CMR INSTITUTE OF TECHNOLOGY

UGC AUTONOMOUS

Approved by AICTE, Permanent Affiliation to JNTUH & NBA Accredited

Kandlakoya (V), Medchal Dist-501 401

www.cmritonline.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IOT WITH CLOUD COMPUTING LAB MANUAL

Prepared by

S PARAMESH

Assistant Professor, Dept. of CSE

CMR INSTITUTE OF TECHNOLOGY

Kandlakoya(V), Medchal Road, Hyderabad – 501 401

Ph. No. 08418-222042, 22106 Fax No. 08418-222106

2024-25

IOT and Cloud Computing Lab

III-B.Tech.-II-Sem.	**L T P C**
Subject Code: 22CSPC64	**0 0 2 1**

Course Outcomes: Upon completion of the course, the students will be able to

1. Identify various IoT devices.
2. Use IoT Devices in various applications
3. Develop automation work-flow in IoT enabled cloud environment
4. Take part in practicing and monitoring remotely
5. Make use of various IoT Protocols in Cloud

List of Experiments (Minimum **10** experiments to be conducted**)**

1. Install necessary software for Arduino and Raspberry Pi.
2. Familiarization with Arduino and Raspberry Pi boards.
3. Write a program to transfer sensor data to smart phone using Bluetooth on Arduino.
4. Write a program to implement RFID using Arduino.
5. Write a program to monitor temperature and humidity using Arduino and Raspberry Pi.
6. Write a program to interface IR Sensor with Arduino using IoT cloud application.
7. Write a program to upload temperature and humidity data to the cloud using Arduino or Raspberry Pi.
8. Write a program to retrieve temperature and humidity data to the cloud using Arduino or Raspberry Pi.
9. Write a program to create TCP Server on cloud using Arduino and Respond with humidity data to TCP Client when requested.
10. Write a program to create UDP Server on cloud using Arduino and Respond with humidity data to UDP Client when requested.

Reference:

1. Internet of Things Lab Manual, Department of CSE, CMRIT, Hyd.

Micro-Projects: Student must submit a report on one of the following Micro–Projects before commencement of second internal examination.

1. Air Pollution Meter
2. Smart Garbage Collector
3. IoT Based Smart street light
4. Humidity & Temperature Monitoring
5. Baggage Tracker
6. IoT based Gas leakage Monitoring
7. Circuit Breakage Detection
8. IoT based Smart irrigation system
9. Anti-Theft Flooring System
10. IoT based water level Monitoring system

WEEK-1
Arduino Installation

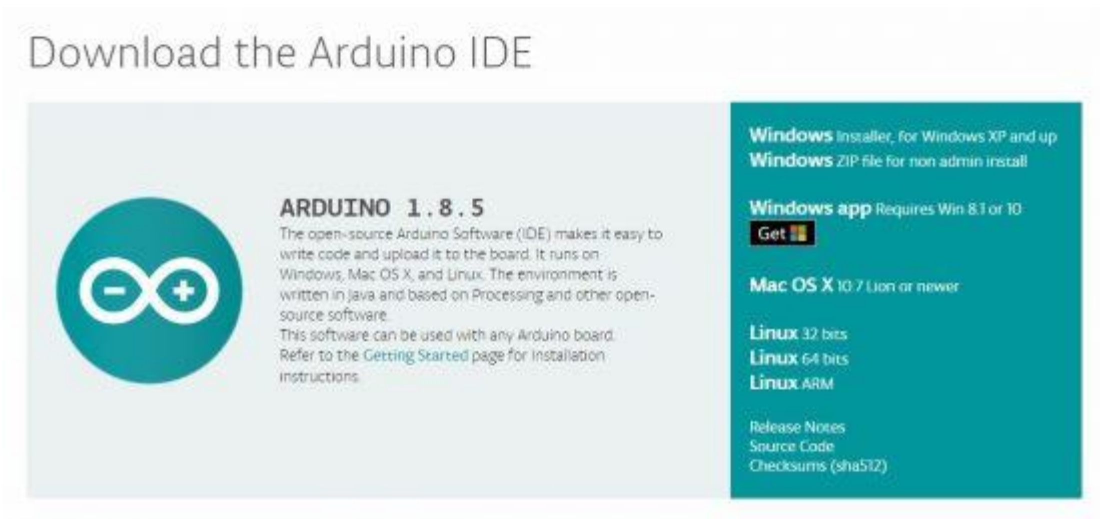
1. **Aim:** Install necessary software for Arduino and Raspberry Pi.

Arduino:

- Arduino is a platform that makes it easy for you to build projects using electronics.
- IoT is a way of using electronics - to make electronic modules talk to each other remotely and wirelessly (often using a Cloud) to solve problems.
- Now, Arduino can also help you easily build IoT projects in two ways: Using traditional Arduino boards and attaching communication breakout modules (like nRF Bluetooth, WiFi, LoRA, GSM, etc) to them.
- Arduino is a micro controller that can be connected to one or more sensors and help you capture the data or information and then pass it on to processor. If you know the full stack of IoT then you should also look at Raspberry.
- RaspPi is a microprocessor so the basic difference between Arduino and RasPi is that RaspPi is controller plus processor and Arduino is just a micro controller.
- They suit the need for different use cases. You can easily read online about this both.

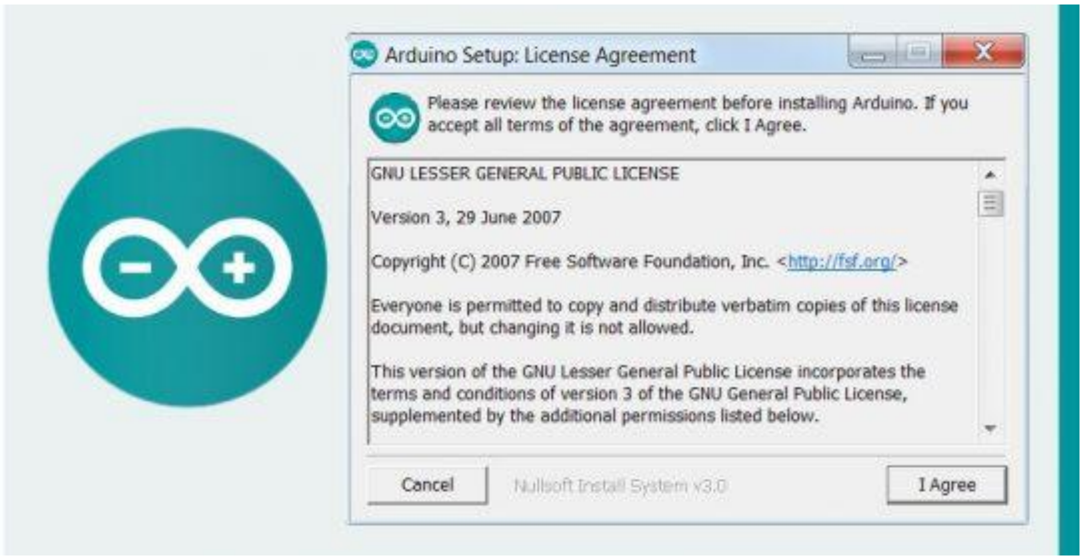
Download and install the Arduino software (Arduino IDE 1.8.5)

- Go to the Arduino website and click the download link to go to the download page.
- After downloading, locate the downloaded file on the computer and extract the folder from the downloaded zipped file. Copy the folder to a suitable place such as your desktop.


Download the Arduino IDE

ARDUINO 1.8.5
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install
Windows app Requires Win 8.1 or 10
Get
Mac OS X 10.7 Lion or newer
Linux 32 bits
Linux 64 bits
Linux ARM
Release Notes
Source Code
Checksums (sha512)



Please review the license agreement before installing Arduino. If you accept all terms of the agreement, click I Agree.

GNU LESSER GENERAL PUBLIC LICENSE

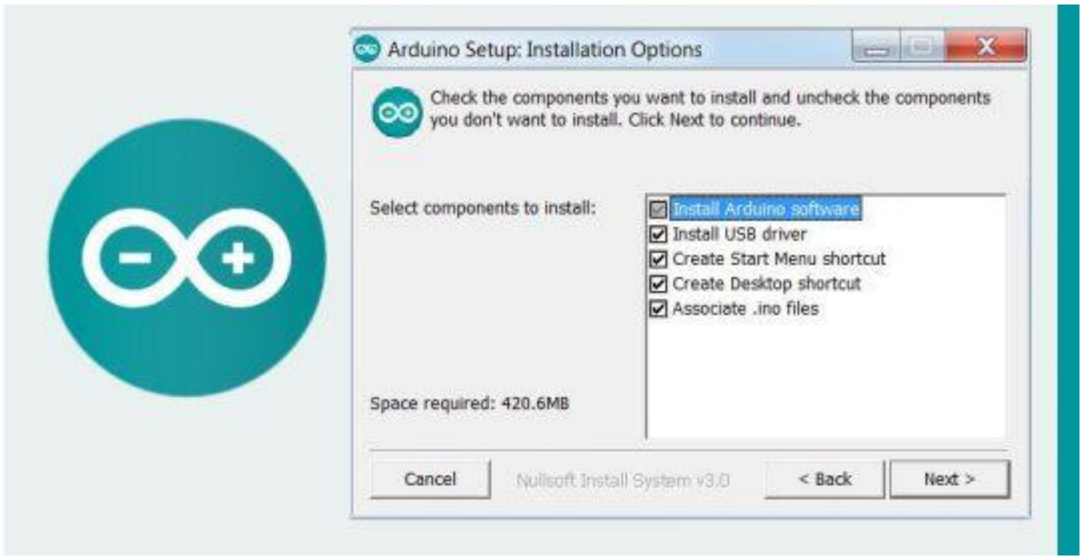
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

Read the Arduino License agreement and click the “I Agree” button.


Arduino Setup: Installation Options

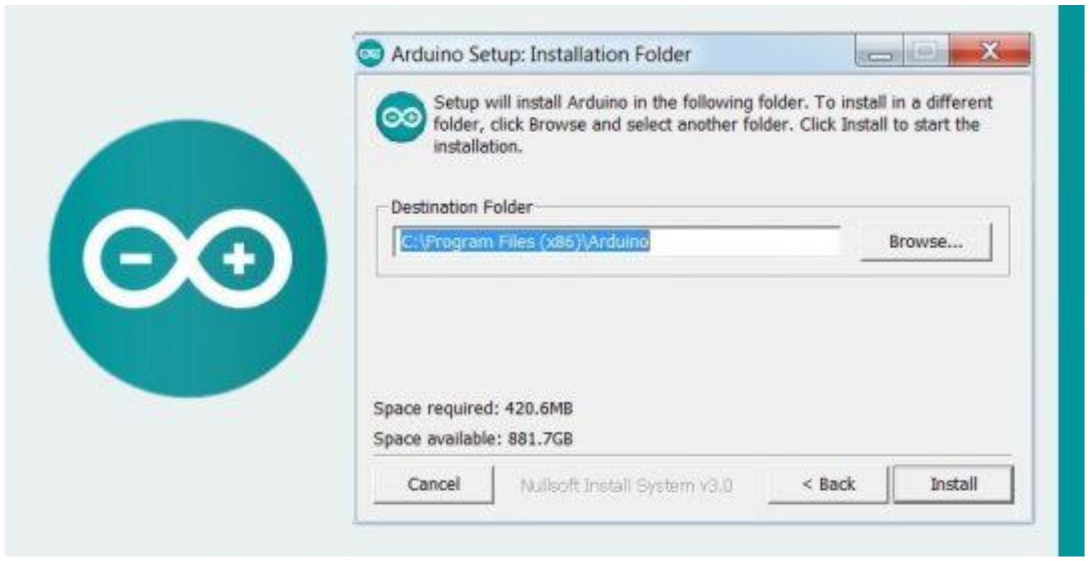
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

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

Space required: 420.6MB

Cancel
Nullsoft Install System v3.0
< Back
Next >

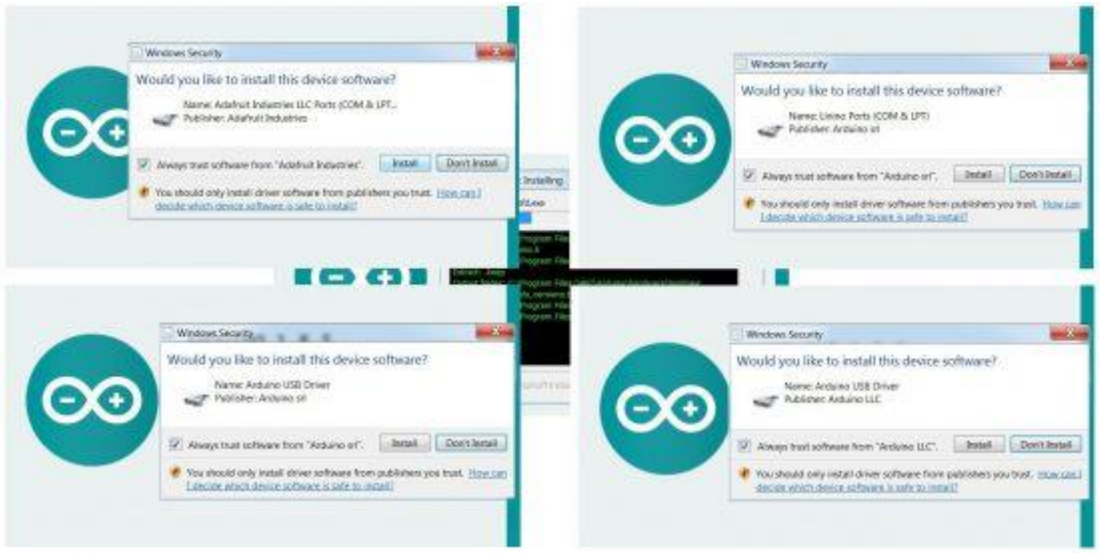

Arduino Setup: Installation Folder

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

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

Space required: 420.6MB
Space available: 881.7GB

The Arduino software will start to install.


