Internet of Things (IoT)

Koshish Foundation Research Lab (KFRL)

Create Your Own IoT Devices

Powered by:
Koshish Foundation Research Lab (KFRL)
Computer & Information System Engineering Dept
N.E.D University of Engineering & Technology

Contents

CHAPTER: 1
INTRODUCTION

1.1 MICROCONTROLLER VS. MICROPROCESSOR

Table: 1 Comparison between Microcontroller and Microprocessor

Microcontroller	Microprocessor
Microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip.	Microprocessor is an IC which has only the CPU inside i.e. only the processing powers such as Intel's Pentium 1,2,3,4, core 2 duo, i3, i5 etc.
Designed to perform specific tasks.	Designed to perform unspecific tasks.
The relationship between input and output is not defined.	The relationship of input and output is defined.
Need small resources like RAM, ROM, I/O ports etc.	Need high amount of resources like RAM, ROM, I/O ports etc.
Applications include developing software, games, websites, photo editing, creating documents etc.	Application includes Desktop PC's, Laptops, notepads etc.

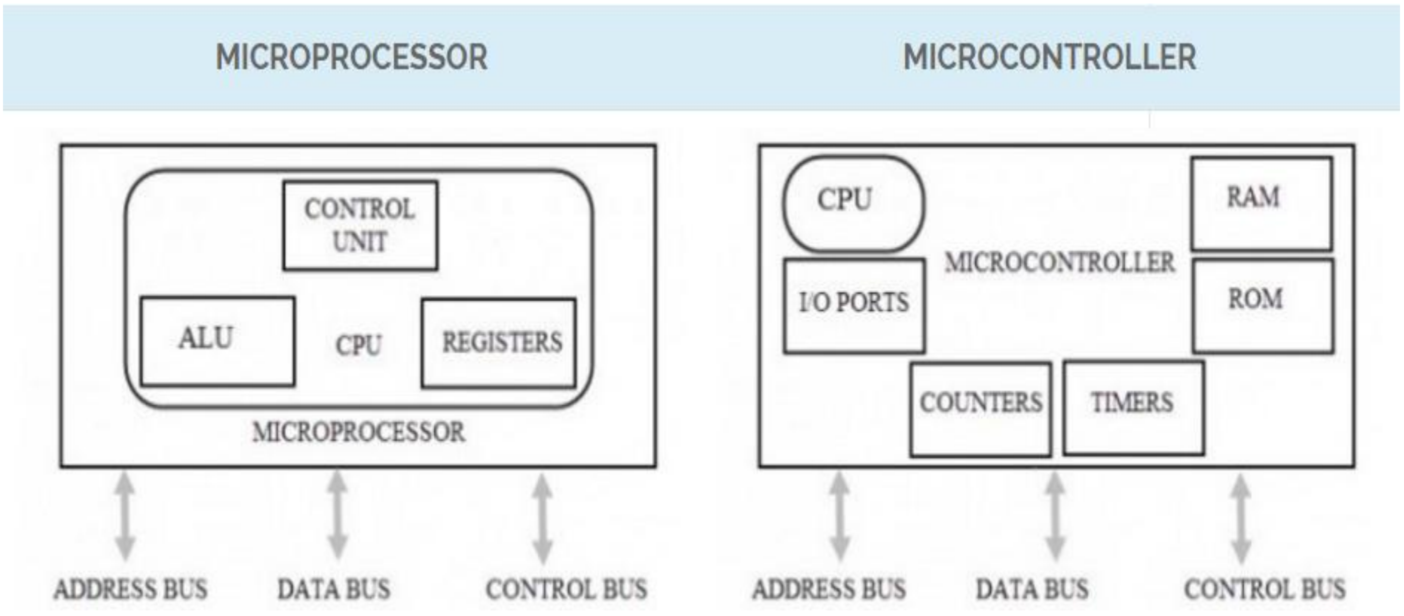


Fig.1: Block diagram of Microcontroller & Microprocessor

1.2 ARDUINO:

Arduino is an open-source prototyping platform based on easy-to-use hardware and software. Arduino boards are able to read inputs, light on a sensor, a finger on a button, or a Twitter message and turn it into an output, activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

We will be using Arduino UNO board during this workshop. The UNO is the best board to get started with electronics and coding. It is the most robust board you can start playing with. It is the most used and documented board of the whole Arduino family.

1.2.1 PINOUT CONFIGURATION:

The Arduino Uno is a microcontroller board based on the ATmega328p microcontroller. It has following pin out:

- 14 digital input/output pins (6 can be used as Pulse Width Modulation (PWM) outputs)
- 6 analog inputs
- Crystal oscillator
- USB connection
- Power jack
- In-Circuit Serial Programming ICSP header
- Reset button

It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. It makes easy to build and debug projects.

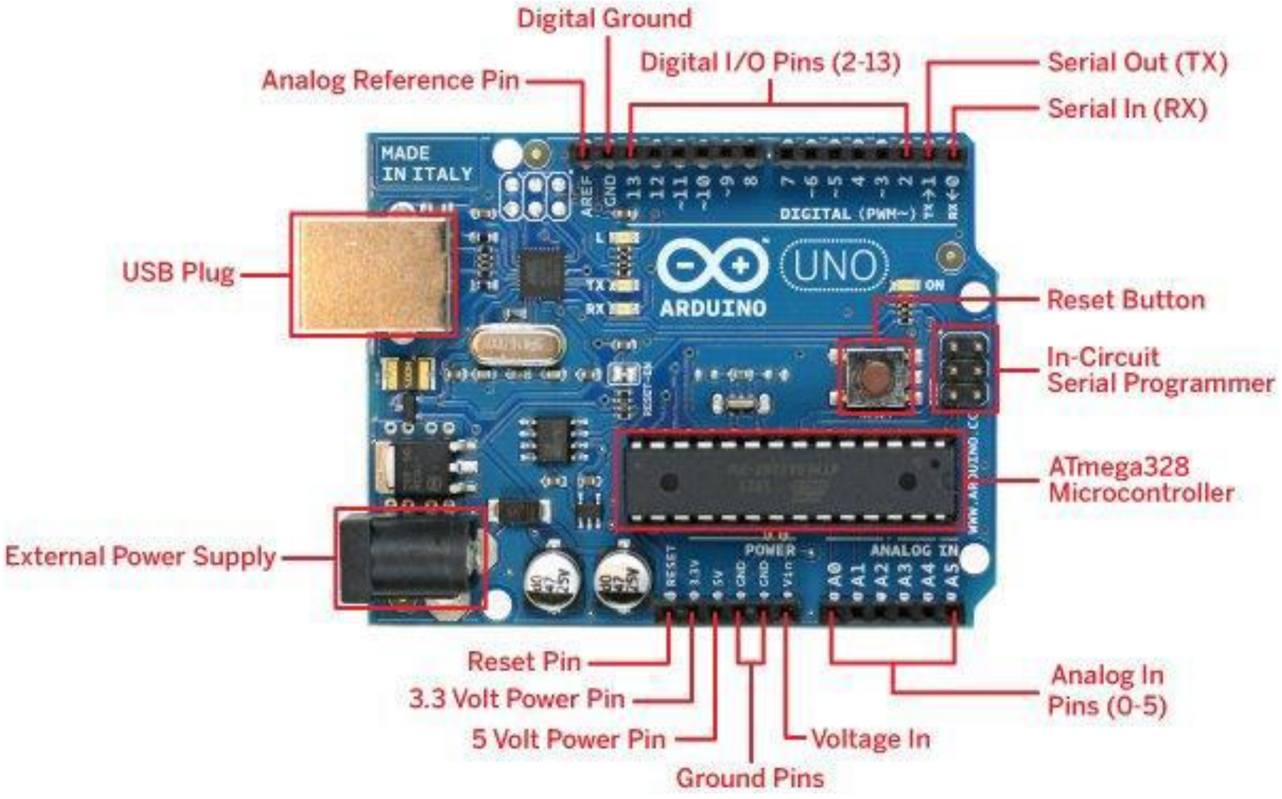


Fig. 2: Arduino UNO board pin description

- **Digital pins:**
These pins are used with digitalRead(), digitalWrite(), and analogWrite(). analogWrite() works only on the pins with the PWM symbol as shown in Fig. 2.
- **Pin 13 LED:**
This pin includes only actuator built-in to your board. Besides being a handy target for your first blink sketch, this LED is very useful for debugging.
- **Power LED:**
This pin indicates that your Arduino is receiving power. It is very useful for debugging.
- **ATmega microcontroller:**
The microcontroller is the heart of your board.
- **Analog in:**
These pins are use with analogRead().
- **GND and 5V:**
These pins are used to provide +5V power and ground to your circuits.
- **Power connector:**
This is used to power your Arduino when it's not plugged into a USB port for power. It can accept voltages between 7-12V.
- **TX and RX LEDs:**
These LEDs indicate communication between your Arduino and your computer. They also flicker rapidly during sketch upload as well as during serial communication. These are also useful for debugging.
- **USB port:**
This port is used for powering your Arduino Uno, uploading your sketches to your Arduino, and for communicating with your Arduino.
- **Reset button:**
This button is used to resets the ATmega microcontroller.

1.2.2 Arduino Software (IDE):

The Arduino Integrated Development Environment (IDE) contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and to carry out communication between them. The software and step-by-step directions for its installation is available at: http://arduino.cc/en/Main/Software

1.2.2.1 Install the Drivers for Windows:

After successfully installing Arduino IDE the next step is to install the supported drivers for the cable that has been used for connecting the Arduino board to Computer. Following steps need to be followed for successful installation of the drivers.

1) Plug in board via USB and wait for Windows to begin its driver installation process. After a few moments, the process will fail i.e., this is not unexpected.
2) Click on the "Start Menu", and open up the "Control Panel".
3) While in the "Control Panel", navigate to "System and Security". Next, click on "System". Once the System window is up, open the "Device Manager".
4) Look under Ports (COM & LPT). There must be an open port named "Arduino UNO (COMxx)".
5) Right click on the "Arduino UNO (COMxx)" port and choose the "Update Driver Software" option.

6) Next, choose the "Browse my computer for Driver software" option.
7) Finally, navigate to and select the Uno's driver file, named "ArduinoUNO.inf", located in the "Drivers" folder of the Arduino Software download.
8) Windows will finish up the driver installation from there.

1.2.2.2 Integrated Development Environment (IDE):

Different menus with variety of options are available in Arduino IDE which makes it easy to use. Few important and most frequently used icons are discussed as under.

