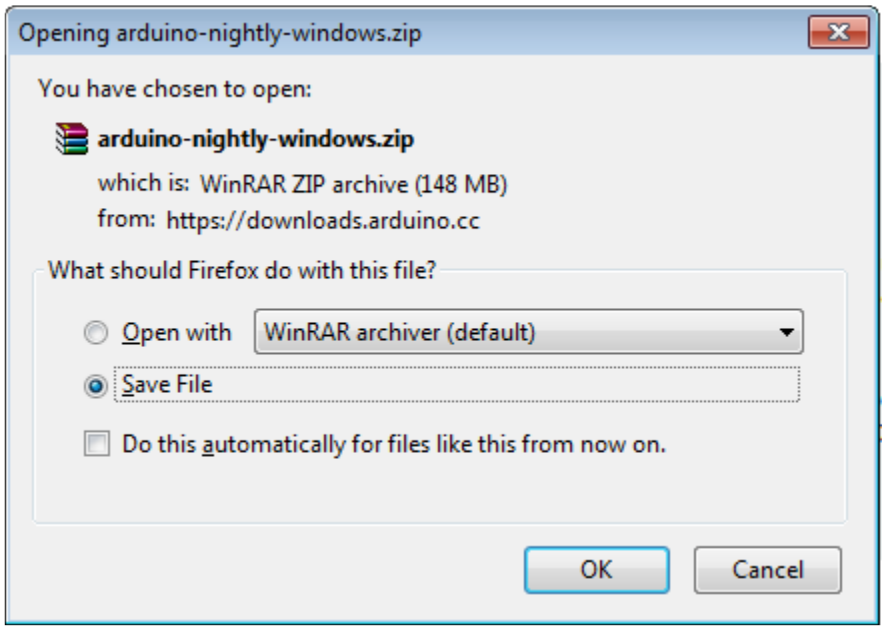
In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

**Step 1:** First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

**Figure 5.1:** USB Cable

**Step 2: Download Arduino IDE Software**.

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.



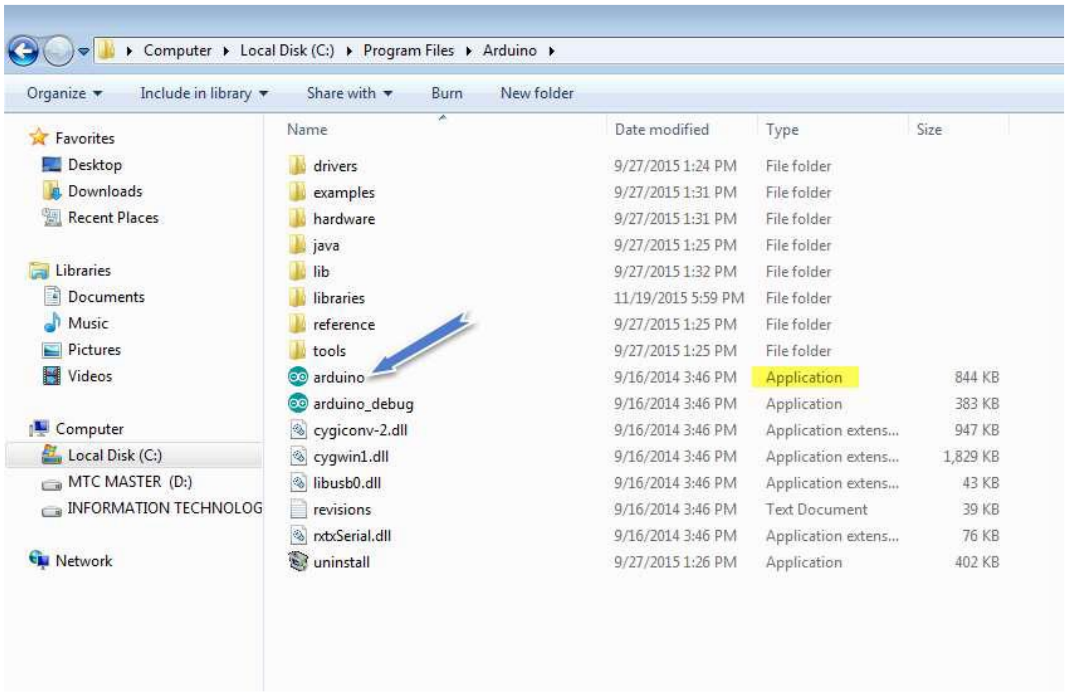

**Step 3: Power up your board**.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from

the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port. Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

**Step 4: Launch Arduino IDE.**

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double click the icon to start the IDE.