Windows Security

Would you like to install this device software?

Name: Adafruit Industries LLC Ports (COM & LPT...
Publisher: Adafruit Industries

Always trust software from "Adafruit Industries".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Windows Security

Would you like to install this device software?

Name: Linino Ports (COM & LPT)
Publisher: Arduino srl

Always trust software from "Arduino srl".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Windows Security

Would you like to install this device software?

Name: Arduino USB Driver
Publisher: Arduino srl

Always trust software from "Arduino srl".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Windows Security

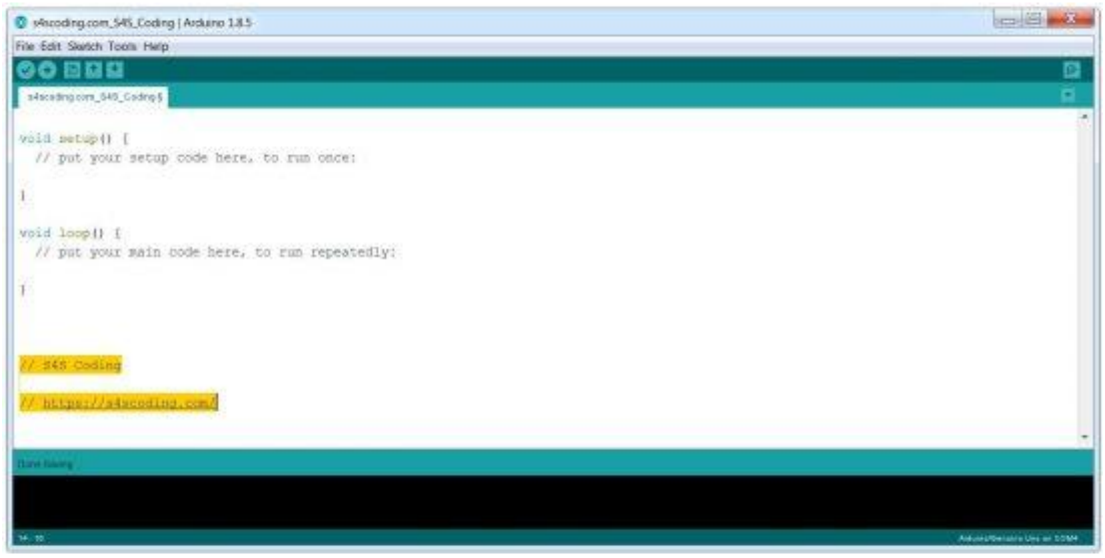
Would you like to install this device software?

Name: Arduino USB Driver
Publisher: Arduino LLC

Always trust software from "Arduino LLC".

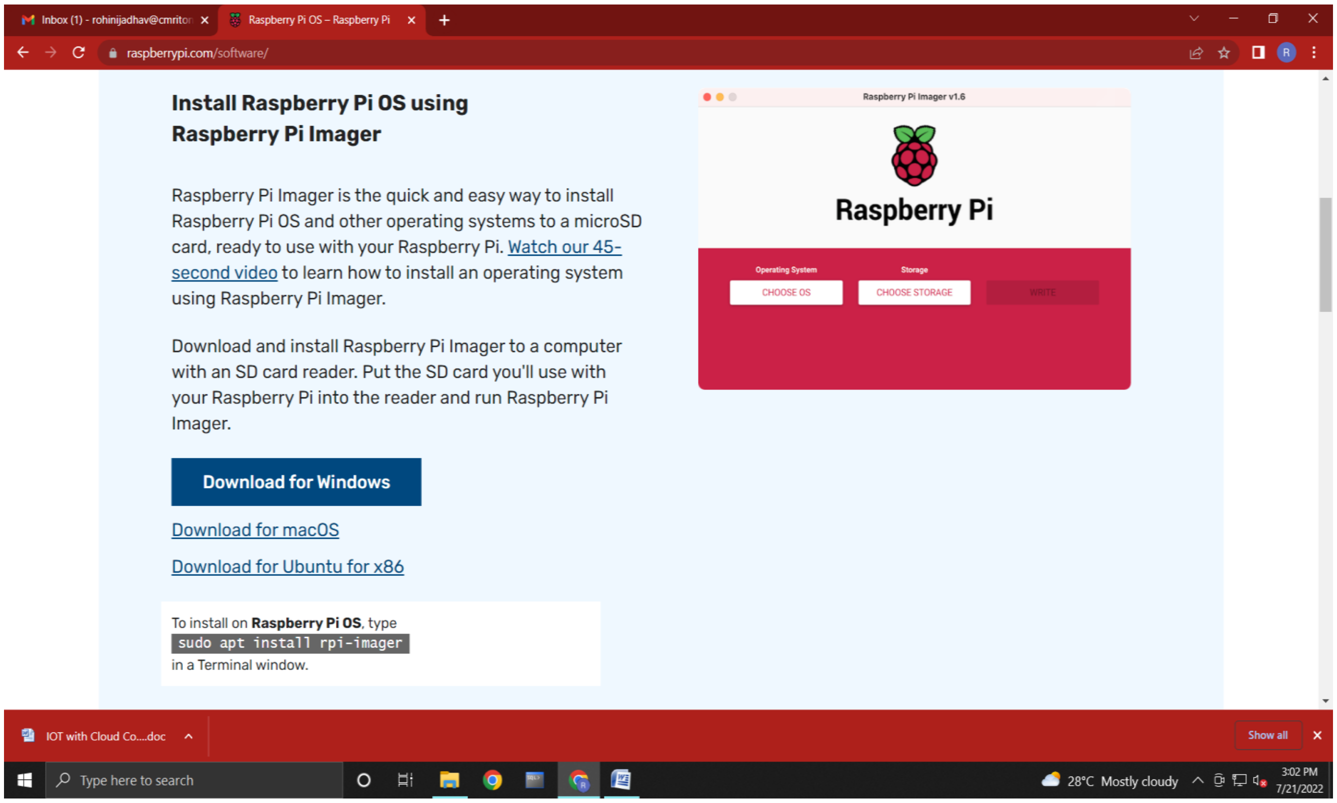
You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Running the Arduino IDE Software


s4scoding.com_S4S_Coding | Arduino 1.8.5
File Edit Sketch Tools Help
s4scoding.com_S4S_Coding §
Done Saving.

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

}

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

}



// S4S Coding

// https://s4scoding.com/
```

Raspberry Pi



Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. Watch our 45-second video to learn how to install an operating system using Raspberry Pi Imager.

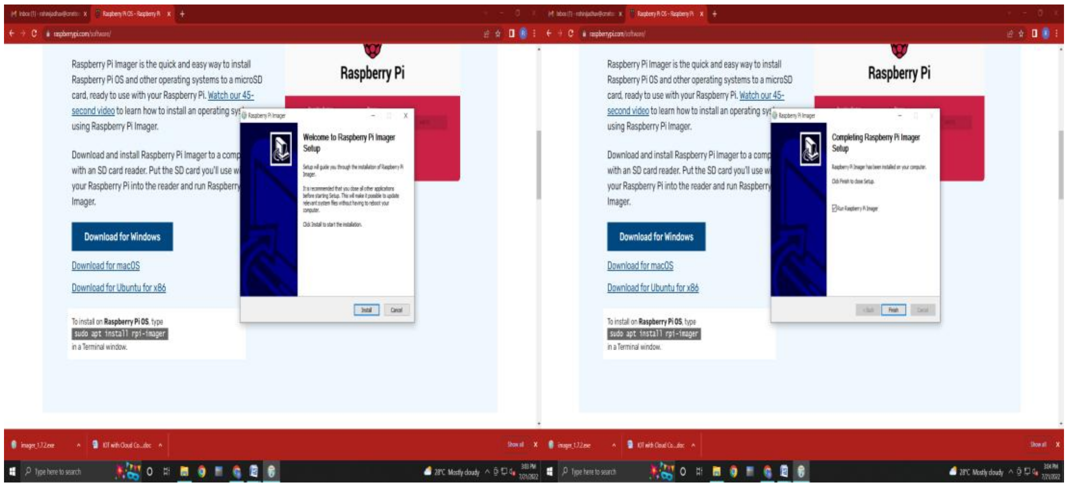
Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

Download for Windows

Download for macOS

Download for Ubuntu for x86

To install on **Raspberry Pi OS**, type
`sudo apt install rpi-imager`
in a Terminal window.



Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. Watch our 45-second video to learn how to install an operating sys using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a comp with an SD card reader. Put the SD card you'll use w your Raspberry Pi into the reader and run Raspberry Imager.

Download for Windows

Download for macOS

Download for Ubuntu for x86

To install on **Raspberry Pi OS**, type