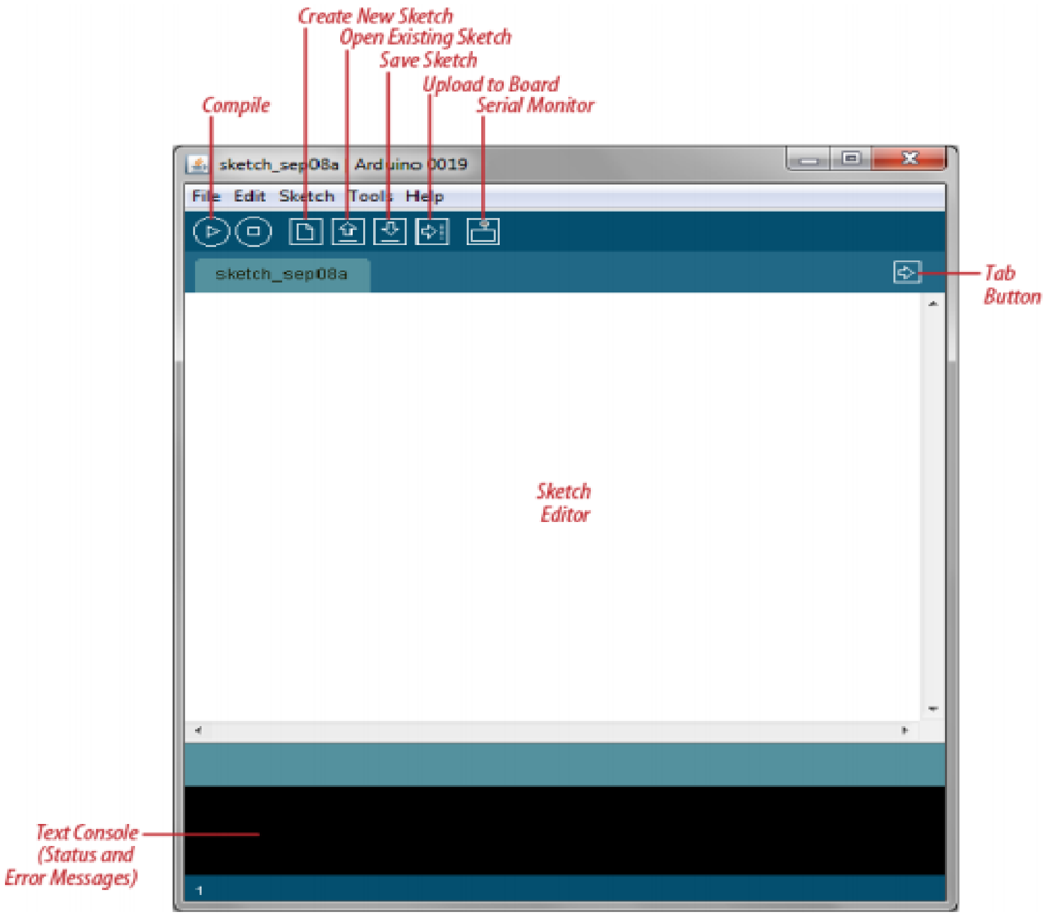


Fig. 3: IDE Window

- **Compile:**

Before program "code" can be sent to the board, it first needs to be converted into instructions that the board understands. This process is called compiling.

- **Stop:**

This stops the compilation process.

- **Create new Sketch:**

This opens a new window to create a new sketch.

- **Open Existing Sketch:**

This loads a sketch from a file on computer.

- **Save Sketch:**

This saves the changes to the working sketch.

- **Upload to Board:**

This compiles the code and then transmits it over the USB cable to board.

- **Serial Monitor:**

This is used to receive data from external devices through USB cable /DB9 connector.

- **Tab Button:**

  This lets you to create multiple files in sketch. This is for more advanced programming that is not a part of this course.
- **Sketch Editor:**

  This is where the sketches are written.
- **Text Console :**

  This shows what the IDE is currently doing and error messages are also displayed here.

1.2.2.3 How to Upload a Sketch:

The Arduino IDE consists of few example sketches which can be uploaded to Arduino board by using following steps:

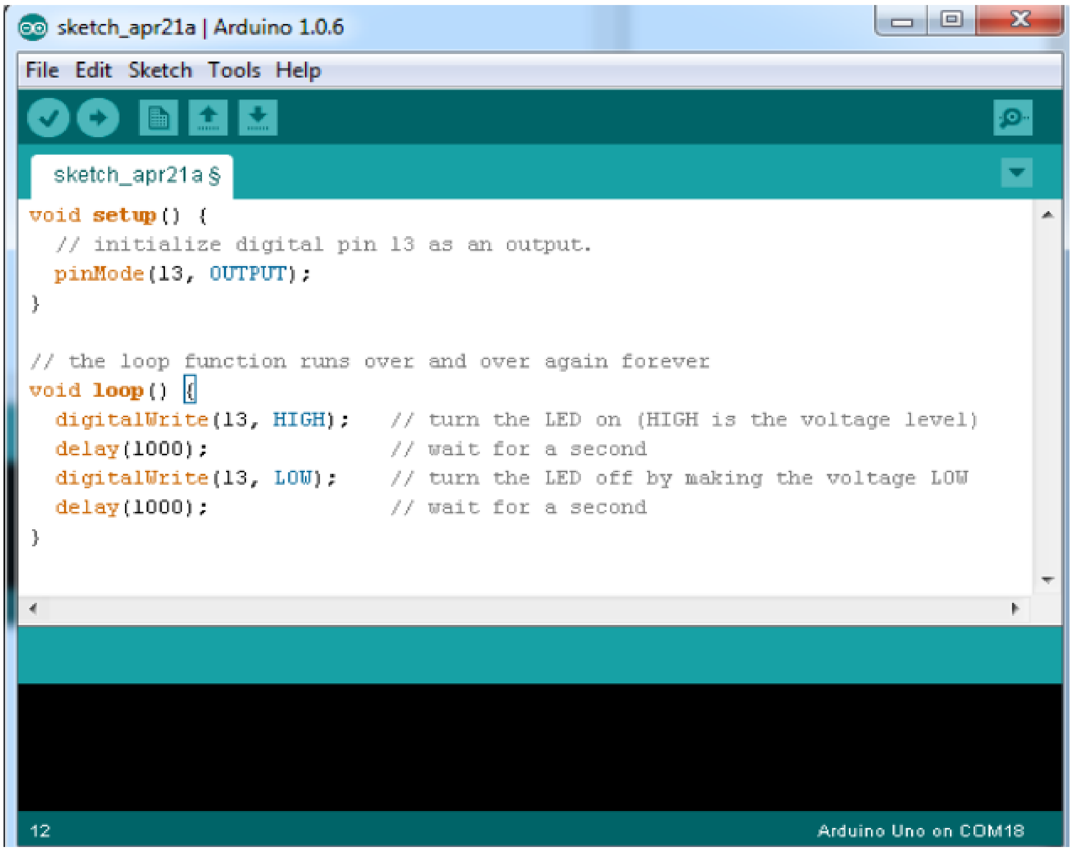
1. Double-click the Arduino application.
2. Open the LED blink example sketch: File > Examples > 1.Basics > Blink
3. Select Arduino Uno under the Tools > Board menu.
4. Select serial port (if not aware of exact port, disconnect the UNO and the entry that disappears is the right one.)
5. Click the Upload button.
6. After the message “Done uploading” appears, the LED on Arduino will start blinking once a second.

Each sketch of Arduino IDE must contain at least two functions.

- **setup ():**

  This function is called once when the program starts.
- **loop ():**

  This function is called repetitively over and over again.



```
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

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

Fig. 4: Function of setup and loop

1.3 DIGITAL PINS:

Digital pins are much easy to understand as there are only two states, either ON or OFF. In Arduino sketch terms, an ON state is known as HIGH (5V) and OFF state is known as LOW (0V).

1.3.1 PINMODE ():

This command configures the specified pin to behave either as an input or an output.

SYNTAX: pinMode(pin, mode)

1.3.2 DIGITALWIRTE ():

This command writes a **HIGH** or a **LOW** value to a digital pin.

- If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.
- If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pull up on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor

SYNTAX: digitalWrite(pin, value)

1.3.3 DIGITALREAD():

This command reads the value from a specified digital pin, either HIGH or LOW .

SYNTAX: digitalRead(pin)

SAMPLE PROGRAM:

```
int val = 0;   // variable to store the read value

void setup()
{
pinMode(13, OUTPUT);      // sets the digital pin 13 as output
pinMode(7, INPUT);       // sets the digital pin 7 as input
pinMode(8, OUTPUT);
Serial.begin(9600);
}

void loop()
{
  digitalWrite(8, HIGH);
  val = digitalRead(7);   // read the input pin
  Serial.print(val);
  }
```

1.4 SERIAL MONITOR:

The Arduino IDE has a feature that can be of a great help in debugging sketches or controlling Arduino from your computer's keyboard. It displays serial data being sent from Arduino UNO (USB or serial board).

1.4.1 CONFIGURATION OF SERIAL MONITOR

1. Connect Arduino UNO board by USB to your computer to activate the Serial Monitor.
2. To get familiar with using the Serial Monitor, Write the following Sketch into a blank Arduino IDE window.
3. Then verify it and if it's OK, Upload it.
4. Click on Serial Monitor icon at right corner
5. A pop up in a new window
6. Look at the Serial Monitor window.
    - The small upper box is where you can type in characters (hit or click "Send")
    - The larger area (Corner can be dragged to enlarge) is where characters sent From Arduino will be displayed.
    - At the bottom are two pull-downs:
        - One sets the "line ending" that will be sent to Arduino when you or click Send
        - The other sets the Baud Rate (9600) for communications. (If this does not match the value set up in your sketch in Setup, characters will be unreadable).

- SETUP:

In Setup you need to begin Serial Communications and set the Baud Rate (speed) that data will be transferred at. That looks like this:

```
Serial.begin(9600);          // set up Serial library at 9600 bps
```

- LOOP:

Here we can use following commands to send data serially & configure it to the Serial Monitor.

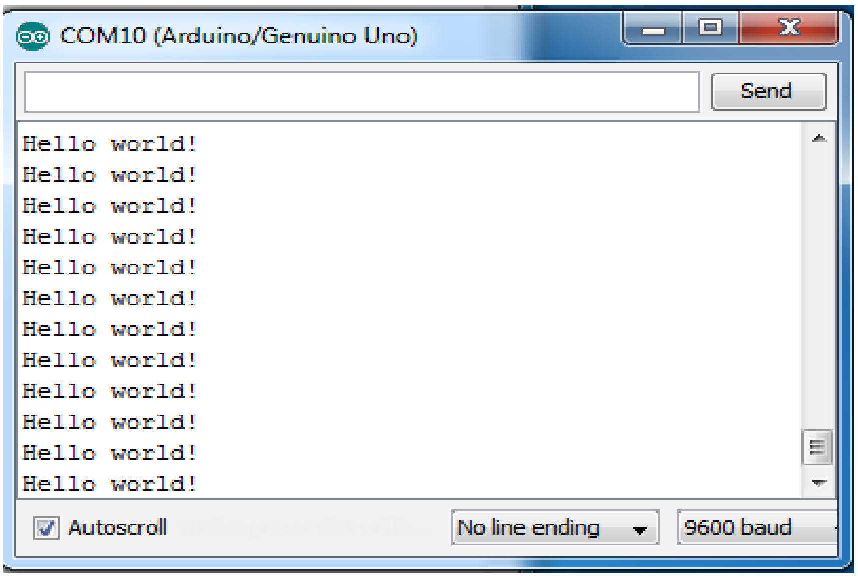
EXAMPLES:

- SERIAL.AVAILABE():
  This command is use to get the number of bytes (characters) available for reading from the serial port.
- SERIAL.FLUSH() :
  This command is used to wait for the transmission of outgoing serial data to complete.
- SERIAL.PRINT(DATA):
  This command is used to print data to the serial port as human-readable ASCII text.
- SERIAL.READ():
  This command is used to reads incoming serial data.

Sample Sketch:

```
/*
* Hello World!
*
* This is the Hello World! For Arduino.
* It shows how to send data to the computer
*/

void setup()                    // run once, when the sketch starts
{
Serial.begin(9600);          // set up Serial library at 9600 bps
}

void loop()                     // run over and over again
{
Serial.println("Hello world!");  // prints hello continuously

delay(1000);  }
```

Output:


COM10 (Arduino/Genuino Uno)

```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

Autoscroll [✓] No line ending 9600 baud

1.5 SWITCHES & RELAYS:

1.5.1 DIP SWITCHES:

A dip switch is a set of small switches in a dual in-line package (DIP) that is used to change the operating mode of a device. They are easier to operate and less likely to get lost.

DIP switches enable you to configure a circuit board for a particular application. The installation instructions should tell you how to set the switches. DIP switches are always toggle switches which mean they have two possible positions on or off. (Instead of on and off, you may see the numbers 1 and 0.)

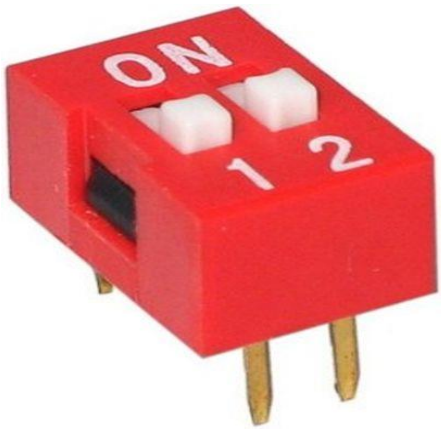


Fig. 5: DIP Switches

1.5.2 RELAYS:

Relay is an electromagnetic device which is used to isolate two circuits electrically and connect them magnetically. They are very useful devices and allow one circuit to switch another one while they are completely separate. They are often used to interface an electronic circuit (working at a low voltage) to an electrical circuit which works at very high voltage. For example, a relay can make a 5V DC battery circuit to switch a 230V AC mains circuit. Thus a small sensor circuit can drive, say, a fan or an electric bulb.

1.5.2.1 PIN CONFIGURATION:

- There are 5 Pins in a relay.
- Two pins A and B are two ends of a coil that are kept inside the relay.
- COM/POLE is always connected to NC (Normally connected) pin.
- As current is passed through the coil A, B, the pole gets connected to NO (Normally Open) pin of the relay.

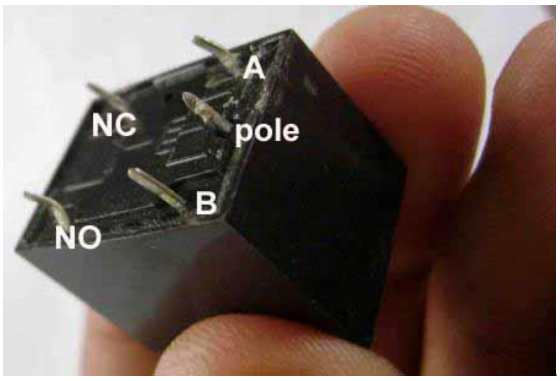


Fig. 6: Relays

Sample Program:

```
const int button1Pin =11 ;     // the number of the pushbutton pin
const int button2Pin=10;
const int ledPin =12;      // the number of the LED pin

// variables will change:
int button1State = 0;         // variable for reading the pushbutton status
int button2State = 0;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);

}

void loop() {
  // read the state of the pushbutton value:
  button1State = digitalRead(button1Pin);
  button2State = digitalRead(button2Pin);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (button1State == HIGH || button2State == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Output:
If switch1 or switch 2 is ON, LED will glow.

Task:

1. Write a program to interface LEDs by using digital pins.
2. Write a program to interface 7 segment displays by using digital pins and print 'INTERFACE' on serial monitor.

Chapter: 2
SENSOR'S INTERFACING

2.1 Analog Input Pins:

The main function of analog pins is to read analog sensors; analog pins also have all the functionality of general purpose input/output (GPIO) pins (same as digital pins 0 - 13).

2.1.1 Pin mapping:

The analog pins can be used identically to digital pins, using aliases A0 (for analog input 0), A1, etc. For example, the code would look like this to set analog pin 0 to an output, and to set it HIGH:

```
pinMode(A0, OUTPUT);
digitalWrite(A0, HIGH);
```

2.1.2 Pullup resistors:

The analog pins also have pullup resistors, which work identically to pull up resistors on digital pins. They are enabled by issuing a command such as:

```
digitalWrite(A0, HIGH);  // set pullup on analog pin 0
```

While the pin is an input, be aware however that turning on a pullup will affect the values reported by analogRead().

2.1.3 analogWrite():

This command writes an analog value to a pin. After a call to analogWrite(),pin will generate a steady square wave of specified duty cycle until the next call to analogWrite() . On Arduino Uno pins 5 and 6 have a frequency of approximately 980 Hz.
In most of the Arduino boards this function works on pins 3, 5, 6, 9, 10, and 11.

Syntax: analogWrite(pin, value)

2.1.4 analogRead():

This command reads the value from the specified analog pin. It will map input voltages between 0 and 5 volts into integer values between 0 and 1023.

Syntax: analogRead(pin)

2.2 PULSE WIDTH MODULATION (PWM):

PWM stands for Pulse Width Modulation. Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means.

Arduino uses this powerful PWM technique for controlling analog circuits with its digital outputs. Digital control uses to be only turn on (full 5v) or off (0v) in the binary format, and this on/off pattern can generate a square wave signal. For example if you want a LED to be half bright, you can either reduce the current across the LED into half or using this the more flexible PWM technique by sending 50% duty cycle square wave signal to the LED.

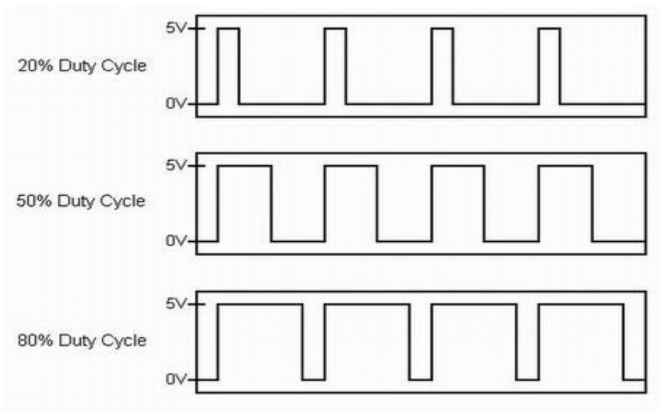


Fig.6: PWM cycle

On an Arduino Uno, PWM output is possible on digital I/O pins 3, 5, 6, 9, 10 and 11. On these pins the analogWrite function is used to set the duty cycle of a PWM pulse train that operates at approximately 500 Hz[2]. Thus, with a frequency fc = 500 Hz, the period is τc = 1/fc ~ 2 ms. As with conventional digital I/O, the pinMode function must be called first to configure the digital pin for output.

PWM has several uses:

- Dimming an LED
- Providing an analog output; if the digital output is filtered, it will provide an analog voltage between 0% and 100%.
- Generating audio signals.
- Providing variable speed control for motors.
- Generating a modulated signal, for example to drive an infrared LED for a remote control.

Sample Program:

Program 1:

```
//You can "manually" implement PWM on any pin by
repeatedly turning the pin on and off for the desired times.
void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
  digitalWrite(13, HIGH);
  delayMicroseconds(100); // Approximately 10% duty
cycle @ 1KHz
  digitalWrite(13, LOW);
  delayMicroseconds(1000 - 100);
}
```

Program 2:

```
//PWM program by using the PWM pin
int LED_pin = 11;
int level1 = 20;
int level2 = 200;

void setup() {
pinMode(LED_pin, OUTPUT);
}
void loop() {

analogWrite(LED_pin, level1);
delay(1000);
analogWrite(LED_pin, level2);
delay(1000);
}
```

Output:

Program 1 causes an LED to glow at low brightness level.
Program 2 cause an LED to glow at two different brightness levels.

2.3 SENSORS:

2.3.1 LIGHT DEPENDANT RESISTORS (LDR):

A light-dependent resistor (LDR) is a light-controlled variable resistor. The resistance of a LDR decreases with increasing incident light intensity. In the dark, a LDR can have a resistance as high as several mega-ohms (MΩ), while in the light; a LDR can have a resistance as low as a few hundred ohms.

2.3.1.1 APPLICATIONS:

LDRs applications can be found in many consumer items such as camera light meters, clock radios, alarm devices, night lights, outdoor clocks, solar street lamps and solar road studs, etc.

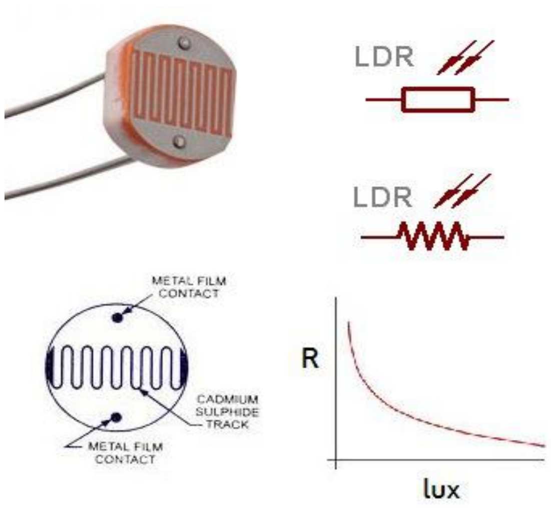


Fig. 7: LDR

SAMPLE PROGRAM:

```
int sensorPin = A0;    // select the input pin for the potentiometer
int Relay = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
pinMode(Relay, OUTPUT);
Serial.begin(9600);
}
void loop() {
  // read the value from the sensor:
sensorValue = analogRead(sensorPin);
Serial.println(sensorValue);
if (sensorValue<=500){
digitalWrite(Relay, HIGH);
Serial.print(Relay);
delay(90); }
else{
  // turn the ledPin off:
digitalWrite(Relay, LOW);
Serial.print(Relay);
delay(90); }}
```

Output:

If the sensor value is in range of 900 relay becomes low and if less than 900 relay becomes high.

2.3.2 Humidity & Temperature Sensor:

The DHT11 is a relatively cheap sensor for measuring temperature and humidity. The DHT11 sensor includes a resistive-type humidity measurement component, an NTC temperature measurement component and a high-performance 8-bit microcontroller inside, and provides calibrated digital signal output.

2.3.2.1 Pin Configuration:

- VCC connected to +3.3V~5V
- DATA connected to the microcontroller IO port
- NC (Normally connected) is Null
- GND connected to ground

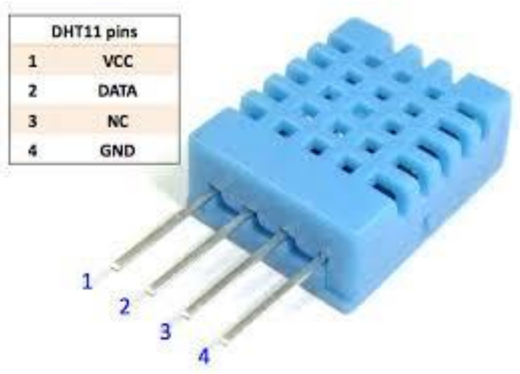


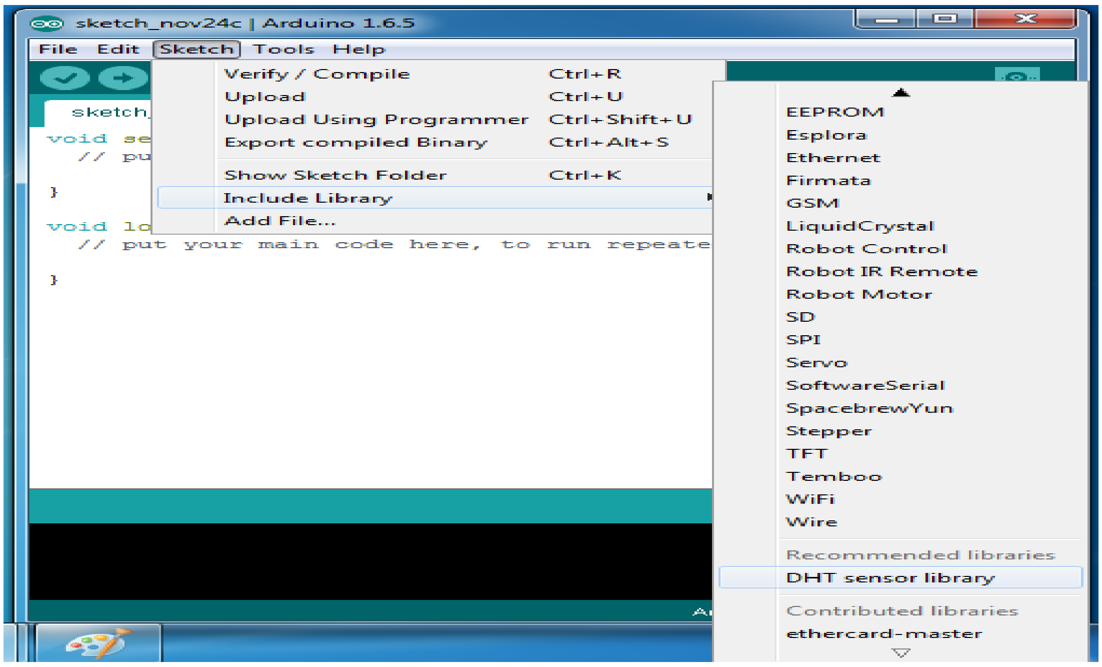
Fig. 8: DHT11 sensor

Temperature range: 0 / +50°C ±2°C
Humidity range: 20-90%RH ±5%RH

2.3.2.2 Download DHT11 Library:

This is an Arduino library for the DHT series of temperature/humidity sensors.

1. Click on the link https://github.com/adafruit/DHT-sensor-library
2. To download, click the Download ZIP button in the top right corner.
3. Uncompressed the ZIP folder. Check that the DHT folder contains DHT.cpp and DHT.h
4. Copy the DHT library folder & paste it in your Arduino libraries folder. You may need to create the libraries subfolder if it's your first library.
5. Restart the IDE.



```
sketch_
void se
  // pu
}

void lo
  // put your main code here, to run repeate

}
```

Fig. 9: DHT11 library

2.3.2.3 Arduino Interfacing with temp/humidity Sensor:

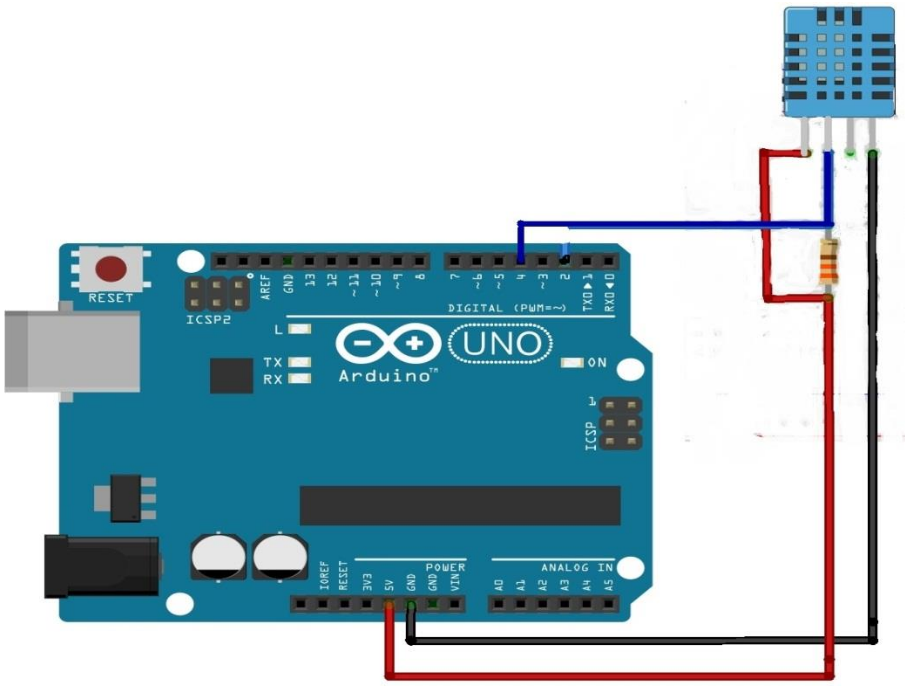
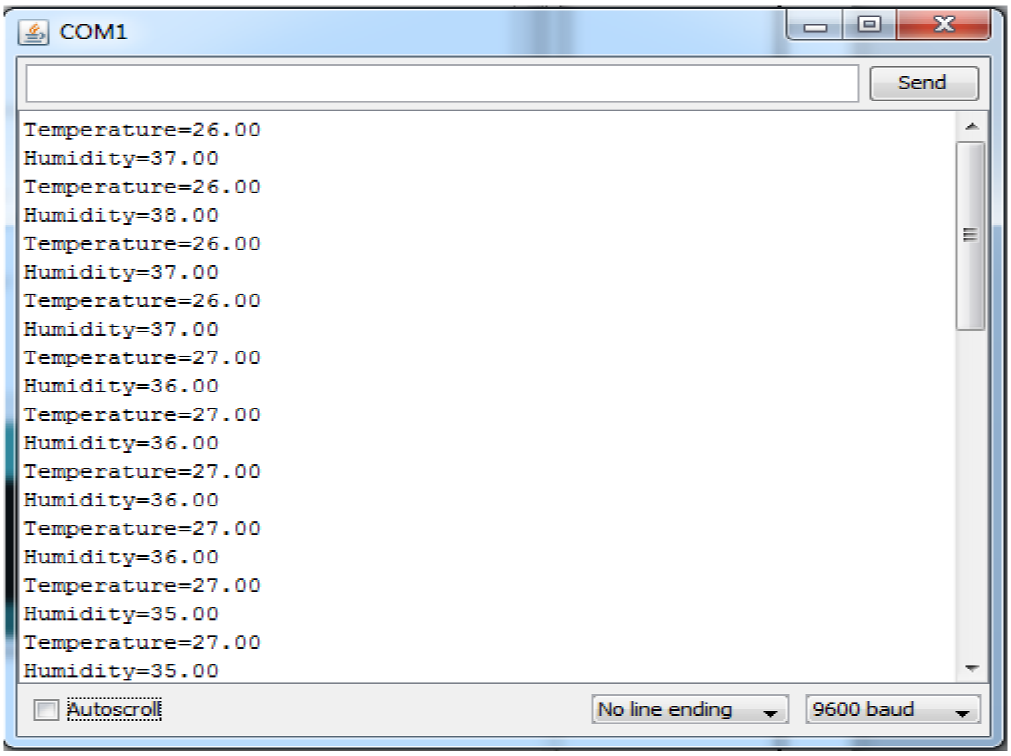


Fig. 10 Humidity/Temperature (DHT11) Sensor Interfacing

Sample program:

```
        #include <dht.h>
        dht DHT;
        #define dht_dpin 4
        void setup() {
          // put your setup code here, to run once:
        Serial.begin(9600);
        }

        void loop() {
          // put your main code here, to run repeatedly:
        DHT.read11(dht_dpin);
        float Hum=DHT.humidity;
        float Tem=DHT.temperature;
        Serial.print("Temperature=");
        Serial.println(Tem);
        Serial.print("Humidity=");
        Serial.println(Hum);
        delay(1000);
}
```

OUTPUT:



```
Temperature=26.00
Humidity=37.00
Temperature=26.00
Humidity=38.00
Temperature=26.00
Humidity=37.00
Temperature=26.00
Humidity=37.00
Temperature=27.00
Humidity=36.00
Temperature=27.00
Humidity=36.00
Temperature=27.00
Humidity=36.00
Temperature=27.00
Humidity=36.00
Temperature=27.00
Humidity=35.00
Temperature=27.00
Humidity=35.00
```

TASK:

1. Write a program to control LEDs by varying potentiometer value.

CHAPTER: 3
WI-FI MODULE (ESP8266)

3.1 SOFTWARE SERIAL:

SoftwareSerial is used to create an instance of a SoftwareSerial object. You need to call SoftwareSerial.begin() to enable communication. The Arduino hardware has built-in support for serial communication on pins 0 and 1 (which also goes to the computer via the USB connection). You can use 1 at a time either serial monitor or pin (0, 1). To see result on serial monitor you have to make another two pins as RX & TX. This hardware allows the Atmega chip to receive serial communication even while working on other tasks.

3.1.1 SOFTWARESERIAL LIBRARY:

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality.

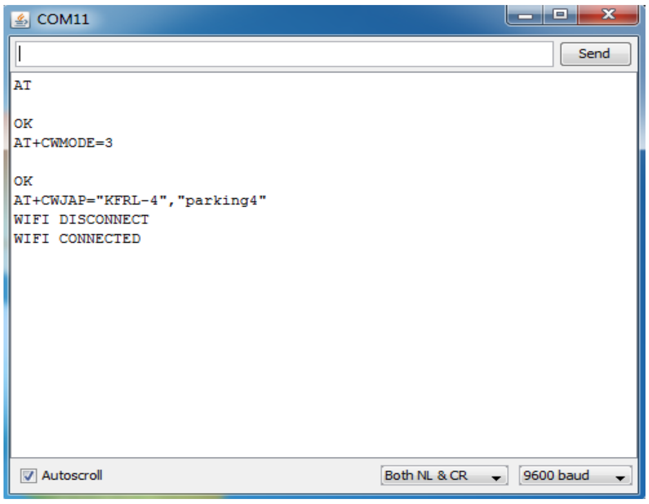
3.1.2 PARAMETERS:

- **Rx Pin**:
  This pin is used to receive serial data.
- **Tx Pin**:
  This pin is used to transmit serial data.
- **inverse_logic**:
  This command is used to invert the sense of incoming bits (the default is normal logic).

PROGRAM:

```
#include <SoftwareSerial.h>
#define DEBUG true
SoftwareSerial esp8266(7,6);
String espmode = "3";
String SSid = "KFRL-4";       //enter your SSID here
String password = "parking4"; //enter your password here
void wifisetup(String mode,String ssid,String pass);
void setup()
{
  Serial.begin(9600);
  esp8266.begin(9600);
}

 void loop()
{
  wifisetup(espmode,SSid,password);
  if(esp8266.available()){

if(esp8266.find("+IPD,")){
{
delay(1000);
int connectionId = esp8266.read()-48; // subtract 48 because the read() function returns
                                      // the ASCII decimal value and 0 (the first decimal number) starts at 48
}}}}
 //SendData Function
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())
{
char c = esp8266.read(); // read the next character.
response+=c;
}
}
 if(debug)
 {
     Serial.print(response);
 }
   return response;
}

//WiFi connectivity Functunction
void wifisetup(String mode,String ssid,String pass)
{
  sendData("AT\r\n",2000,DEBUG); // reset module
 String mode0= "AT+CWMODE=";
 mode0 += mode;
 mode0 += "\r\n";
 sendData(mode0,2000,DEBUG); // configure as access point
 String ssidset="AT+CWJAP=\"";
  ssidset += ssid ;
  ssidset += "\",\"" ;
  ssidset += pass ;
  ssidset += "\"" ;
  ssidset += "\r\n" ;
  sendData(ssidset,5000,DEBUG);
  return;
}
```

OUTPUT:



```
AT

OK
AT+CWMODE=3

OK
AT+CWJAP="KFRL-4","parking4"
WIFI DISCONNECT
WIFI CONNECTED
```

3.2 ESP-8266 Wi-Fi Module:

ESP8266 is an impressive, low cost Wi-Fi module. It is highly integrated chip designed to provide full internet connectivity in a small package.

ESP8266 can be used as an external Wi-Fi module. It uses the standard AT Command set by connecting it to any microcontroller.

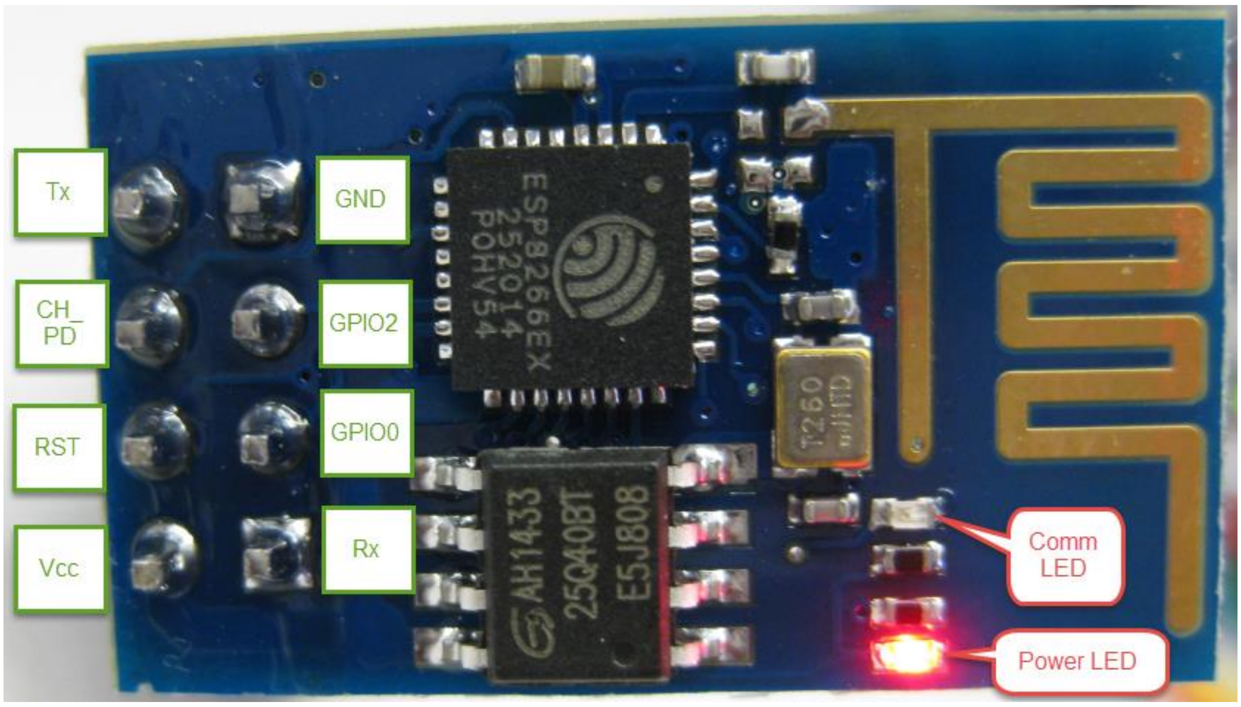


Fig.11 ESP8266 Wi-Fi Module

3.2.1 Features:

The feature list is impressive and includes:

- 802.11 b/g/n protocol
- Wi-Fi Direct (P2P), soft-AP
- Integrated TCP/IP protocol stack

3.2.2 USAGE:

The module accepts commands via a simple serial interface. It then responds back with the operation's outcome (assuming everything is running correctly). Also, once the device is connected and is set to accept connections, it will send unsolicited messages whenever a new connection or a new request is issued.

3.2.3 Pin Configuration:

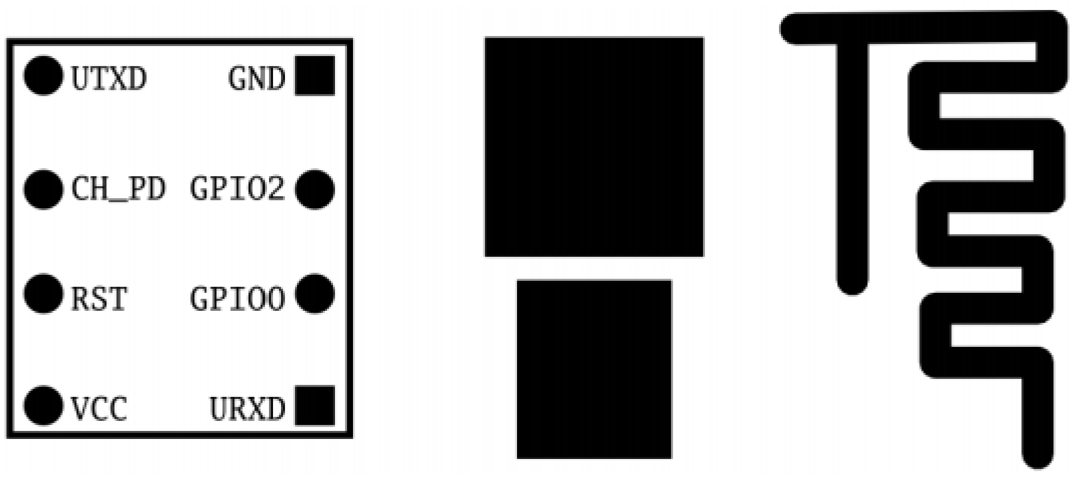


Fig.12: ESP8266 Pin Configuration

- VCC shall be connected to the 3.3V power supply
- GPIO0 and GPIO2:
- These pins are general purpose digital I/O ports. All modules can make thes pins accessible.
- CH_PD: This pin is used for Chip enables.
- RST: Reset. Keep it on high (3.3V) for normal operation. Put it on 0V to reset the chip.

3.2.4 Interfacing with Arduino UNO

- Tx <-> Rx
- Rx <-> Arduino's Tx (But needs a voltage adjusting)
- GND <-> Arduino's ground
- VCC <-> LF33 OR LM317 pin 3.3 Vout
- CH_PD <-> LF33 or LM317 pin 3.3 Vout

Warning!

- The ESP8266 does not have tolerant inputs so you could destroy your Wi-Fi module.
- The ESP8266 needs 3.3V to operate so when connecting it to the computer you should use a voltage controller from 5V to 3.3V.

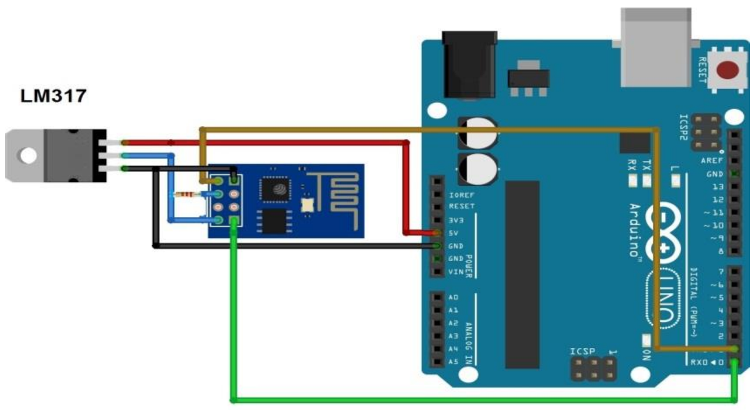


Fig.13: Arduino UNO interfacing with ESP8266

1. Plug in the Wi-Fi module
2. Chose correct serial port from the Tools> Serial Port menu.
3. Open Serial monitor via the menu or magnifying glass icon on the editor window.
4. For the default firmware version, Both NL & CR is selected in the line ending pop-up menu at the bottom of the serial monitor.
5. For the default firmware version, ensure the communication speed is set to 115200 baud. Change it to 9600 baud by using command “AT+CIOBAUD=9600”

You’re now ready to send your first command!

3.3 Acting as a Wi-Fi Access Point:

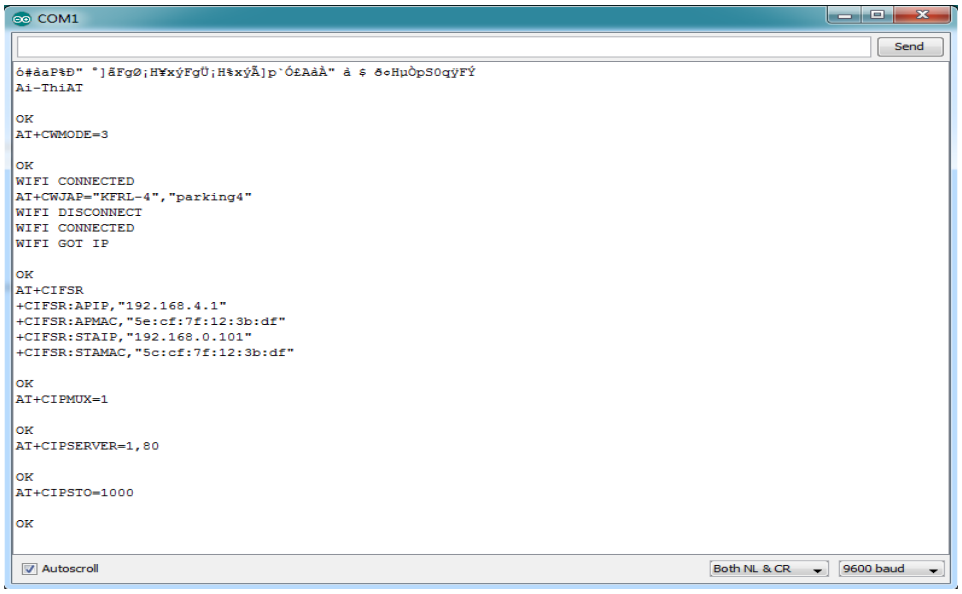
In addition to connecting to Wi-Fi Access Points the module can also act as an Access Point. This means you can connect devices to the module without any other network infrastructure in place.

Program:

```
#include <SoftwareSerial.h>
#define DEBUG true
int connectionId;
SoftwareSerial esp8266(7,6);
String espmode = "3";
String SSid = "KFRL-4";                    //Enter your SSID here
String password = "parking4";               //Enter your password
void wifisetup(String mode,String ssid,String pass);
void st_server(String H1,String Data,String H1color);

void setup()
{
  pinMode(12, OUTPUT);
  Serial.begin(9600);
  esp8266.begin(9600); // your esp's baud rate might be different
  digitalWrite(10, LOW);
  delay(1000);
  wifisetup(espmode,SSid,password);
   delay(1000);
   digitalWrite(10, HIGH);
}
String Heading = "Internet Of Things";
String Colour = "blue";
String DataSend = "Hello World";

void loop()
{
 digitalWrite(10, LOW);
 delay(1000);
st_server(Heading,DataSend,Colour);
delay(1000);
digitalWrite(10, HIGH);
}

//Funtions
//SentData Funtion
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())
{
char c = esp8266.read(); // read the next character.
response+=c;
}  }
 if(debug)
    {
      Serial.print(response);
    }
    return response;
}
//Connectvity Function
  void wifisetup(String mode,String ssid,String pass)
{
  sendData("AT\r\n",2000,DEBUG); // reset module
 String mode0= "AT+CWMODE=";
 mode0 += mode;
 mode0 += "\r\n";
  sendData(mode0,2000,DEBUG);
  String ssidset="AT+CWJAP=\"";
  ssidset += ssid ;
  ssidset += "\",\"" ;
  ssidset += pass ;
  ssidset += "\"" ;
  ssidset += "\r\n" ;
  sendData(ssidset,10000,DEBUG);
  delay(900);
  sendData("AT+CIFSR\r\n",4000,DEBUG); // get ip address
  sendData("AT+CIPMUX=1\r\n",2000,DEBUG); // configure for multiple connections
  sendData("AT+CIPSERVER=1,80\r\n",2000,DEBUG); // turn on server on port 80
  sendData("AT+CIPSTO=1000\r\n",1000,DEBUG);
return ;}

//Sending Data
void st_server(String H1,String Data,String H1color){
if(esp8266.available()){
if(esp8266.find("+IPD,")){
{
delay(1000);
int connectionId = esp8266.read()-48; // subtract 48 because the read() function returns
                                        // the ASCII decimal value and 0 (the first decimal number) starts at 48
 String webpage = "<head></head><body bgcolor=\"";
webpage += H1color;
webpage += "\"><h1>";
webpage += H1;
webpage+= "</h1>";
String cipSend = "AT+CIPSEND=";
cipSend += connectionId;
cipSend += ",";
cipSend +=webpage.length();
cipSend +="\r\n";
sendData(cipSend,2000,DEBUG);
sendData(webpage,2000,DEBUG);
String Data1 = Data;
cipSend = "AT+CIPSEND=";
cipSend += connectionId;
cipSend += ",";
cipSend +=Data1.length();
cipSend +="\r\n";
sendData(cipSend,1000,DEBUG);
sendData(Data1,1000,DEBUG);
 }
 String closeCommand = "AT+CIPCLOSE=";
closeCommand+=connectionId; // append connection id
closeCommand+="\r\n";
 sendData(closeCommand,3000,DEBUG);
 return;}}}
```

Serial Monitor Output:



```
ó#àaP$Ð" °]ãFgØ¡H¥xýFgÜ¡H$xýÃ]p`Ó£AàÀ" à $ ð∘HµÒpS0qÿFÝ
Ai-ThiAT

OK
AT+CWMODE=3

OK
WIFI CONNECTED
AT+CWJAP="KFRL-4","parking4"
WIFI DISCONNECT
WIFI CONNECTED
WIFI GOT IP

OK
AT+CIFSR
+CIFSR:APIP,"192.168.4.1"
+CIFSR:APMAC,"5e:cf:7f:12:3b:df"
+CIFSR:STAIP,"192.168.0.101"
+CIFSR:STAMAC,"5c:cf:7f:12:3b:df"

OK
AT+CIPMUX=1

OK
AT+CIPSERVER=1,80

OK
AT+CIPSTO=1000

OK
```

AT Commands:

Table.2: AT commands sequence

Function	AT Command	Response
Restart	AT+RST	OK
Wi-Fi Mode	AT+CWMODE=3	BOTH
Join Access Point	AT+CWJAP=”SSID”,”Password”	
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105
TCP/UDP Connection	AT+CIPMUX=1	Allow Multiple Connections
Set as Server	AT+CIPSERVER=<mode>[<port>]	Mode 0 to close server mode ; mode 1 to open; port=port
Set the Server timeout	AT+CIPSTO=<time>	<time0~28800 in seconds
Send TCP/IP data	(CIPMUX=1)AT+CIPSEND=<id>,<length>	
Close TCP/IP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	

Task:

1. Write a program to send data of interfaced sensors by using Wi-Fi module as an access point.
2. Analyse data of sensors and plot it on MS-Excel.

CHAPTER: 4
WEATHER STATION

4.1 TCP CONNECTION:

TCP is a connection-oriented protocol. It enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

4.1.1 ACTING AS A TCP CLIENT:

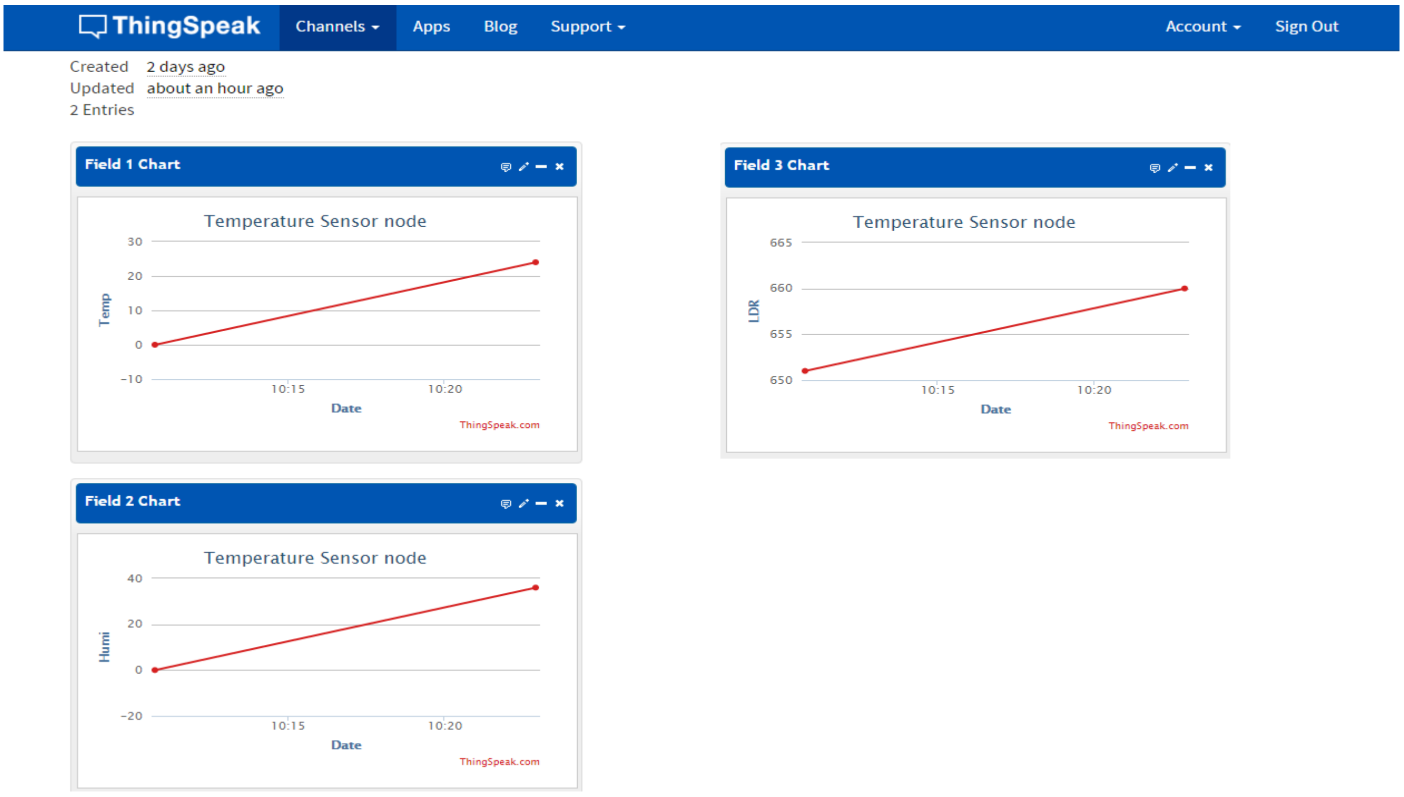
You can also connect to an internet server (such as web server).
Change your SSID, password and API key of ‘ThingSpeak’ website in the below program.