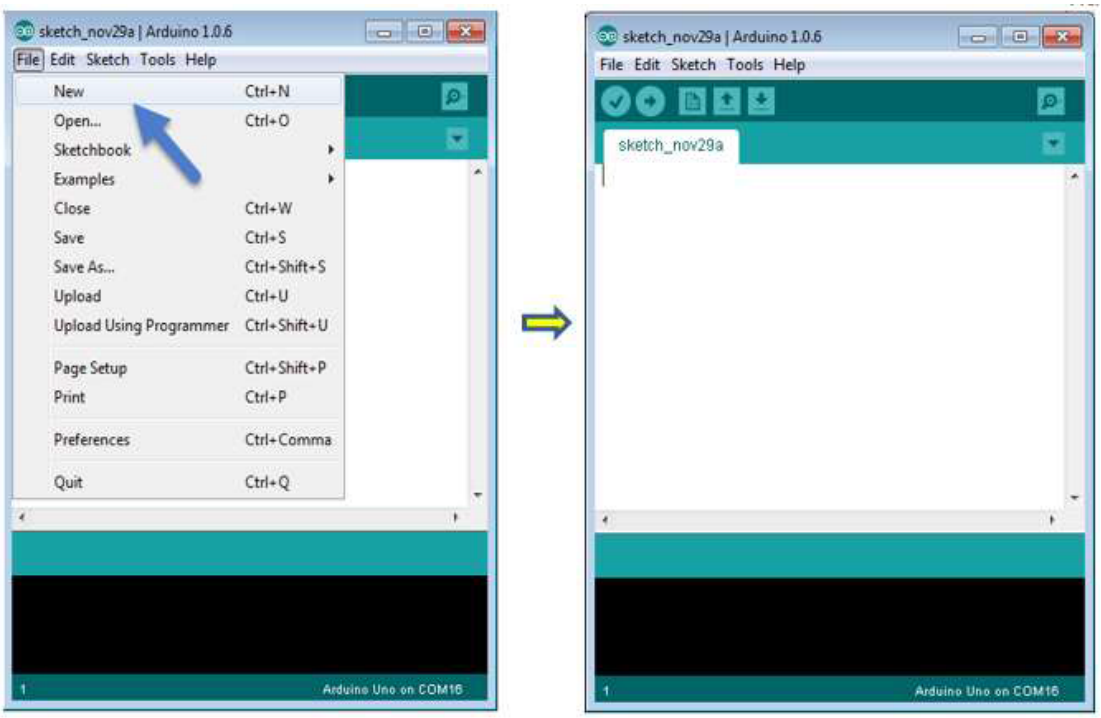



**Step 5: Open your first project.**

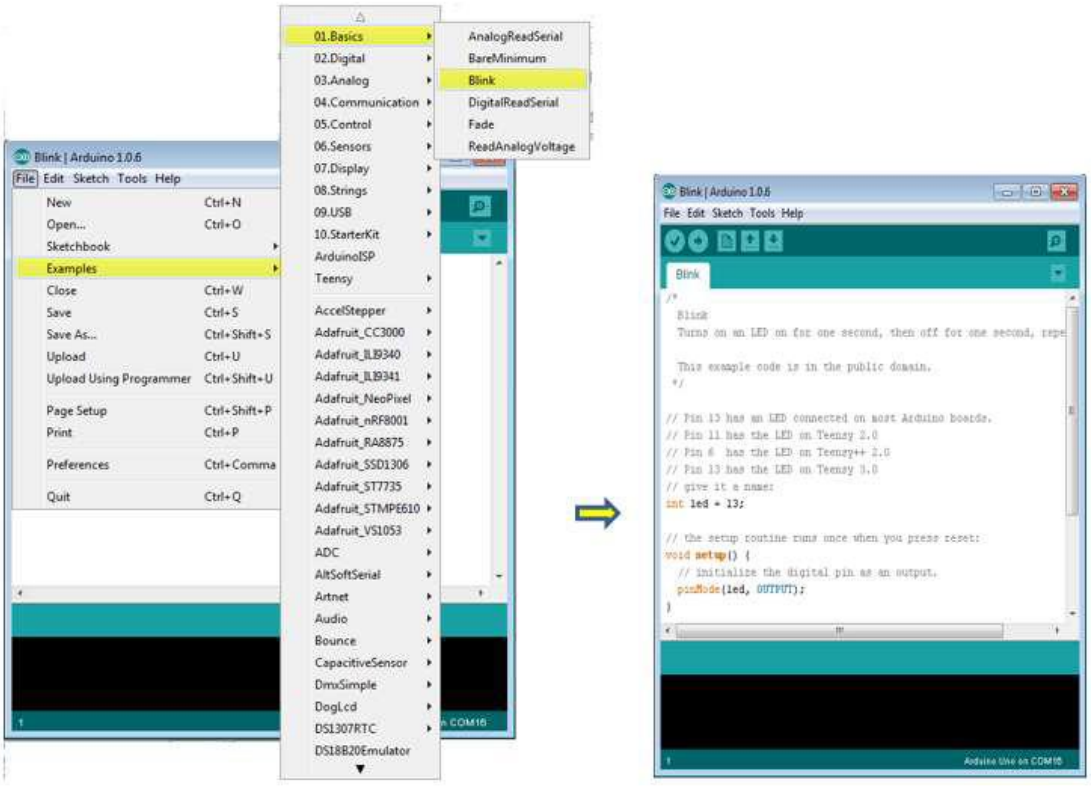
Once the software starts, you have two options:

- Create a new project.
- Open an existing project example.

To create a new project, select File --> New. To open




To open an existing project example, select File -> Example -> Basics -> Blink.