```
sudo apt install rpi-imager
```

in a Terminal window.

Welcome to Raspberry Pi Imager Setup

Setup will guide you through the installation of Raspberry Pi Imager.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

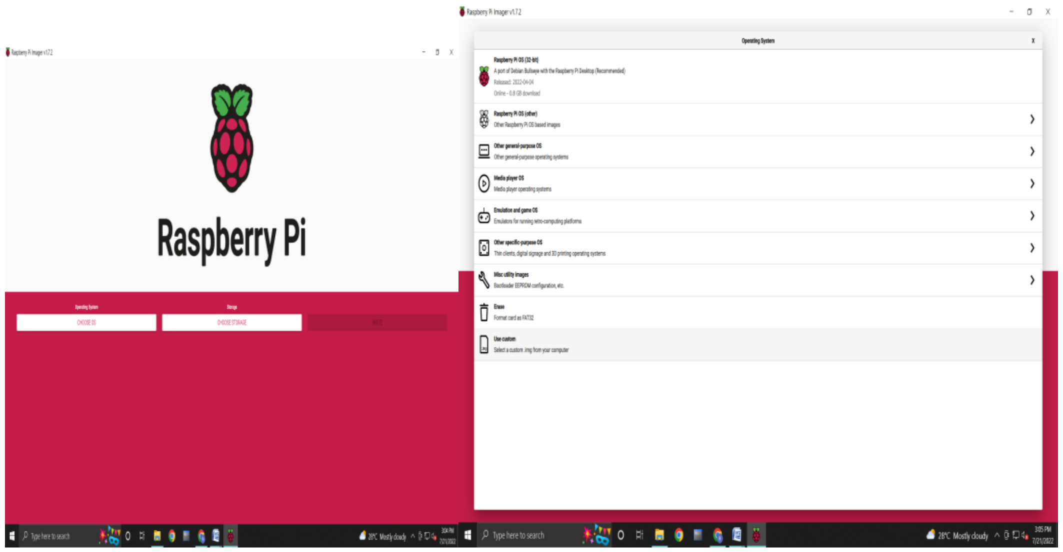
Click Install to start the installation.

Completing Raspberry Pi Imager Setup

Raspberry Pi Imager has been installed on your computer.

Click Finish to close Setup.

☑ Run Raspberry Pi Imager



Raspberry Pi Imager v1.7.2

Operating System

Raspberry Pi OS (32-bit)
A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended)
Released: 2022-04-04
Online - 0.8 GB download

Raspberry Pi OS (other)
Other Raspberry Pi OS based images

Other general-purpose OS
Other general-purpose operating systems

Media player OS
Media player operating systems

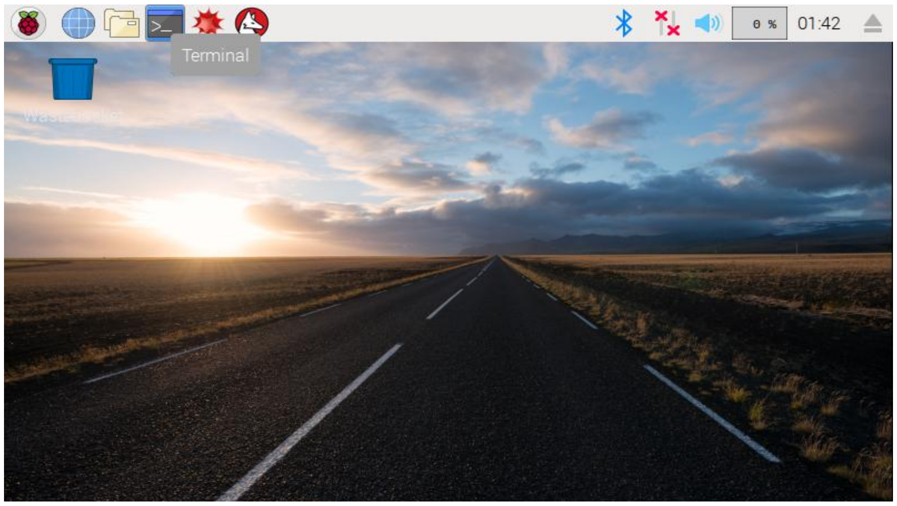
Emulation and game OS
Emulators for running retro-computing platforms

Other specific-purpose OS
Thin clients, digital signage and 3D printing operating systems

Misc utility images
Bootloader EEPROM configuration, etc.

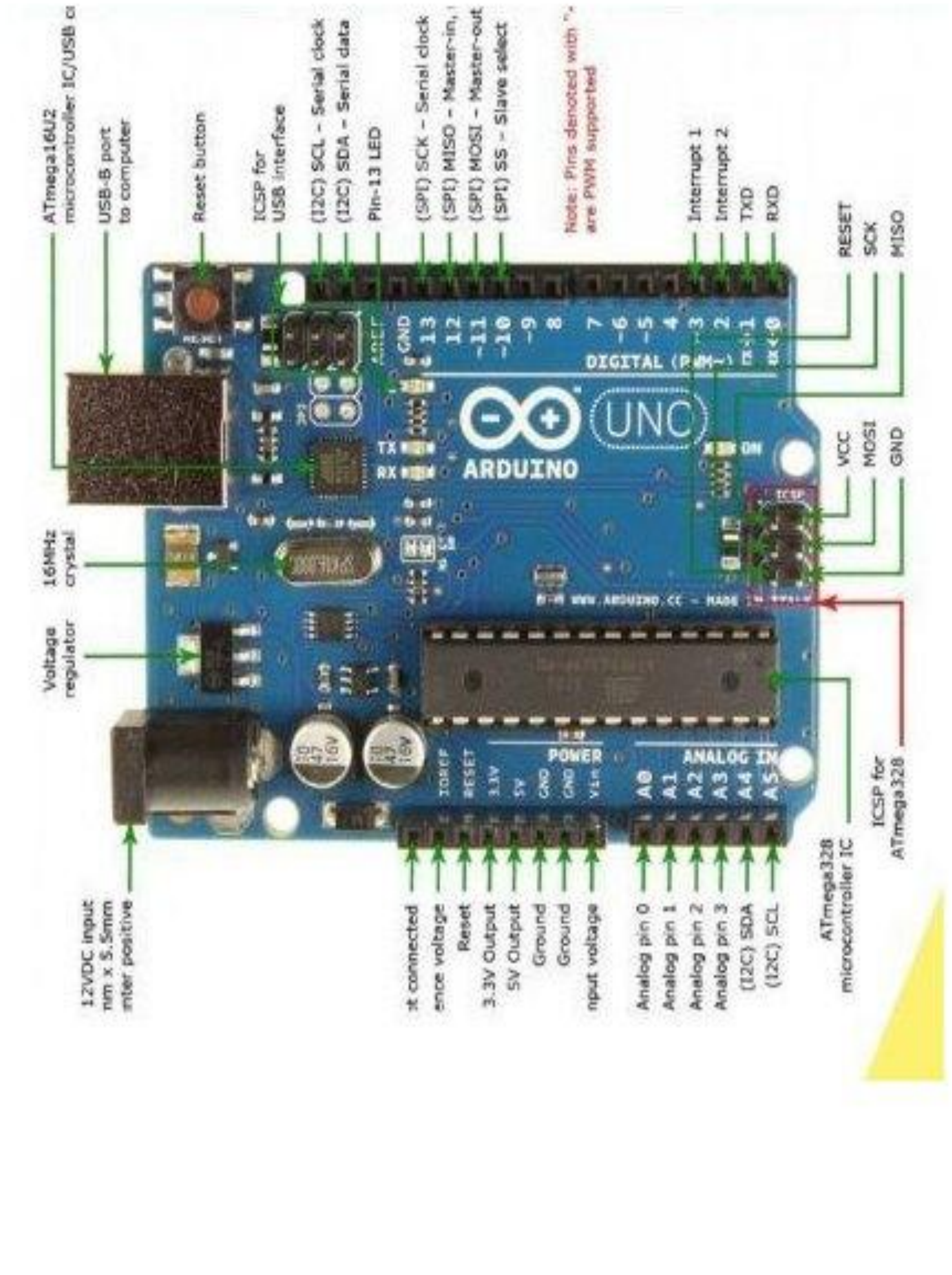
Erase
Format card as FAT32

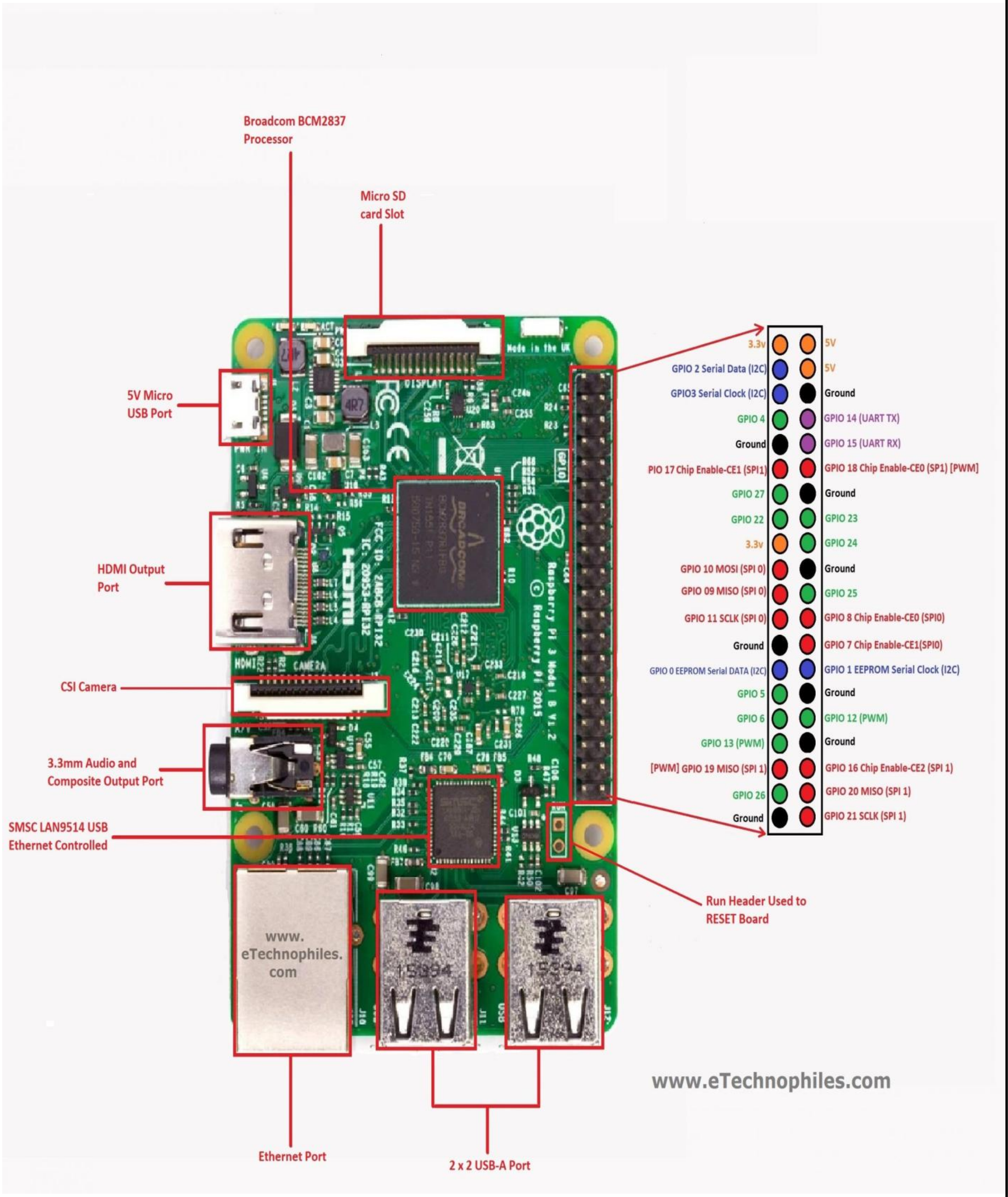
Use custom
Select a custom .img from your computer



WEEK-2

AIM: Familiarization with Arduino and Raspberry Pi boards.




Broadcom BCM2837 Processor
Micro SD card Slot
5V Micro USB Port
HDMI Output Port
CSI Camera
3.3mm Audio and Composite Output Port
SMSC LAN9514 USB Ethernet Controlled
Ethernet Port
2 x 2 USB-A Port
Run Header Used to RESET Board
www.eTechnophiles.com
3.3v
5V
GPIO 2 Serial Data (I2C)
5V
GPIO3 Serial Clock (I2C)
Ground
GPIO 4
GPIO 14 (UART TX)
Ground
GPIO 15 (UART RX)
PIO 17 Chip Enable-CE1 (SPI1)
GPIO 18 Chip Enable-CE0 (SP1) [PWM]
GPIO 27
Ground
GPIO 22
GPIO 23
3.3v
GPIO 24
GPIO 10 MOSI (SPI 0)
Ground
GPIO 09 MISO (SPI 0)
GPIO 25
GPIO 11 SCLK (SPI 0)
GPIO 8 Chip Enable-CE0 (SPI0)
Ground
GPIO 7 Chip Enable-CE1(SPI0)
GPIO 0 EEPROM Serial DATA (I2C)
GPIO 1 EEPROM Serial Clock (I2C)
GPIO 5
Ground
GPIO 6
GPIO 12 (PWM)
GPIO 13 (PWM)
Ground
[PWM] GPIO 19 MISO (SPI 1)
GPIO 16 Chip Enable-CE2 (SPI 1)
GPIO 26
GPIO 20 MISO (SPI 1)
Ground
GPIO 21 SCLK (SPI 1)

WEEK-3

Aim: Write a program to transfer sensor data to smart phone using Bluetooth on Arduino.

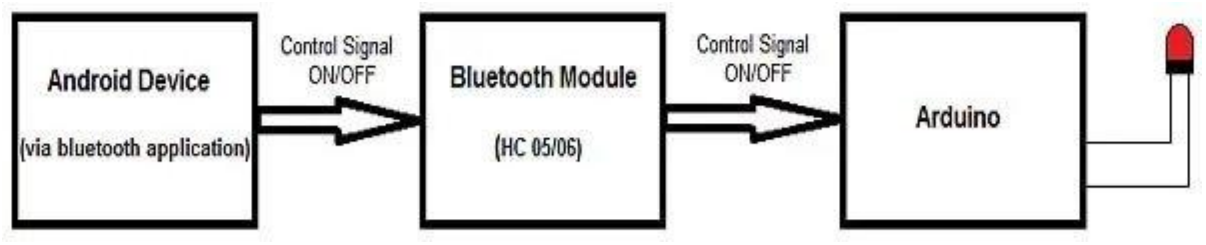
Hardware Requirements:

1. Arduino UNO
2. Android Smartphone that has Bluetooth.
3. **HC-05** Bluetooth Module
4. Android Studio (To develop the required Android app)
5. USB cable for programming and powering the Arduino

Procedure:

There are three main parts to this project.

- Smartphone
- Bluetooth transceiver
- Arduino.



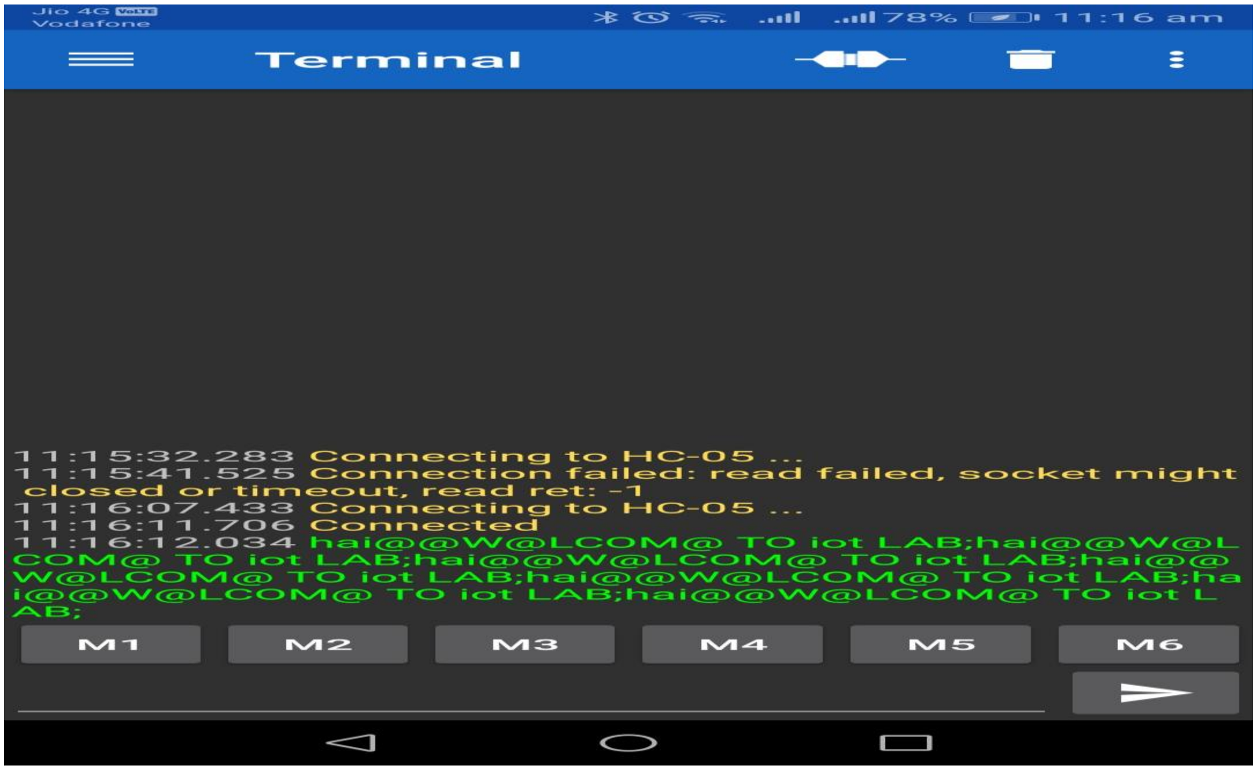
1. HC 05/06 works on serial communication.
2. The Android app is designed to send serial data to the Arduino Bluetooth module when a button is pressed on the app.
3. The Arduino Bluetooth module at the other end receives the data and sends it to the Arduino through the TX pin of the Bluetooth module (connected to RX pin of Arduino).
4. The code uploaded to the Arduino checks the received data and compares it. If the received data is 1, the LED turns ON.
5. The LED turns OFF when the received data is 0. You can open the serial monitor and watch the received data while connecting.

Source Code:

```
#include <SoftwareSerial.h>

SoftwareSerial BTserial(10, 11); // RX | TX

void setup() {

BTserial.begin(9600); }

void loop() {
```

```
BTserial.print("hai");
BTserial.print("CMRIT");
BTserial.print("WELCOME TO IOT LAB");
BTserial.print("OK");
delay(3000);
}
```

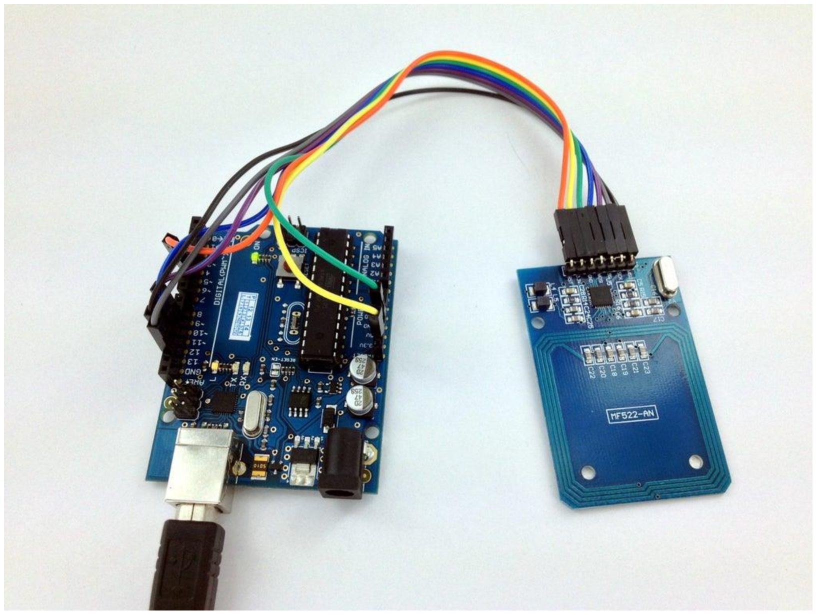
OUTPUT:



```
11:15:32.283 Connecting to HC-05 ...
11:15:41.525 Connection failed: read failed, socket might
 closed or timeout, read ret: -1
11:16:07.433 Connecting to HC-05 ...
11:16:11.706 Connected
11:16:12.034 hai@@W@LCOM@ TO iot LAB;hai@@W@L
COM@ TO iot LAB;hai@@W@LCOM@ TO iot LAB;hai@@
W@LCOM@ TO iot LAB;hai@@W@LCOM@ TO iot LAB;ha
i@@W@LCOM@ TO iot LAB;hai@@W@LCOM@ TO iot L
AB;
```

WEEK-4

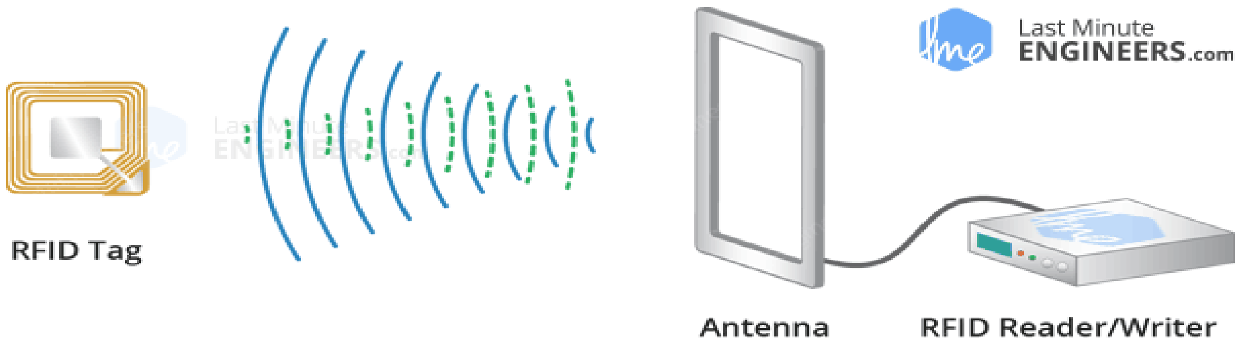
AIM: Write a program to implement RFID using Arduino.



What is RFID technology and how does it work?

An RFID or radio frequency identification system consists of two main components, a tag attached to the object to be identified, and a reader that reads the tag.

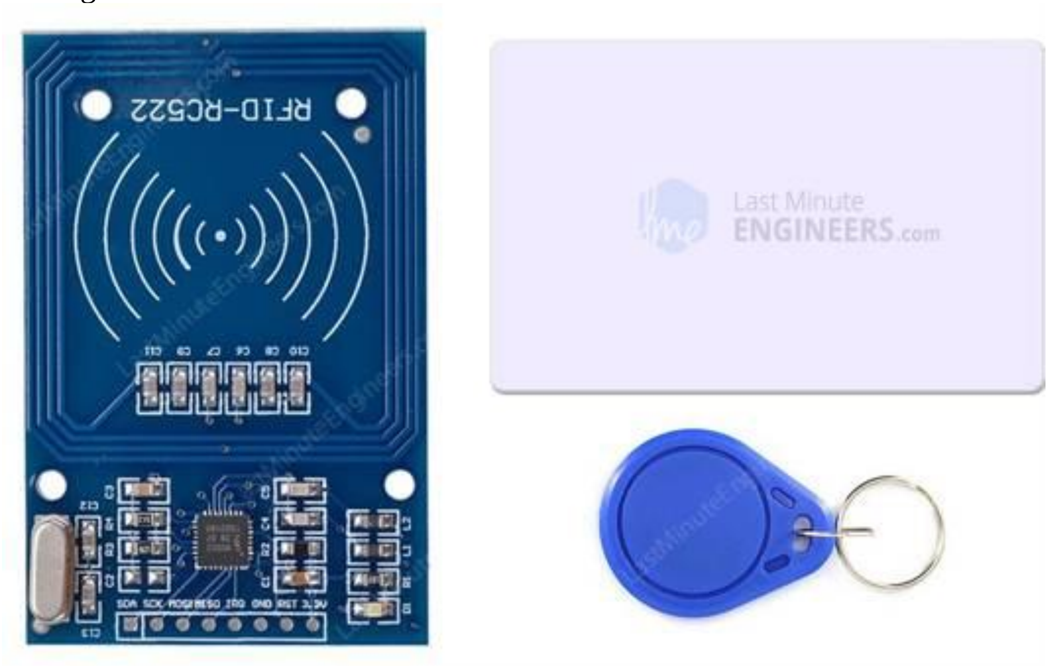
A reader consists of a radio frequency module and an antenna that generates a high frequency electromagnetic field. Whereas the tag is usually a passive device (it does not have a battery). It consists of a microchip that stores and processes information, and an antenna for receiving and transmitting a signal.



When the tag is brought close to the reader, the reader generates an electromagnetic field. This causes electrons to move through the tag's antenna and subsequently powers the chip.

Hardware Overview

The RC522 RFID module based on the MFRC522 IC is one of the cheapest RFID options you can get online for less than four dollars. It usually comes with an RFID card tag and a key fob tag with 1KB of memory. And the best part is that it can write a tag that means you can store any message in it.



The RC522 RFID reader module is designed to create a 13.56MHz electromagnetic field and communicate with RFID tags (ISO 14443A standard tags).

The reader can communicate with a microcontroller over a 4-pin SPI with a maximum data rate of 10 Mbps. It also supports communication over I2C and UART protocols.

The RC522 RFID module can be programmed to generate an interrupt, allowing the module to alert us when a tag approaches it, instead of constantly asking the module "Is there a card nearby?".

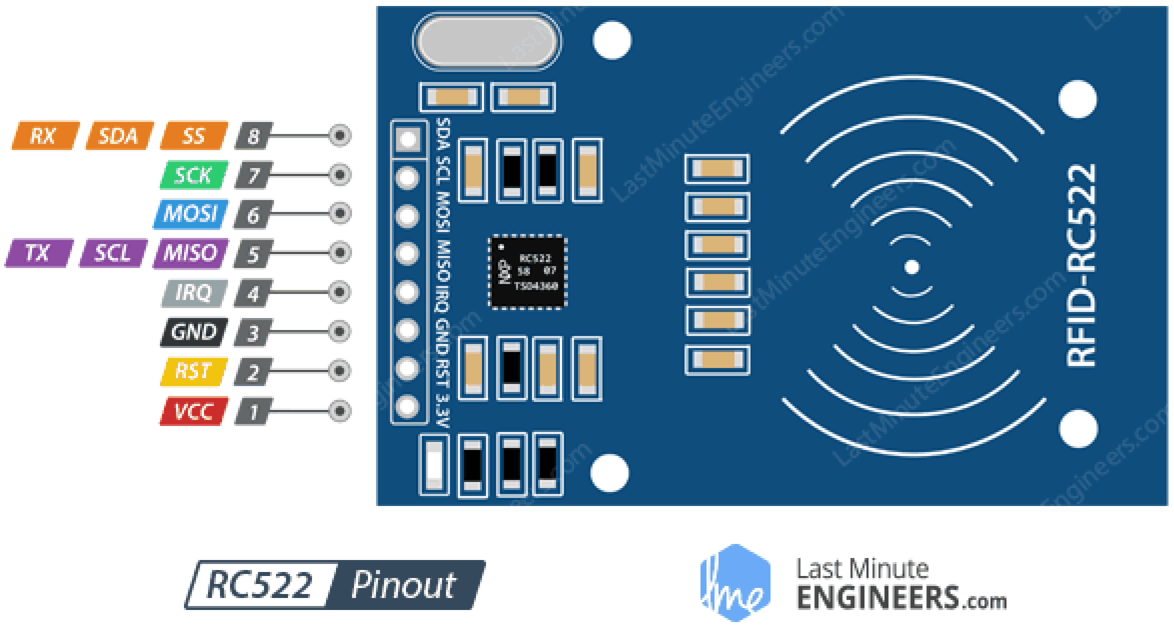
The module's operating voltage ranges from 2.5 to 3.3V, but the good news is that the logic pins are 5-volt tolerant, so we can easily connect it to an Arduino or any 5V logic microcontroller without using a logic level converter.

Technical Specifications

Frequency Range	13.56 MHz ISM Band
Host Interface	SPI / I2C / UART
Operating Supply Voltage	2.5 V to 3.3 V
Max. Operating Current	13-26mA

Min. Current(Power down)	10µA
Logic Inputs	5V Tolerant
Read Range	5 cm

The RC522 module has a total of 8 pins that connect it to the outside world. The connections are as follows:



VCC supplies power to the module. This can be anywhere from 2.5 to 3.3 volts. You can connect it to the 3.3V output from your Arduino. But remember that connecting it to the 5V pin will probably destroy your module!
RST is an input for reset and power-down. When this pin goes low the module enters power-down mode. In which the oscillator is turned off and the input pins are disconnected from the outside world. Whereas the module is reset on the rising edge of the signal.
GND is the ground pin and needs to be connected to the GND pin on the Arduino.
IRQ is an interrupt pin that alerts the microcontroller when an RFID tag is in the vicinity.
MISO / SCL / Tx pin acts as master-in-slave-out when SPI interface is enabled, as serial clock when I2C interface is enabled and as serial data output when the UART interface is enabled.
MOSI (Master Out Slave In) is the SPI input to the RC522 module.
SCK (Serial Clock) accepts the clock pulses provided by the SPI bus master i.e. Arduino.
SS / SDA / Rx pin acts as a signal input when the SPI interface is enabled, as serial data when the I2C interface is enabled and as a serial data input when the UART interface is enabled. This pin is usually marked by encasing the pin in a square so that it can be used as a reference to identify other pins.

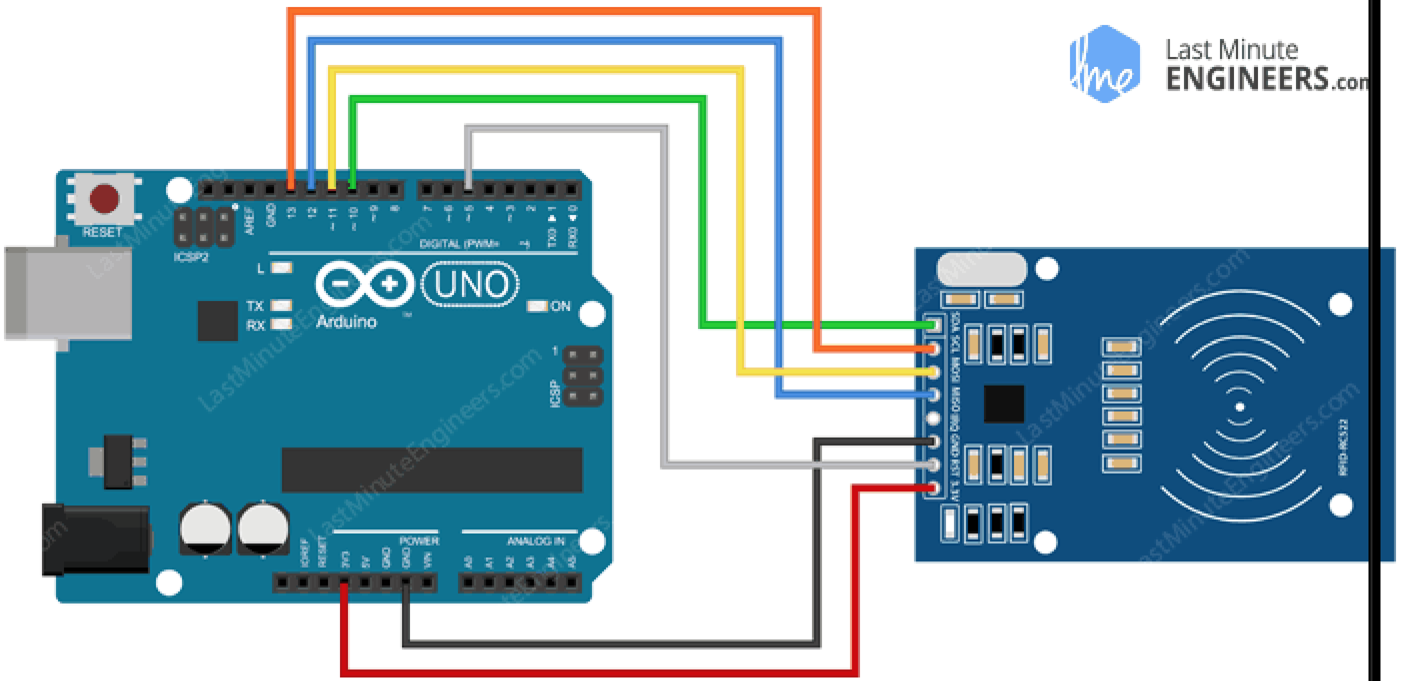
First connect the VCC pin on the module to 3.3V and the GND pin to ground on the Arduino. Pin RST can be connected to any digital pin on the Arduino. In our case, it is connected to digital pin #5. The IRQ pin is left unconnected because the Arduino library we are going to use does not support it.

Now we are left with the pins that are used for SPI communication. Since RC522 modules require a lot of data transfer, they will give the best performance when connected to the hardware SPI pins on the microcontroller.

Note that each Arduino board has different SPI pins that must be connected accordingly. Check the table below for quick understanding.

	MOSI	MISO	SCK	CS
Arduino Uno	11	12	13	10
Arduino Nano	11	12	13	10
Arduino Mega	51	50	52	53

If you are using a different Arduino than the boards mentioned above, please check the Arduino's official documentation before proceeding.



Wiring RC522 RFID Reader Writer Module with Arduino UNO

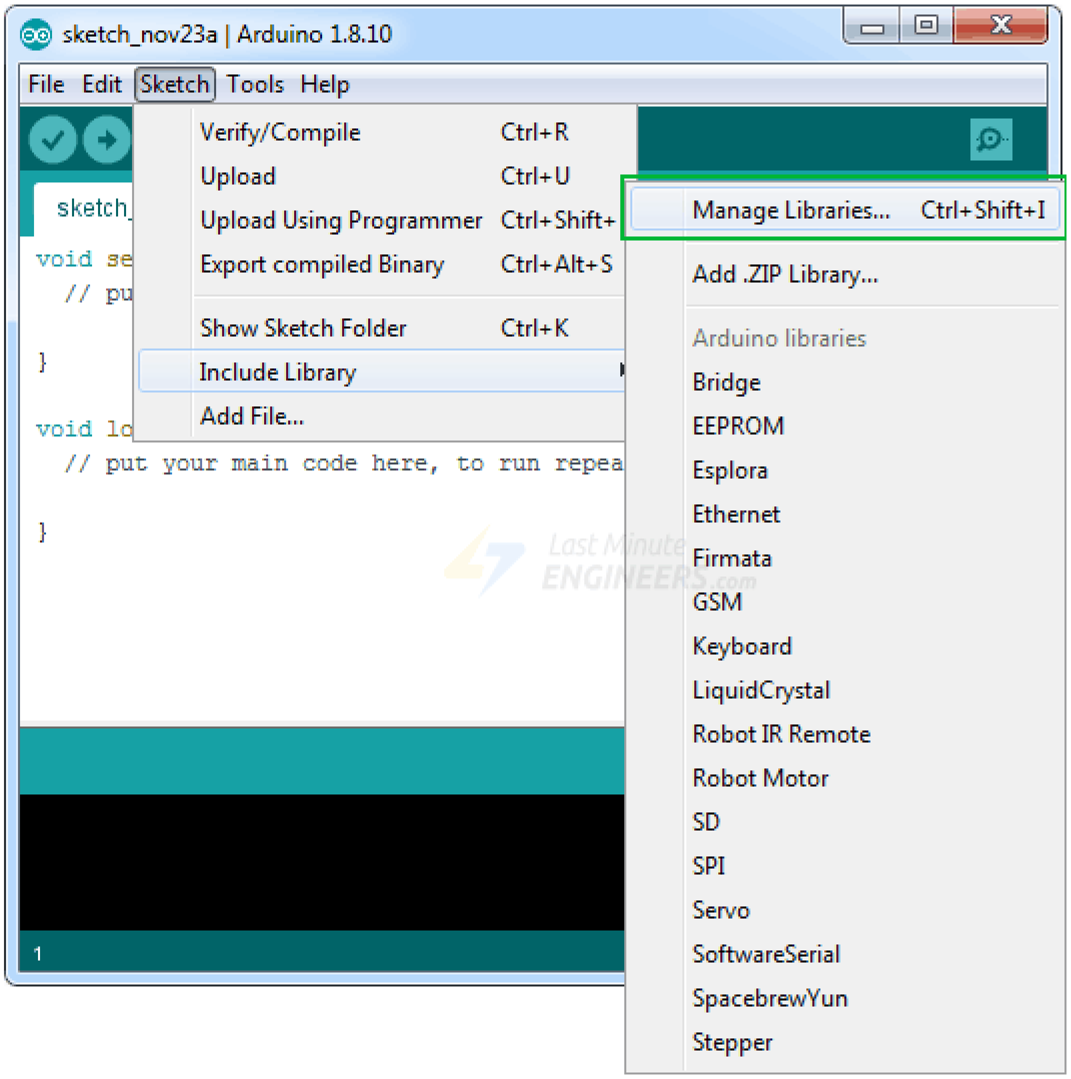
Once you have connected everything you are ready to go!

Library Installation

Communicating with an RC522 RFID module is a lot of work, but luckily for us there is a library called the MFRC522 library that makes reading and writing RFID tags simple.

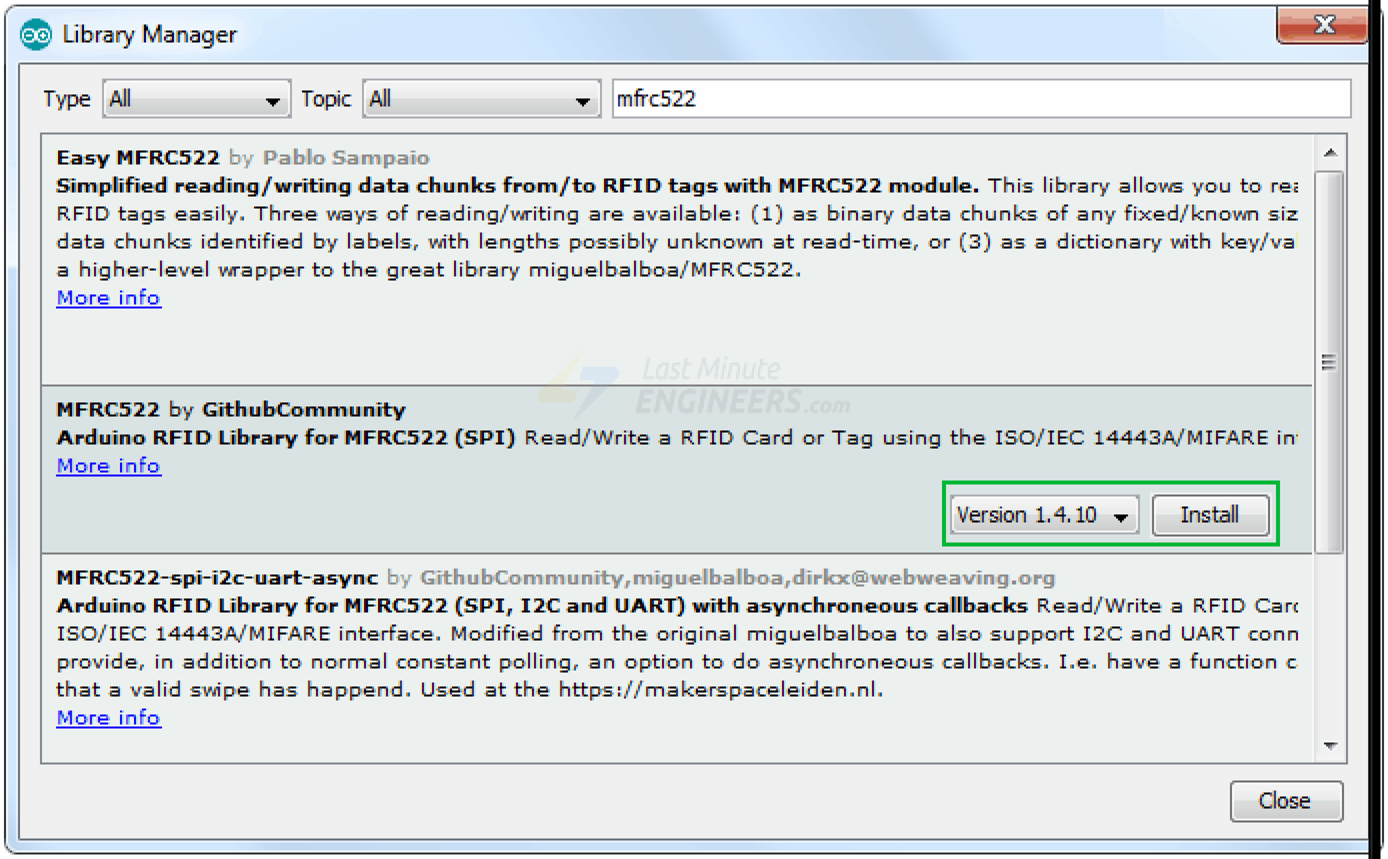
This library is not included in the Arduino IDE, so you will need to install it first.

To install the library navigate to Sketch > Include Libraries > Manage Libraries… Wait for Library Manager to download the library index and update the list of installed libraries.