```
#include <SoftwareSerial.h>
SoftwareSerial esp8266(7, 6);
#include <dht.h>
dht DHT;
#define dht_dpin 4
int sensorPin = A0;
#define DEBUG true
#include <avr/wdt.h>
#include <avr/sleep.h>
String espmode = "3";
String SSid = "KFRL-4";                                    //Enter your SSID here
String password = "parking4";                              //Enter your password here
String serverip = "184.106.153.149";
String apikeey = "Q2S6X3TD0NB8RGVN";       //Enter your API key here

void wifisetup(String mode,String ssid,String pass);
void wifidata(float tempe,float hume,int ldrval,String s_ip,String apikey);

void setup()
{
  wdt_enable (WDTO_8S);
  Serial.begin(9600);
  esp8266.begin(9600); // your esp's baud rate might be different
 wdt_reset ();
pinMode(10, OUTPUT);
}

void loop()
{
  digitalWrite(10, LOW);
 myWatchdogEnable (0b100001);  // 8 seconds
  wdt_reset ();

 wifisetup(espmode,SSid,password);

 DHT.read11(dht_dpin);
float hum=DHT.humidity;
float tem=DHT.temperature;
wdt_reset ();
```

```
int ldr = analogRead(sensorPin);

wifidata(tem,hum,ldr,serverip,apikeey);

digitalWrite(10, HIGH);
myWatchdogEnable (0b100001);  // 8 seconds

for(int a=0;a<=75;a++){
myWatchdogEnable (0b100001);  // 8 seconds
}
  myWatchdogEnable (0b100001);  // 8 seconds
  myWatchdogEnable (0b100000);  // 4 seconds
}

//Functions

//SendData Function
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();
 wdt_reset ();
while( (time+timeout) > millis())
{
   wdt_reset ();
while(esp8266.available())
{
   wdt_reset ();
char c = esp8266.read(); // read the next character.
response+=c;
}
}
 if(debug)
{
   wdt_reset ();
      Serial.print(response);
}
     wdt_reset ();
    return response;
}

//WDT Vector Function
ISR(WDT_vect)
 {
 wdt_disable();  // disable watchdog
 }

//Watchdog Function
void myWatchdogEnable(const byte interval)
  {
  MCUSR = 0;                          // reset various flags
  WDTCSR |= 0b00011000;               // see docs, set WDCE, WDE
  WDTCSR =  0b01000000 | interval;    // set WDIE, and appropriate delay
  wdt_reset();
  set_sleep_mode (SLEEP_MODE_PWR_DOWN);
  sleep_mode();           // now goes to Sleep and waits for the interrupt
  }


  //Wifi Connectivity Function
  void wifisetup(String mode,String ssid,String pass)
{

 wdt_reset ();
  sendData("AT\r\n",2000,DEBUG); // reset module
 String mode0= "AT+CWMODE=";
 mode0 += mode;
 mode0 += "\r\n";
  wdt_reset ();
  sendData(mode0,2000,DEBUG); // configure as access point
  wdt_reset ();
  String ssidset="AT+CWJAP=\"";
  ssidset += ssid ;
  ssidset += "\",\"" ;
  ssidset += pass ;
  ssidset += "\"" ;
  ssidset += "\r\n" ;
  sendData(ssidset,10000,DEBUG);
wdt_reset ();
return ;
}

 //Data Sending Function
void wifidata(float temp,float hum,int ldrval,String s_ip,String apikey){
wdt_reset ();
char buffer[10];
  String HumiF = dtostrf(hum, 4, 1, buffer);
  String tempF = dtostrf(temp, 4, 1, buffer);
  String LDRval = dtostrf(ldrval,4,1,buffer);
   wdt_reset ();
   String api = "GET /update?key=" ;
   api += apikey;
String Data1 = api+'&'+"field1="+tempF+'&'+"field2="+HumiF+'&'+"field3="+LDRval;
Data1 +="\r\n";
 wdt_reset ();
 String scnt = "AT+CIPSTART=\"TCP\",\"" ;
 scnt += s_ip ;
 scnt += "\",80\r\n" ;
sendData(scnt,1000,DEBUG);
String cipSend = "AT+CIPSEND=";
 wdt_reset ();
cipSend +=Data1.length();
cipSend +="\r\n";
 wdt_reset ();
sendData(cipSend,2000,DEBUG);