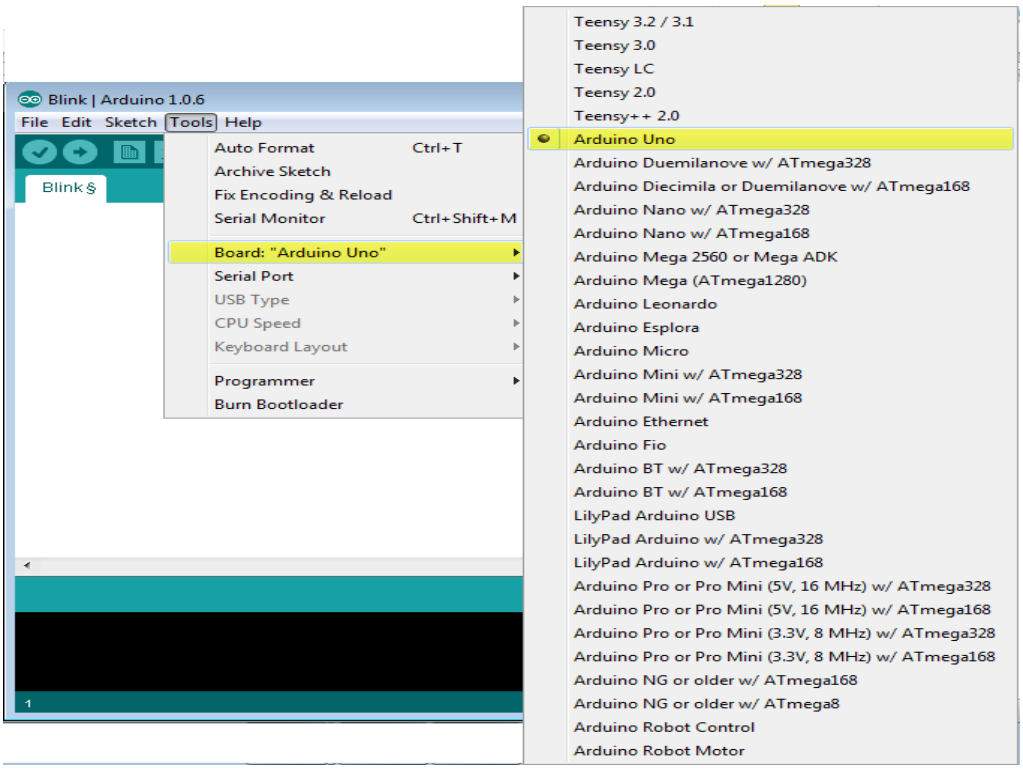



Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

**Step 6: Select your Arduino board.