```
void se
  // pu

}

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

}
```

Filter your search by typing ‘mfrc522‘. Look for the library by GithubCommunity. Click on that entry, and then select Install.


Library Manager

Type All Topic All mfrc522

Easy MFRC522 by Pablo Sampaio
Simplified reading/writing data chunks from/to RFID tags with MFRC522 module. This library allows you to rea
RFID tags easily. Three ways of reading/writing are available: (1) as binary data chunks of any fixed/known siz
data chunks identified by labels, with lengths possibly unknown at read-time, or (3) as a dictionary with key/va
a higher-level wrapper to the great library miguelbalboa/MFRC522.
More info

MFRC522 by **GithubCommunity**
Arduino RFID Library for MFRC522 (SPI) Read/Write a RFID Card or Tag using the ISO/IEC 14443A/MIFARE in
More info

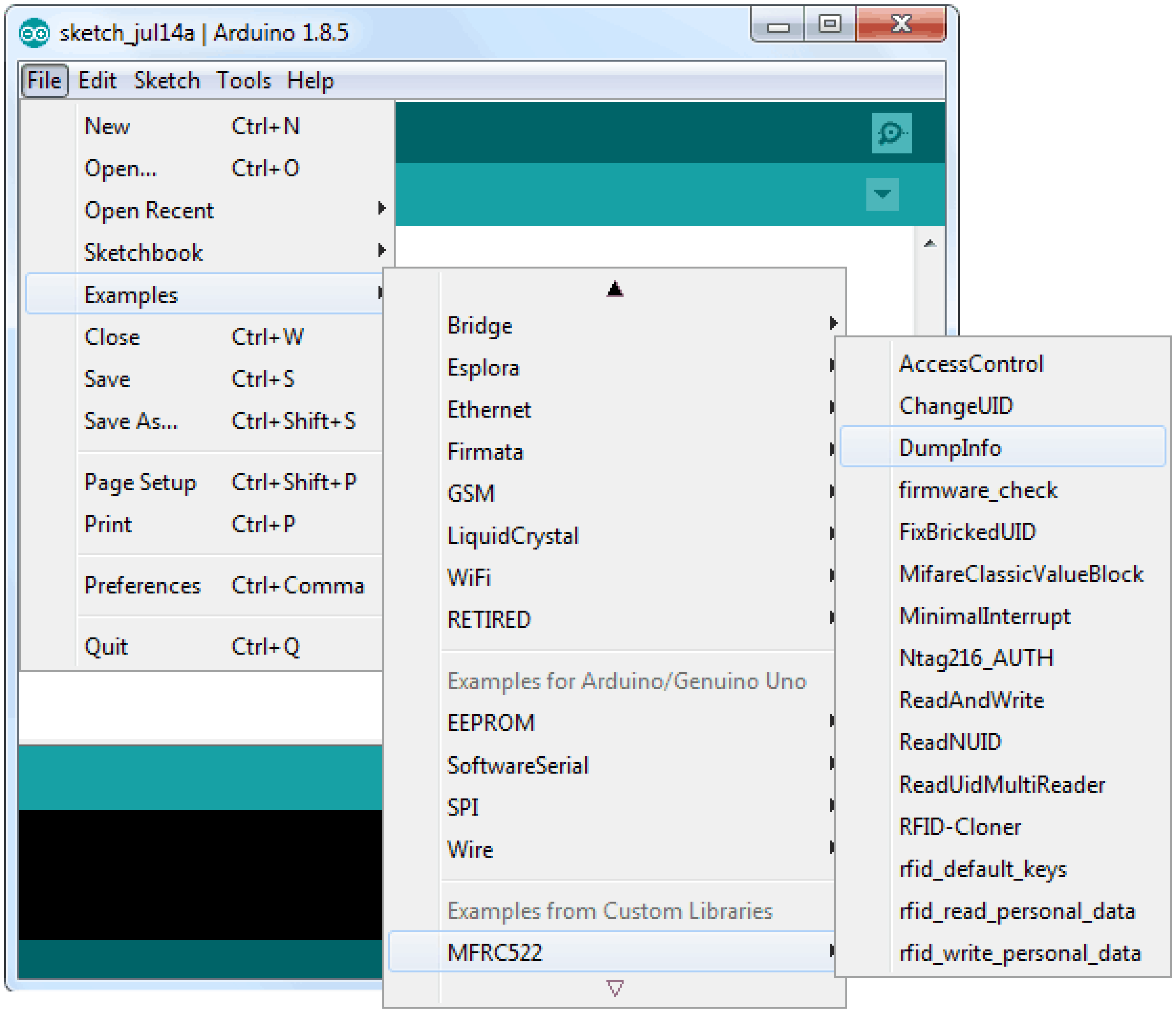
Version 1.4.10 Install

MFRC522-spi-i2c-uart-async by GithubCommunity,miguelbalboa,dirkx@webweaving.org
Arduino RFID Library for MFRC522 (SPI, I2C and UART) with asynchroneous callbacks Read/Write a RFID Card
ISO/IEC 14443A/MIFARE interface. Modified from the original miguelbalboa to also support I2C and UART conn
provide, in addition to normal constant polling, an option to do asynchroneous callbacks. I.e. have a function c
that a valid swipe has happend. Used at the https://makerspaceleiden.nl.
More info

Close

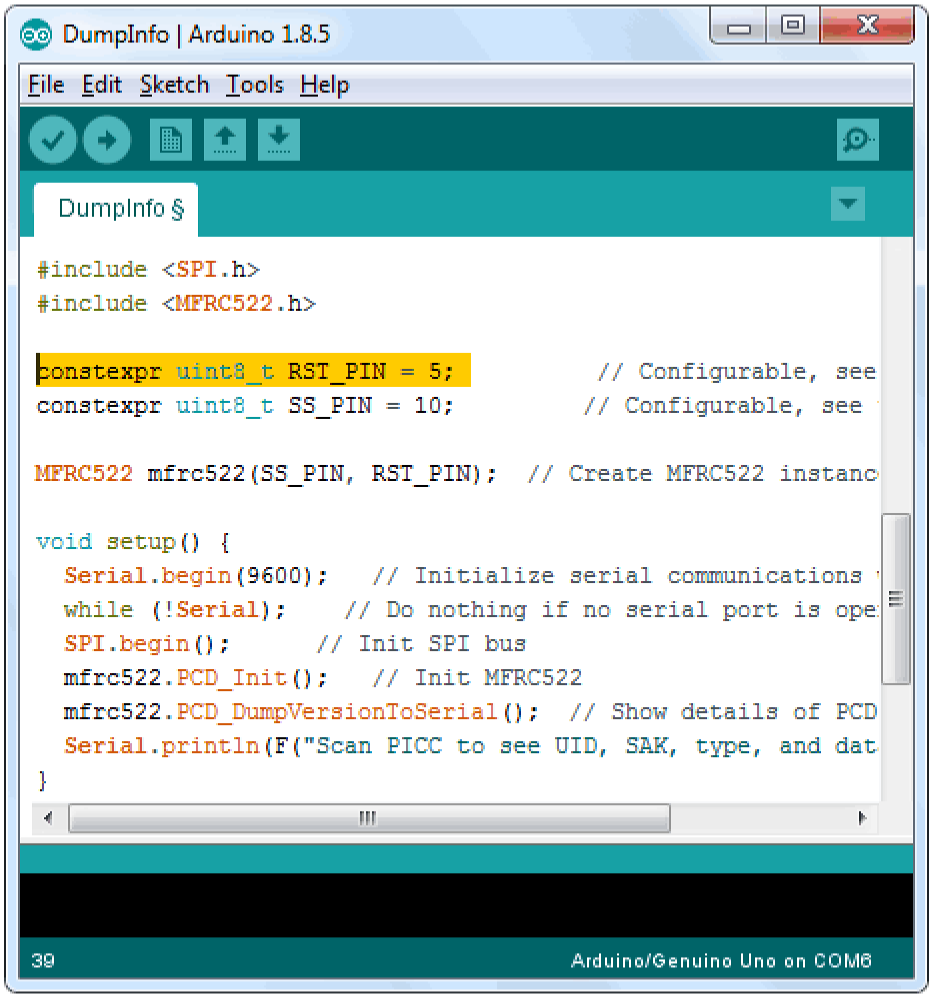
Arduino Code – Reading an RFID Tag

Once you have installed the library, open the Examples submenu and choose MFRC522 > DumpInfo example sketch.



This sketch just reads the tag and displays the information stored in it. This sketch can be very handy before trying out any new tags!

Go to the beginning of the sketch and make sure RST_PIN is initialized correctly, in our case we are using digital pin #5 so change it to 5.



```
#include <SPI.h>
#include <MFRC522.h>

constexpr uint8_t RST_PIN = 5;          // Configurable, see
constexpr uint8_t SS_PIN = 10;         // Configurable, see

MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instanc

void setup() {
  Serial.begin(9600);   // Initialize serial communications
  while (!Serial);    // Do nothing if no serial port is ope
  SPI.begin();      // Init SPI bus
  mfrc522.PCD_Init();   // Init MFRC522
  mfrc522.PCD_DumpVersionToSerial();  // Show details of PCD
  Serial.println(F("Scan PICC to see UID, SAK, type, and dat
}
```

```
39                                Arduino/Genuino Uno on COM6
```

Now upload the sketch and open Serial Monitor. As you bring the tag closer to the module, you'll get something like the following. Do not move the tag until all the information is displayed.

```
#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN        9          // Configurable, see typical pin layout above
#define SS_PIN        10         // Configurable, see typical pin layout above

MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance

void setup() {
        Serial.begin(9600);             // Initialize serial communications with the PC
        while (!Serial);                // Do nothing if no serial port is opened (added for
Arduinos based on ATMEGA32U4)
        SPI.begin();                    // Init SPI bus
        mfrc522.PCD_Init();             // Init MFRC522
        delay(4);                       // Optional delay. Some board do need more time
after init to be ready, see Readme
        mfrc522.PCD_DumpVersionToSerial();      // Show details of PCD - MFRC522 Card
Reader details
        Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
}

void loop() {
        // Reset the loop if no new card present on the sensor/reader. This saves the entire process
when idle.
        if ( ! mfrc522.PICC_IsNewCardPresent()) {
                return;
        }

        // Select one of the cards
        if ( ! mfrc522.PICC_ReadCardSerial()) {
                return;
        }

        // Dump debug info about the card; PICC_HaltA() is automatically called
        mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```

Reading Data from Card:

```
#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN         9           // Configurable, see typical pin layout above
#define SS_PIN          10          // Configurable, see typical pin layout above

MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance

//*****************************************************************************
*************//
void setup() {
  Serial.begin(9600);                                        // Initialize serial communications with the PC
  SPI.begin();                                               // Init SPI bus
  mfrc522.PCD_Init();                                              // Init MFRC522 card
  Serial.println(F("Read personal data on a MIFARE PICC:"));    //shows in serial that it is ready
to read
}

//*****************************************************************************
*************//
void loop() {

  // Prepare key - all keys are set to FFFFFFFFFFFFh at chip delivery from the factory.
  MFRC522::MIFARE_Key key;
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

  //some variables we need
  byte block;
  byte len;
  MFRC522::StatusCode status;

  //-------------------------------------------

  // Reset the loop if no new card present on the sensor/reader. This saves the entire process when
idle.
  if ( ! mfrc522.PICC_IsNewCardPresent()) {
    return;
  }

  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial()) {
    return;
  }

  Serial.println(F("**Card Detected:**"));

  //-------------------------------------------

  mfrc522.PICC_DumpDetailsToSerial(&(mfrc522.uid)); //dump some details about the card

  //mfrc522.PICC_DumpToSerial(&(mfrc522.uid));      //uncomment this to see all blocks in hex

  //-------------------------------------------

  Serial.print(F("Name: "));

  byte buffer1[18];

  block = 4;
  len = 18;

  //------------------------------------------- GET FIRST NAME
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 4, &key,
&(mfrc522.uid)); //line 834 of MFRC522.cpp file
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Authentication failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  status = mfrc522.MIFARE_Read(block, buffer1, &len);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }

  //PRINT FIRST NAME
  for (uint8_t i = 0; i < 16; i++)
  {
    if (buffer1[i] != 32)
    {
      Serial.write(buffer1[i]);
    }
  }
  Serial.print(" ");

  //---------------------------------------- GET LAST NAME

  byte buffer2[18];
  block = 1;

  status  =  mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,  1,  &key,
&(mfrc522.uid)); //line 834
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Authentication failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }

  status = mfrc522.MIFARE_Read(block, buffer2, &len);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }

  //PRINT LAST NAME
  for (uint8_t i = 0; i < 16; i++) {
    Serial.write(buffer2[i] );
  }


  //----------------------------------------
  Serial.println(F("\n**End Reading**\n"));

  delay(1000); //change value if you want to read cards faster

  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
}
```

Writing Data from Card:

```
#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN         9           // Configurable, see typical pin layout above
#define SS_PIN          10          // Configurable, see typical pin layout above

MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance

void setup() {
  Serial.begin(9600);        // Initialize serial communications with the PC
  SPI.begin();               // Init SPI bus
  mfrc522.PCD_Init();        // Init MFRC522 card
  Serial.println(F("Write personal data on a MIFARE PICC "));
}

void loop() {

  // Prepare key - all keys are set to FFFFFFFFFFFFh at chip delivery from the factory.
  MFRC522::MIFARE_Key key;
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

  // Reset the loop if no new card present on the sensor/reader. This saves the entire process when
idle.
  if ( ! mfrc522.PICC_IsNewCardPresent()) {
    return;
  }

  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial()) {
    return;
  }

  Serial.print(F("Card UID:"));    //Dump UID
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.print(F(" PICC type: "));   // Dump PICC type
  MFRC522::PICC_Type piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
  Serial.println(mfrc522.PICC_GetTypeName(piccType));

  byte buffer[34];
  byte block;
  MFRC522::StatusCode status;
  byte len;
  Serial.setTimeout(20000L) ;     // wait until 20 seconds for input from serial
  // Ask personal data: Family name
  Serial.println(F("Type Family name, ending with #"));
  len = Serial.readBytesUntil('#', (char *) buffer, 30) ; // read family name from serial
  for (byte i = len; i < 30; i++) buffer[i] = ' ';     // pad with spaces

  block = 1;
  //Serial.println(F("Authenticating using key A..."));
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else Serial.println(F("PCD_Authenticate() success: "));

  // Write block
  status = mfrc522.MIFARE_Write(block, buffer, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else Serial.println(F("MIFARE_Write() success: "));

  block = 2;
  //Serial.println(F("Authenticating using key A..."));
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  status = mfrc522.MIFARE_Write(block, &buffer[16], 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else Serial.println(F("MIFARE_Write() success: "));
  // Ask personal data: First name
  Serial.println(F("Type First name, ending with #"));
  len = Serial.readBytesUntil('#', (char *) buffer, 20) ; // read first name from serial
  for (byte i = len; i < 20; i++) buffer[i] = ' ';     // pad with spaces

  block = 4;
  //Serial.println(F("Authenticating using key A..."));
  status   =   mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,   block,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }

  // Write block
  status = mfrc522.MIFARE_Write(block, buffer, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else Serial.println(F("MIFARE_Write() success: "));
  block = 5;
  //Serial.println(F("Authenticating using key A..."));
  status   =   mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,   block,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  status = mfrc522.MIFARE_Write(block, &buffer[16], 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else Serial.println(F("MIFARE_Write() success: "));
  Serial.println(" ");
  mfrc522.PICC_HaltA(); // Halt PICC
  mfrc522.PCD_StopCrypto1();  // Stop encryption on PCD}
```

WEEK-5

AIM: Write a program to monitor temperature and humidity using Arduino and Raspberry Pi.