Serial.println(Data1);
sendData(Data1,5000,DEBUG);
```

```
 wdt_reset ();
String closeCommand = "AT+CIPCLOSE";
closeCommand+="\r\n";
sendData(closeCommand,1000,DEBUG);
wdt_reset ();
return ;
}
```

OUTPUT:



Created 2 days ago
Updated about an hour ago
2 Entries

AT COMMANDS:

Table.3: AT commands sequence for TCP client

FUNCTION	AT COMMAND	RESPONSE
Working	AT	OK
Wi-Fi Mode	AT+CWMODE=3	BOTH
Join Access Point	AT+CWJAP=”SSID”,”Password”	
Close TCP/IP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	AT+CIPCLOSE=<id> or AT+CIPCLOSE
Setup TCP/UDP connection	(CIPMUX=1) AT+CIPSTART=<id><type>,<addr>,<port>	Query Id=0-4,type=TCP/UDP,addr=IP address,port=port
Send TCP/IP data	(CIPMUX=1)AT+CIPSEND=<id>,<length>	

4.2 CLOUD STORAGE:

Cloud storage is a model of data storage where the digital data is stored in logical pools, the physical storage spans multiple and the physical environment is typically owned and managed by a hosting company.

These cloud storage providers are responsible for keeping the data available and accessible, and the physical environment protected and running. People and organizations buy or lease storage capacity from the providers to store user, organization, or application data.

Cloud storage services may be accessed through a co-located cloud computer service, a web service application programming interface (API) or by applications that utilize the API, such as cloud desktop storage, a cloud storage gateway or Web-based content management systems.

Fig.14 Cloud Storage

4.2.1 CLOUD (THINGSPEAK):

Cloud storage we use in workshop is "ThingSpeak".

"ThingSpeak" is an application platform for the Internet of Things. Using "ThingSpeak" you can build an application around data collected by sensors.
Using "ThingSpeak" you can:

- Create a Channel and collect data from things.
- Analyze and visualize your data.
- Act on your data using Apps.

4.2.1.1 FEATURES:

- Time series data collection
- Data analytics using MATLAB®
- MATLAB visualizations
- Apps
- Plug-in

Each channel has:

- Eight fields that can hold any type of data.
- Three location fields.
- One status field.

After you create a ThingSpeak channel you can publish data to the channel, have ThingSpeak process the data, and then have your application retrieve the data.

4.2.2 Thing Speak Quick Start:

1. Sign Up for a ThingSpeak account " https://thingspeak.com/ "

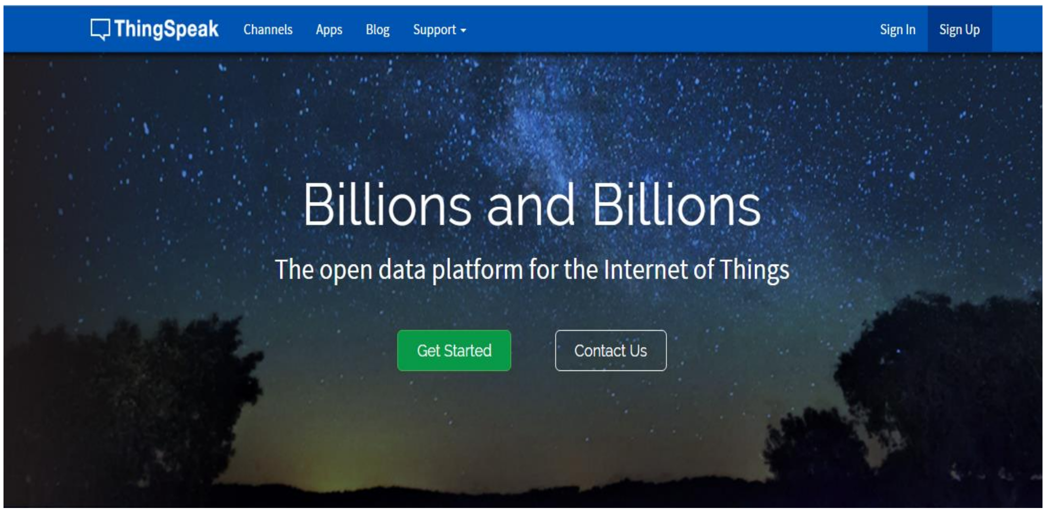
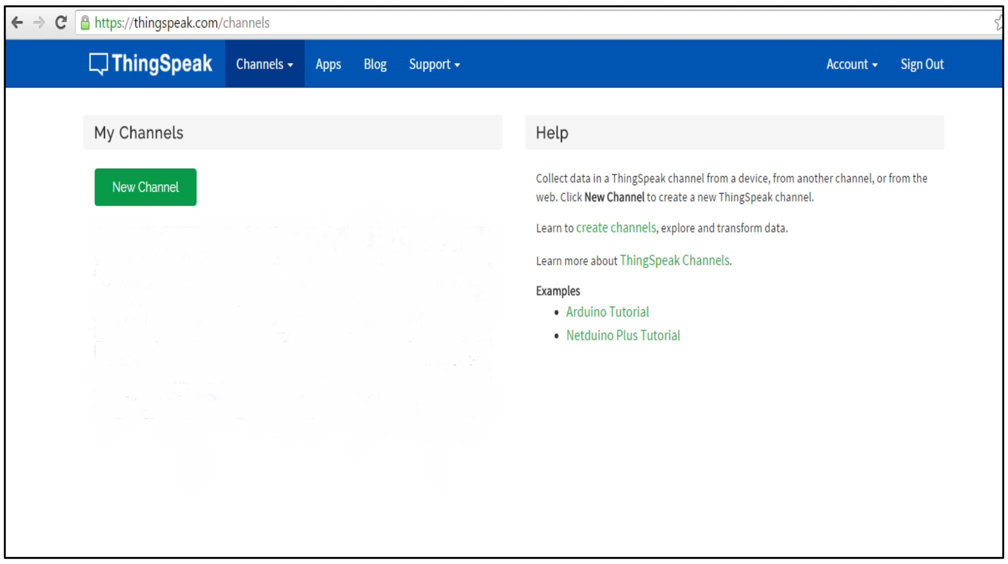


Fig.14: Home page

2. Go to Channels, and click New Channel



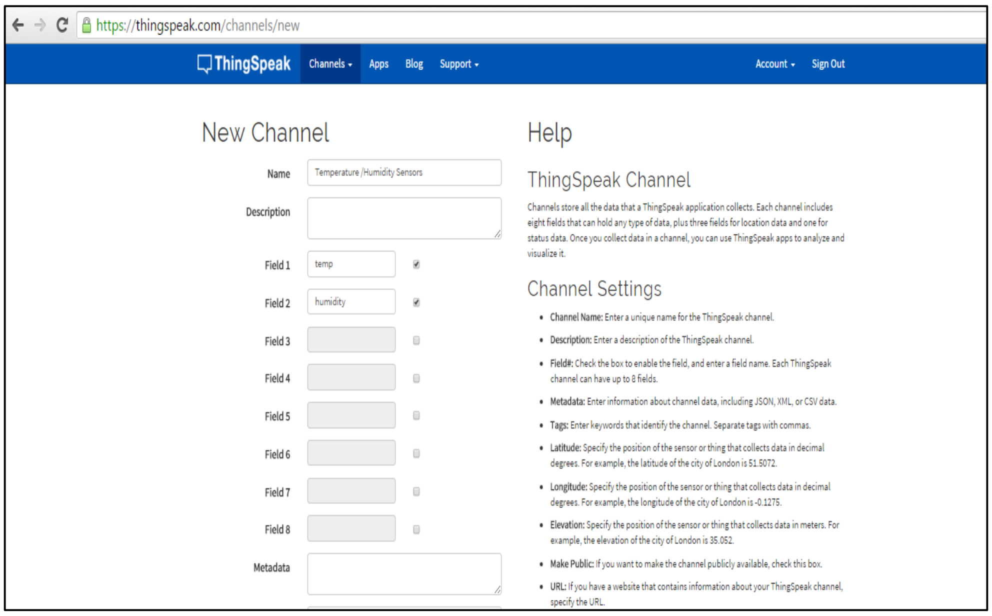
Collect data in a ThingSpeak channel from a device, from another channel, or from the web. Click **New Channel** to create a new ThingSpeak channel.

Learn to create channels, explore and transform data.

Learn more about ThingSpeak Channels.

Fig.15: Create channel

3. Complete the channel settings, and click Save Channel.



New Channel

Name	Temperature /Humidity Sensors	
Description		
Field 1	temp	✓
Field 2	humidity	✓
Field 3		☐
Field 4		☐
Field 5		☐
Field 6		☐
Field 7		☐
Field 8		☐
Metadata		

Help

ThingSpeak Channel

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Latitude:** Specify the position of the sensor or thing that collects data in decimal degrees. For example, the latitude of the city of London is 51.5072.
- **Longitude:** Specify the position of the sensor or thing that collects data in decimal degrees. For example, the longitude of the city of London is -0.1275.
- **Elevation:** Specify the position of the sensor or thing that collects data in meters. For example, the elevation of the city of London is 35.052.
- **Make Public:** If you want to make the channel publicly available, check this box.
- **URL:** If you have a website that contains information about your ThingSpeak channel, specify the URL.

Fig.16: Channel setting

4. View your Channel feed:
   https://api.thingspeak.com/channels/YOUR_CHANNEL_ID/feeds.json

TASK:

1. Write a program to send data of sensors on web server & analyse graph.

Chapter: 5
IoT Enabled Kit

The Internet of Things is an evolution of the Internet. It interconnects not only human and computer as before but also physical objects (things).The IoT Kit is a solution comprised of the software and hardware options that empower you to create innovative IoT projects. It allows to control many devices over the Internet. With additional modules you can extend the kit.

This kit is created by KFRL team which includes of:

- Arduino UNO board
- Wi-Fi module
- Temperature/humidity sensor
- LDR
- DIP Switches
- Relays
- LEDs

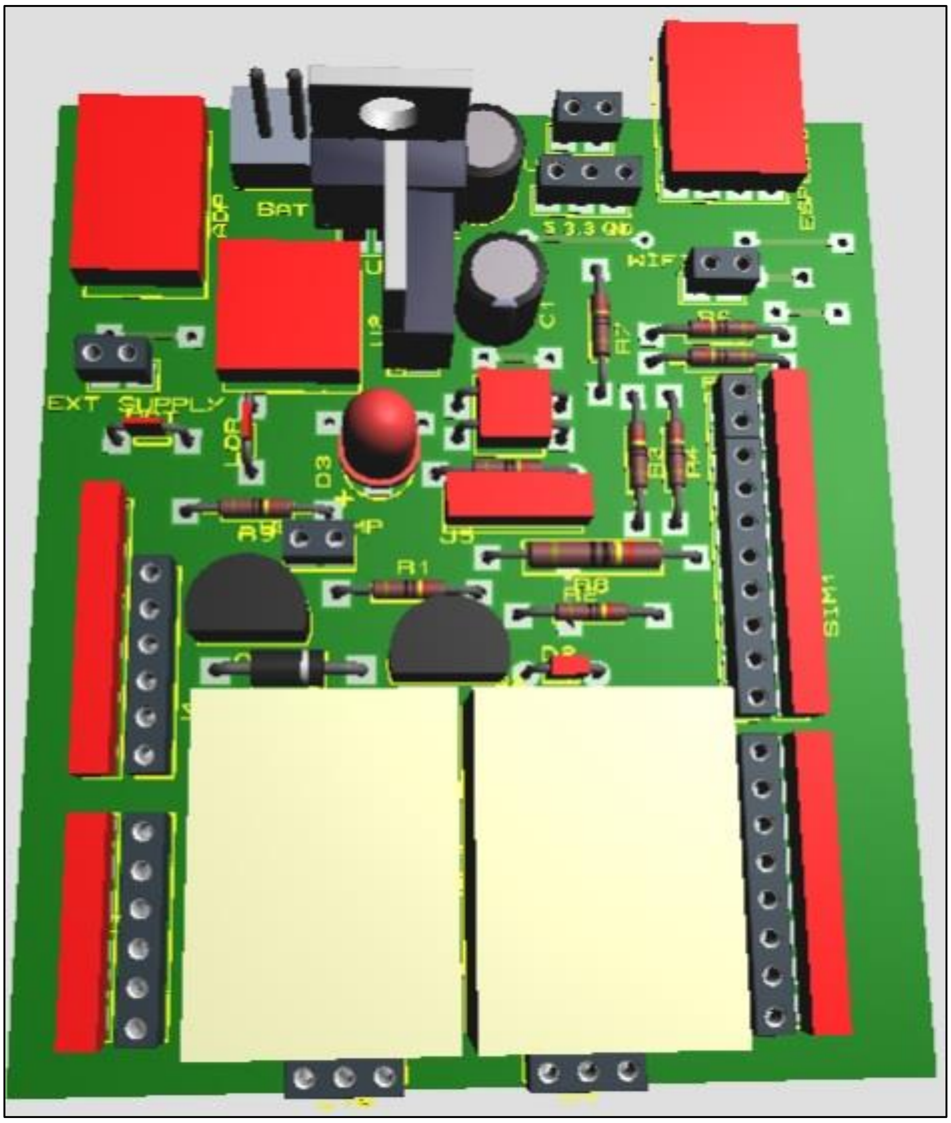


Fig. 17 3D IoT Kit

5.1 FEATURES:

- Low cost
- Long-term stability
- Fast Response
- Simple structure.
- Relative humidity and temperature measurement
- Light intensity measurement
- Transmission of data on cloud station
- Online data monitoring

5.2 PIN CONFIGURATION:

TABLE: 4 INPUT/OUTPUT PINS

INPUT PINS (ARDUINO UNO)	OUTPUT PINS
A0	LDR
D4	Temperature/Humidity sensor
D6	TX (Wi-Fi)
D7	RX (Wi-Fi)
D8	Relay (left side)
D9	Relay (right side)
D10	DIP switch
D11	DIP switch
D12	LED

5.3 APPLICATIONS

The applications of this kit are as follows:

- Climate control:

  This device can not only measure temperature or humidity it also can turn on and off different equipment to sustain climate in your room. You can control the speed of a DC fan according to the temperature read by a DHT11 sensor.

- Metrological:

  Humidity and temperature play an important role in meteorology. Monitoring of the external temperature and relative humidity is essential for the smooth running of wind energy systems. For demanding applications like the requirement to recognize the danger of icing up, special solutions like heated humidity sensors have to be used.

- Agriculture:

  The precise measurement and control of humidity and temperature plays a central role in cold storage. For the correct storage of fruit it is important that the metabolic activity of the fruit and the decomposition value slowed down by low temperatures. A constantly high humidity prevents the fruit from drying out.

- Industrial Measurement:
    - To achieve optimum product quality, both the temperature and the humidity are controlled in the drying process.
    - Measure the absolute humidity in blast furnace gas that is necessary to determine its heating value for further use.

- Greenhouse Monitoring:

  It is used in green houses to control the temperature, humidity and light for the proper growth of plants.

- Automatic light control:

  It is used to track the sunlight and when the sensors goes dark the led will be made ON and when the sensors found light the led will be made OFF.

5.4 FUTURE DEVELOPMENT:

With little modification,

- You can monitor more parameters like Humidity, PH, pressure, water level and so on.
- You can send data using mobile app or user app.
- Two way communication can also be possible

TASK:

Extend IoT kit to monitor temperature of the microcontroller IC and also write a programme to enable it.