**

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

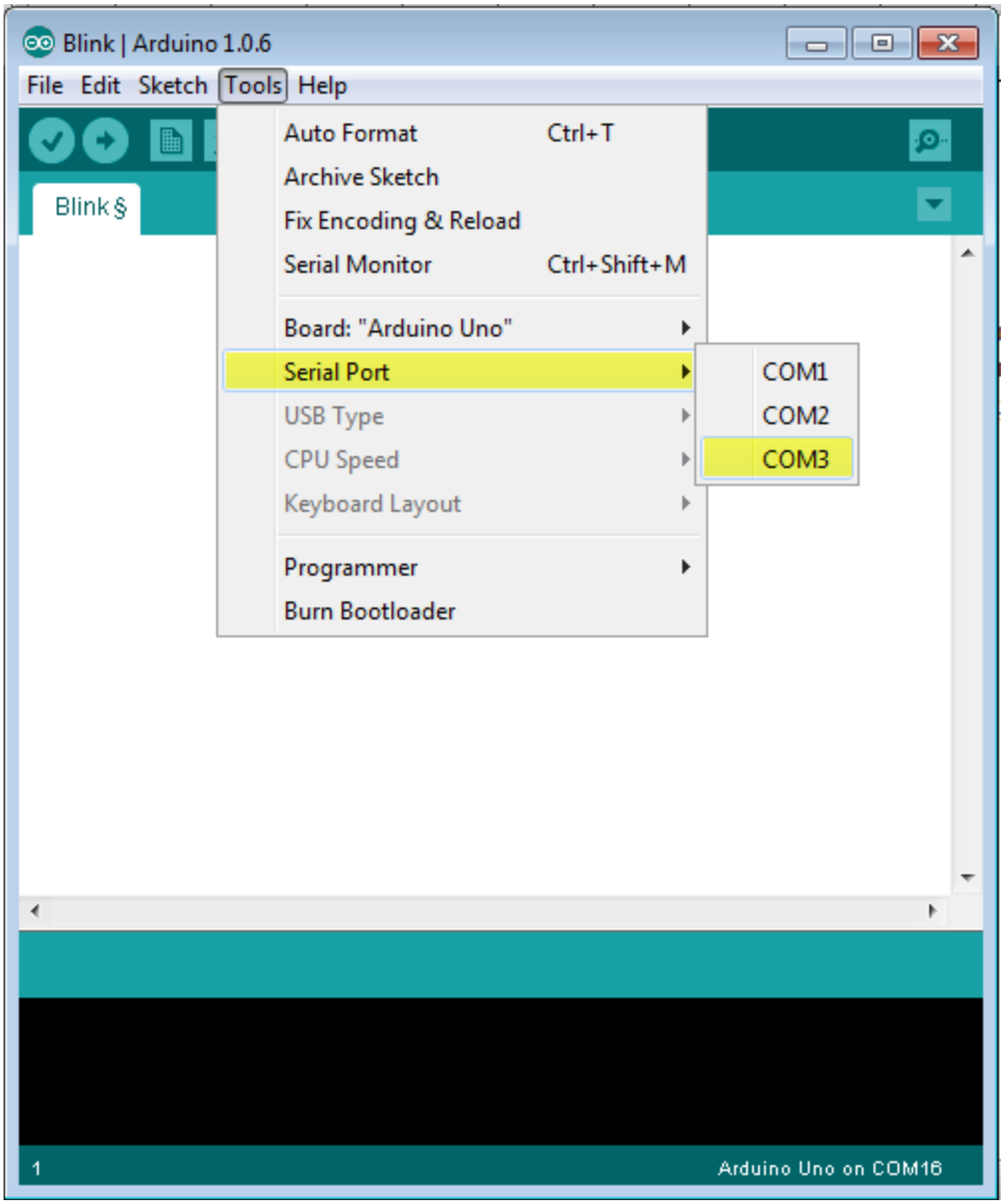
Go to Tools -> Board and select your board