Hardware Requirements:

1. Arduino UNO Board
2. DHT11 Temperature and Humidity Sensor(3 pins)
3. Jumper Wires
4. Bread Board

Procedure:

1. Connect pin 1 (on the left) of the sensor to +5V.

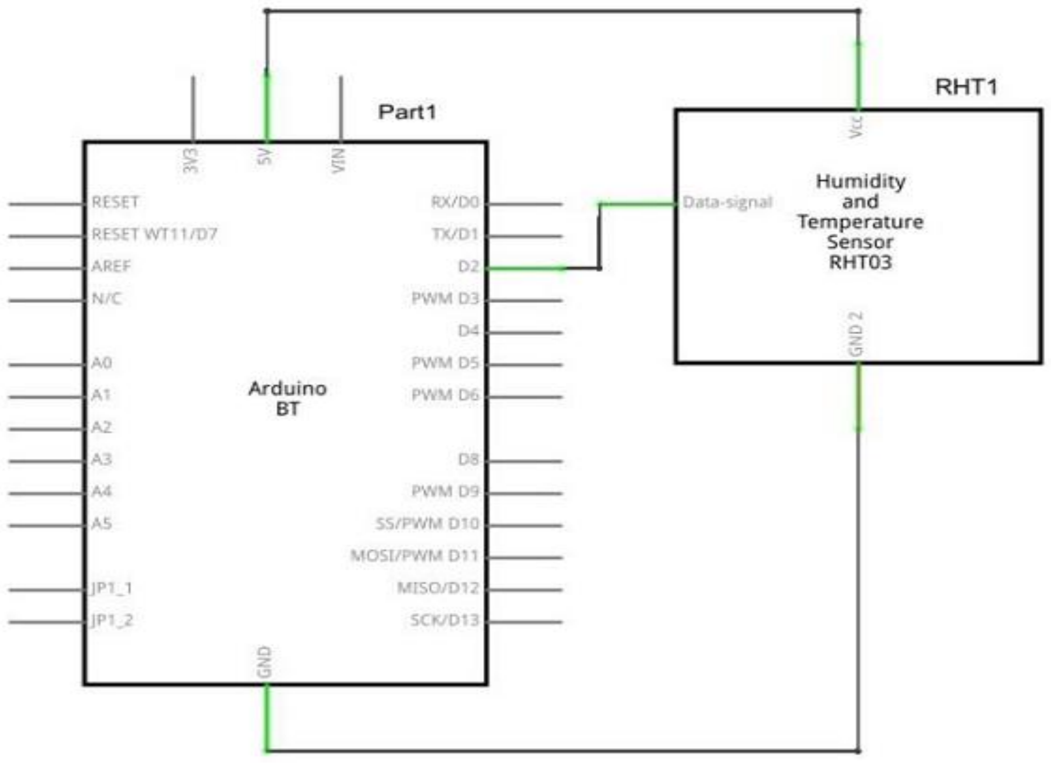
NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1

// to 3.3V instead of 5V!

2. Connect pin 2 of the sensor to whatever your DHTPIN is

3. Connect pin 4 (on the right) of the sensor to GROUND

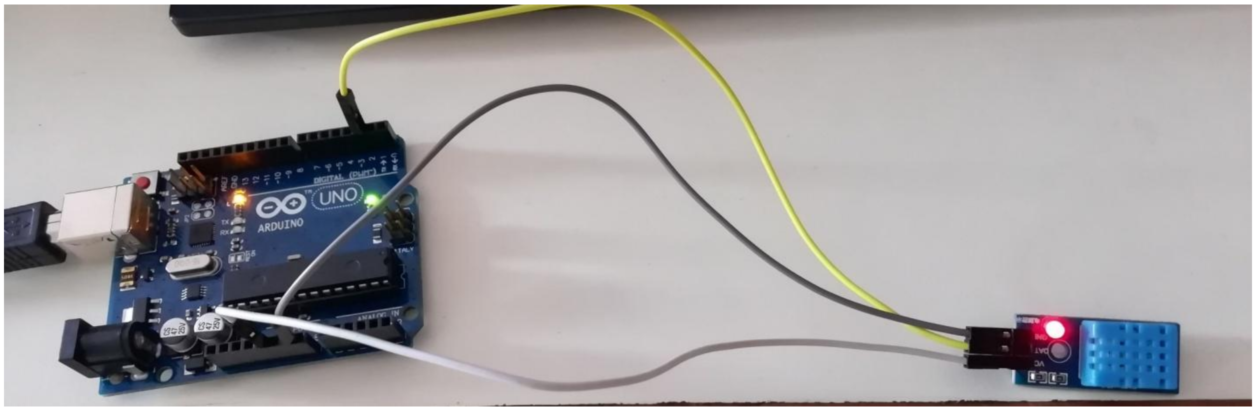
4. Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

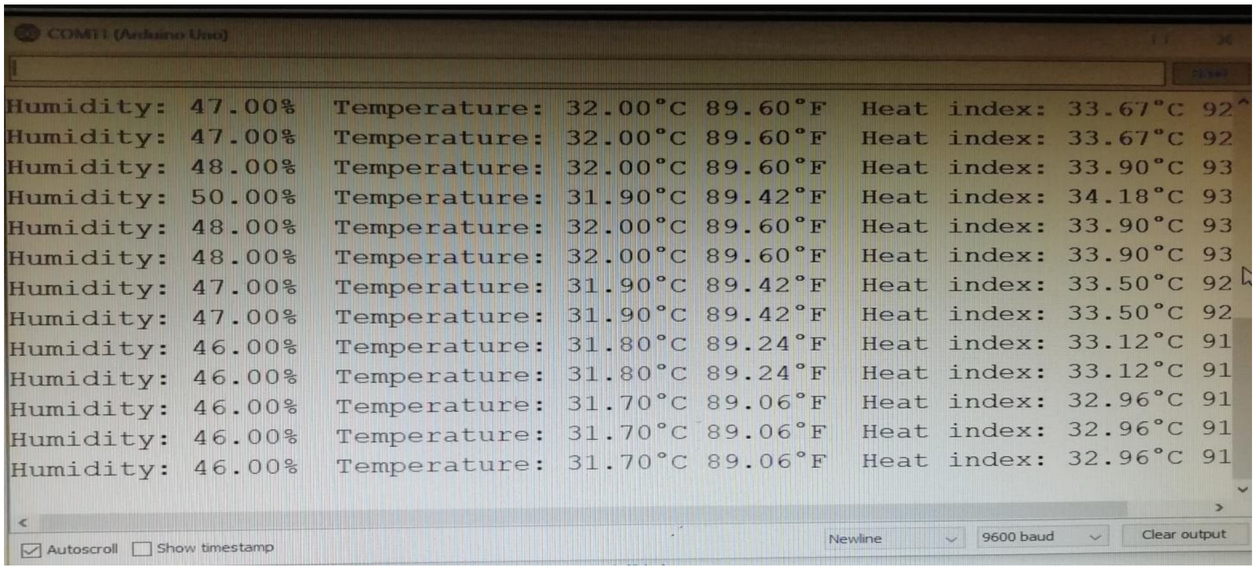


Source Code:

```
#include "DHT.h"
#define DHTPIN 2
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT11   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
DHT dht(DHTPIN, DHTTYPE);
void setup() {
 Serial.begin(9600);
Serial.println(F("DHTxx test!"));
dht.begin();
}
void loop() {
 delay(2000);
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);
  if (isnan(h) || isnan(t) || isnan(f)) {
Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
  float hif = dht.computeHeatIndex(f, h);
  float hic = dht.computeHeatIndex(t, h, false);
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

OUTPUT:





```
Humidity: 47.00%  Temperature: 32.00°C 89.60°F  Heat index: 33.67°C 92
Humidity: 47.00%  Temperature: 32.00°C 89.60°F  Heat index: 33.67°C 92
Humidity: 48.00%  Temperature: 32.00°C 89.60°F  Heat index: 33.90°C 93
Humidity: 50.00%  Temperature: 31.90°C 89.42°F  Heat index: 34.18°C 93
Humidity: 48.00%  Temperature: 32.00°C 89.60°F  Heat index: 33.90°C 93
Humidity: 48.00%  Temperature: 32.00°C 89.60°F  Heat index: 33.90°C 93
Humidity: 47.00%  Temperature: 31.90°C 89.42°F  Heat index: 33.50°C 92
Humidity: 47.00%  Temperature: 31.90°C 89.42°F  Heat index: 33.50°C 92
Humidity: 46.00%  Temperature: 31.80°C 89.24°F  Heat index: 33.12°C 91
Humidity: 46.00%  Temperature: 31.80°C 89.24°F  Heat index: 33.12°C 91
Humidity: 46.00%  Temperature: 31.70°C 89.06°F  Heat index: 32.96°C 91
Humidity: 46.00%  Temperature: 31.70°C 89.06°F  Heat index: 32.96°C 91
Humidity: 46.00%  Temperature: 31.70°C 89.06°F  Heat index: 32.96°C 91
```

WEEK-6

Aim: Write a program to interface IR Sensor with Arduino using IoT cloud application

Hardware Requirements:

1. Arduino UNO Board
2. Momentary button or Switch
3. 10K ohm resistor
4. hook-up wires
5. breadboard

Procedure:

Connect the Arduino board to your computer using the USB cable
Connect three wires to the Arduino board.

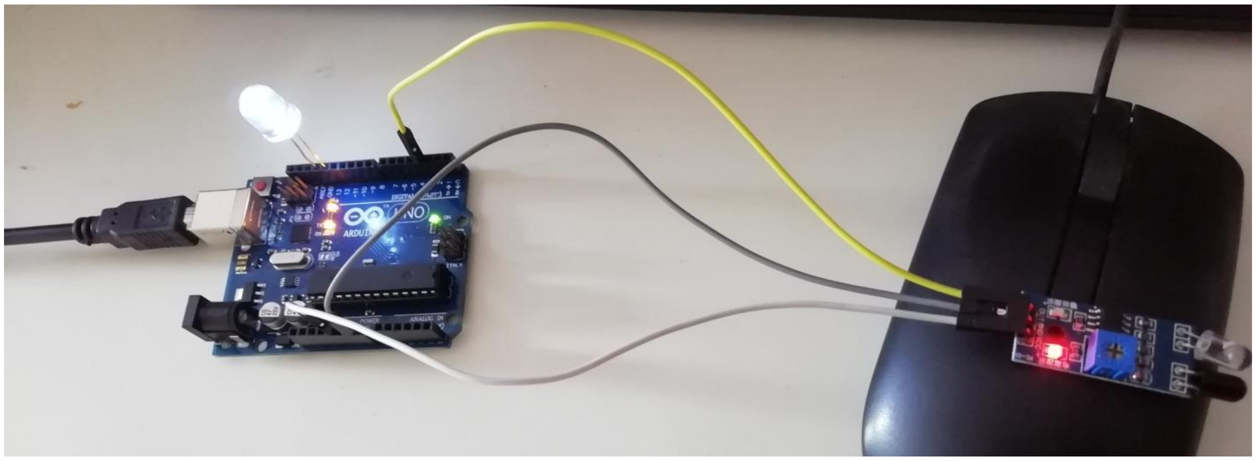
1. The first goes from one leg of the pushbutton through a pull-up resistor (here 2.2 Ohms) to the 5 volt supply.
2. The second goes from the corresponding leg of the pushbutton to ground.
3. The third connects to a digital i/o pin (here pin 7) which reads the button's state.
4. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to 5 volts (through the pull-up resistor) and we read a HIGH.
5. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to ground, so that we read a LOW. (The pin is still connected to 5 volts, but the resistor in-between those mean that the pin is "closer" to ground.)

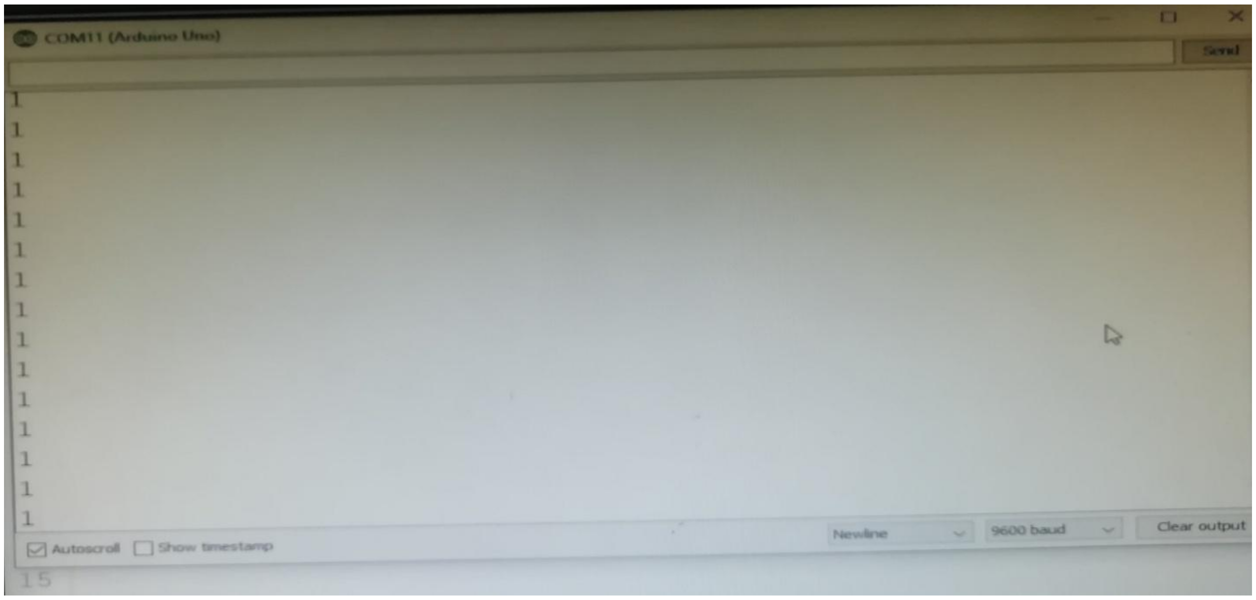
Source code:

```
int IRPin = 3;
int led=13;
int value;
void setup(){
pinMode(IRPin, INPUT);
Serial.begin(9600);
pinMode(13,OUTPUT);

}
void loop(){
 value = digitalRead(IRPin);
Serial.println(value);
  if(digitalRead(IRPin)==1)
 digitalWrite(led,HIGH);

  else

  digitalWrite(led,LOW);
```

}

OUTPUT:





```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

WEEK-7

Aim: Write a program to upload temperature and humidity data to the cloud using Arduino or Raspberry Pi.