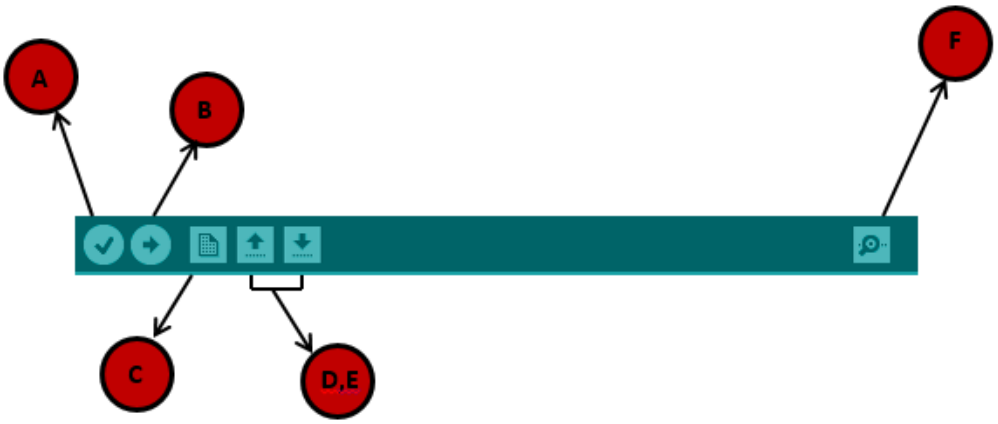

Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using

**Step 7: Select your serial port.**

Select the serial device of the Arduino board. Go to **Tools** ->**Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.




**Step 8: Upload the program to your board.** Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.




**A-** Used to check if there is any compilation error.

**B-** Used to upload a program to the Arduino board.

**C-** Shortcut used to create a new sketch.

**D-** Used to directly open one of the example sketch.

**E-** Used to save your sketch.

**F-** Serial monitor used to receive serial data from the board and send the serial data to the board.

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

**Note:** If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

**Arduino programming structure**

In this chapter, we will study in depth, the Arduino program structure and we will learn more new terminologies used in the Arduino world. The Arduino software is open-source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL.

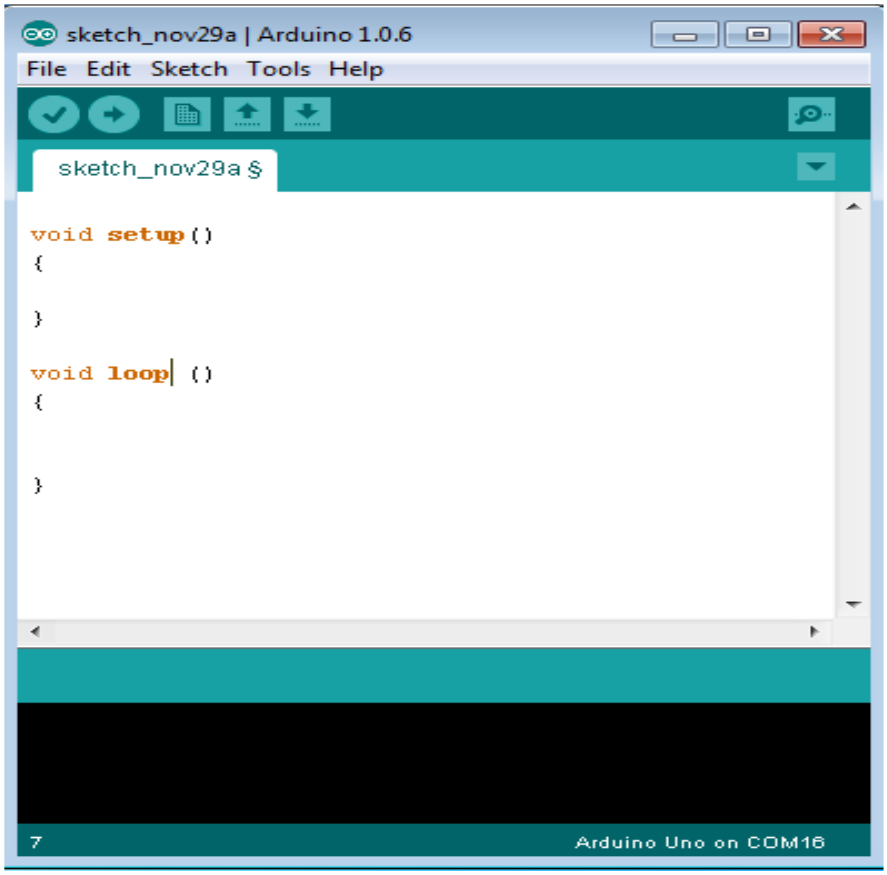
**Sketch:** The first new terminology is the Arduino program called "**sketch**".

**Structure**

Arduino programs can be divided in three main parts: **Structure**, **Values** (variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the **Structure**. Software structure consist of two main functions:

- Setup( ) function
- Loop( ) function




Void setup ( )

{

}

**PURPOSE**:

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

**INPUT**

**OUTPUT**

**RETURN**

Void Loop ( )

{

}

**PURPOSE:**

After creating a **setup()** function, which initializes and sets the initial values, the **loop()** function does precisely what its name suggests, and loops secutively, allowing your program to change and respond. Use it to actively control the Arduino board.

**INPUT**

**OUTPUT**

**RETURN**

# CHAPTER 6

# RESULTS

## 6.1. RESULTS

After the successful connection from home applications to the server, the data of sensor are sent to the web server for monitoring of the system. The figure shows the web server page which will allow us to monitor and control the system. By entering the assigned IP address in the web browser this web server page will appear.

The web server gives the information about the temperature in different places of the house and motion state in the house. It also gives the status of the various electrical appliances like light, fan etc which we can control remotely. The main result we can see, by operating devices from anywhere with our hands.

# CHAPTER 7

# CONCLUSION

## 7.1. CONCLUSION

The next phase for the home automation market will occur based on a few key improvements in the technology available in automation, such as improvements in wireless automation solutions as well as lowering of price points as the market begins to accept home automation usage in larger volumes.

Some trends that we foresee for this phase of the industry are Big companies like Philips, siemens & scheider will eventually bring out fairly mass market automation produces with appealing user interface hot a lower price point today, and more people will be able to afford the products. Some foreign players will have niche in high and automation and focus fun the premium market.

# CHAPTER 8

# REFERENCES

## 8.1 REFERENCES

1. P. S. Pandey, P. Ranjan, M. K. Aghwariya, “The Real-Time Hardware Design and Simulation of Thermoelectric Refrigerator System Based on Peltier Effect” ICICCD 2016 DOI 10.1007/978-981-10-1708-7_66, Vol. 7, pp. 581-589, (2016).

2. C.K.Gomathy.(2010),"Cloud Computing: Business Management for Effective Service Oriented Architecture" International Journal of Power Control Signal and Computation (IJPCSC), Volume 1, Issue IV, Oct - Dec 2010, P.No:22-27, ISSN: 0976-268X .

.