Hardware Requirements:

1. Arduino UNO board
2. NodeMCU ESP8266 Breakout Board
3. DHT-11 temperature and humidity sensor
4. Jumper wires
5. Bread board
6. WIFI Network

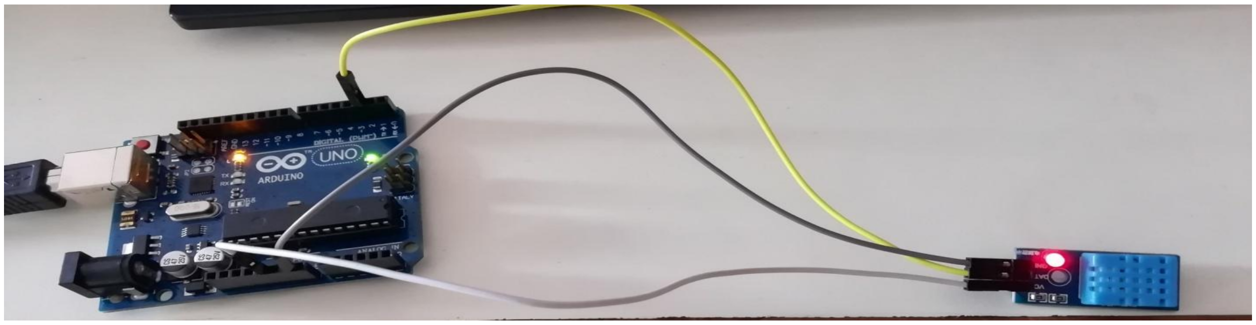
Procedure:

1. Download esp8266 Zip file->go to libraries->add Zip file
2. Connect Node MCU, Go to tools->change board to Node MCU esp8266 and port number
3. Connect DHT-11 temperature and Humidity sensor to Node MCU
4. Sign up to cloud->create channels->copy API key to the source code
5. SSID and password of your WIFI connection should be given in source code
6. Compile and upload the program and verify the temperature and humidity readings in serial monitor
7. Go to cloud and verify the temperature and humidity values in graph.

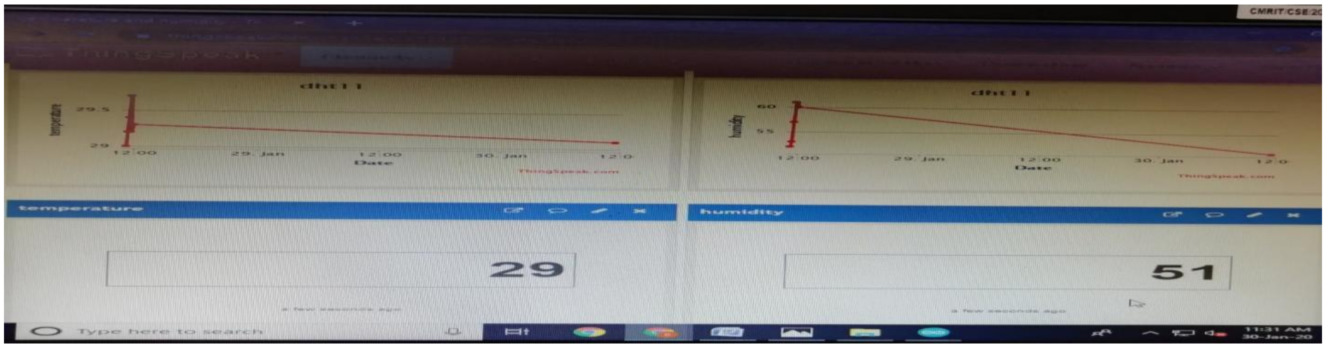
SOURCE CODE:

```
#include <DHT.h>
#include <DHT_U.h>
#include <DHT.h>
#include <ESP8266WiFi.h>

String apiKey = "WAHKMAHIYUR1SEFU";
const char *ssid =  "chitti";
const char *pass =  "sari123#";
const char* server = "api.thingspeak.com";

#define DHTPIN 0

DHT dht(DHTPIN, DHT11);

WiFiClient client;
 void setup()
{
  Serial.begin(115200);
delay(10);
dht.begin();
Serial.println("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED)
{
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

}
void loop()
{

float h = dht.readHumidity();
float t = dht.readTemperature();

if (isnan(h) || isnan(t))
{
Serial.println("Failed to read from DHT sensor!");
return;
}

if (client.connect(server,80))   //   "184.106.153.149" or api.thingspeak.com
{

String postStr = apiKey;
postStr +="&field1=";
postStr += String(t);
postStr +="&field2=";
postStr += String(h);
postStr += "\r\n\r\n";

client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
client.print("Content-Type: application/x-www-form-urlencoded\n");
client.print("Content-Length: ");
client.print(postStr.length());
client.print("\n\n");
client.print(postStr);

Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" degrees Celcius, Humidity: ");
Serial.print(h);
Serial.println("%. Send to Thingspeak.");
}
client.stop();

Serial.println("Waiting...");

delay(10000);
}
```

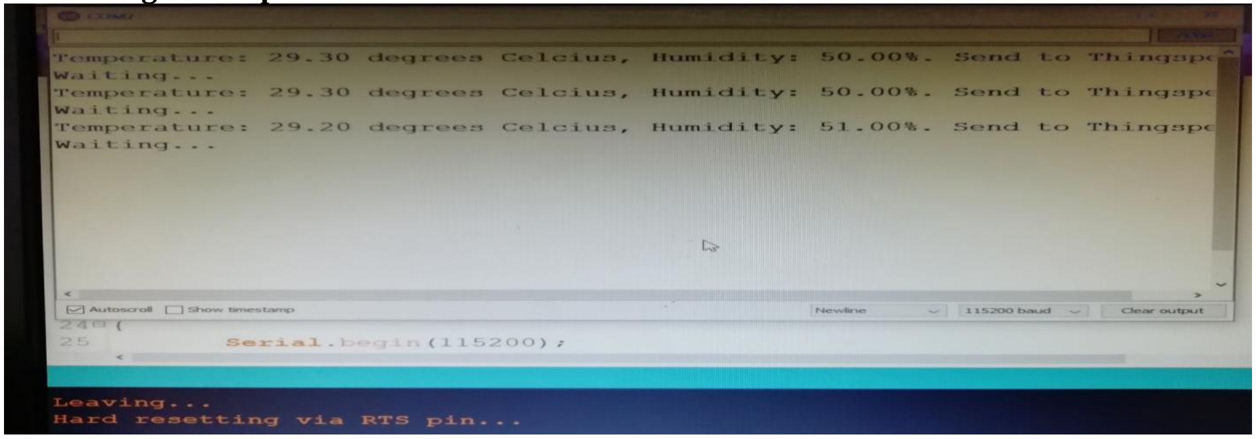
OUTPUT:



Uploading the data in CLOUD:



Visualizing the output in Serial monitor:



```
Temperature: 29.30 degrees Celcius, Humidity: 50.00%. Send to Thingspe
Waiting...
Temperature: 29.30 degrees Celcius, Humidity: 50.00%. Send to Thingspe
Waiting...
Temperature: 29.20 degrees Celcius, Humidity: 51.00%. Send to Thingspe
Waiting...
```

```
Serial.begin(115200);
```

```
Leaving...
Hard resetting via RTS pin...
```

WEEK-8

Aim: Write a program to retrieve temperature and humidity data to the cloud using Arduino or Raspberry Pi.

Hardware Requirements:

1. Arduino UNO board
2. NodeMCU ESP8266 Breakout Board
3. DHT-11 temperature and humidity sensor
4. Jumper wires
5. Bread board

Procedure:

1. Open up the Arduino IDE and head over to the library manager.

2. Install the DHT library (You can also install it by going to Sketch > Include Library > Manage Libraries, and search for adafruit dht library).

DHT sensor with 3 pins:

1. Power supply 3.5V to 5.5V.

2. Data, Outputs both Temperature and Humidity through serial Data.

3. Ground, Connected to the ground of the circuit.

Set up in source code:

1. Set your Wi-Fi SSID and password.

2. Set the API Key

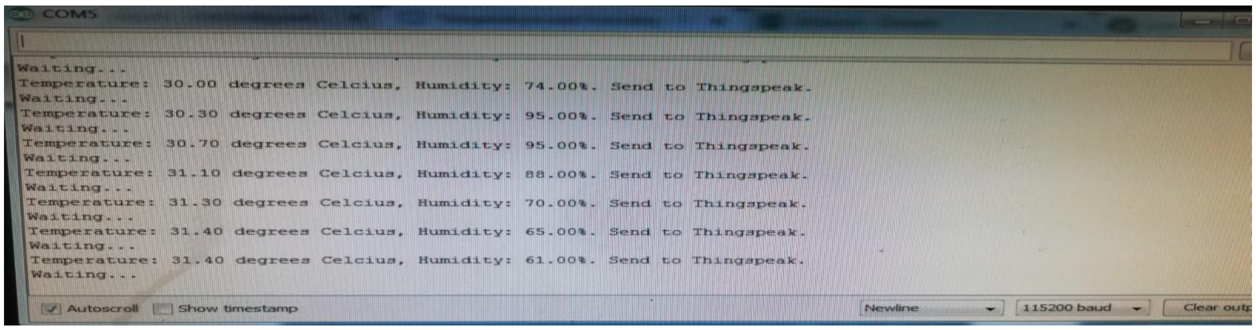
3. Save->Compile->upload->now us can visualize our data in cloud

Source Code:

```
#include <DHT.h>
#include <DHT_U.h>

// Robo India Tutorial
// Simple code upload the tempeature and humidity data using thingspeak.com
// Hardware: NodeMCU,DHT22

#include <DHT.h>  // Including library for dht
```

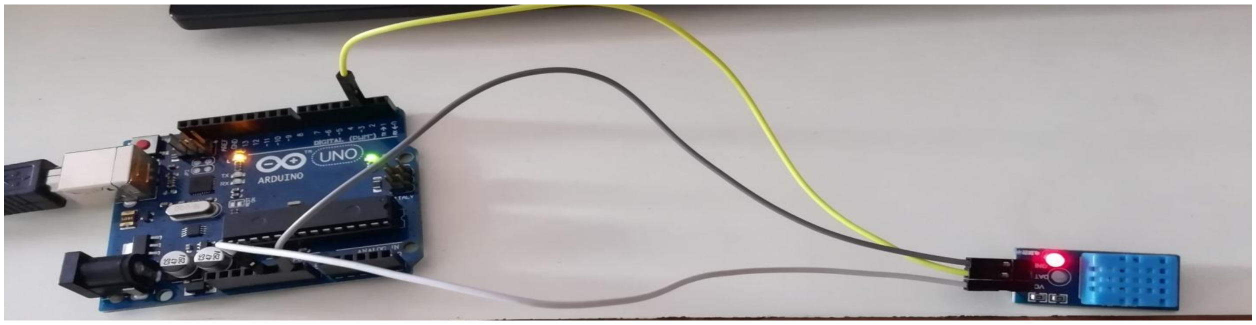
```
#include <ESP8266WiFi.h>

String apiKey = "6YL93NXXHFZ964NA";     //  Enter your Write API key from ThingSpeak
const char *ssid =  "chitti";     // replace with your wifi ssid and wpa2 key
const char *pass =  "sari123#";
const char* server = "api.thingspeak.com";

#define DHTPIN 0          //pin where the dht11 is connected

DHT dht(DHTPIN, DHT11);

WiFiClient client;

void setup()
{
       Serial.begin(115200);
       delay(10);
       dht.begin();

       Serial.println("Connecting to ");
       Serial.println(ssid);


WiFi.begin(ssid, pass);

while (WiFi.status() != WL_CONNECTED)
{
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

}

void loop()
{

float h = dht.readHumidity();
float t = dht.readTemperature();

if (isnan(h) || isnan(t))
{
Serial.println("Failed to read from DHT sensor!");
return;
}
if (client.connect(server,80))   //   "184.106.153.149" or api.thingspeak.com
{

String postStr = apiKey;
postStr +="&field1=";
postStr += String(t);
postStr +="&field2=";
postStr += String(h);
postStr += "\r\n\r\n";

client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
client.print("Content-Type: application/x-www-form-urlencoded\n");
client.print("Content-Length: ");
client.print(postStr.length());
client.print("\n\n");
client.print(postStr);

Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" degrees Celcius, Humidity: ");
Serial.print(h);
Serial.println("%. Send to Thingspeak.");
}
client.stop();

Serial.println("Waiting...");

// thingspeak needs minimum 15 sec delay between updates, i've set it to 30 seconds
delay(10000);
}
```

OUTPUT:

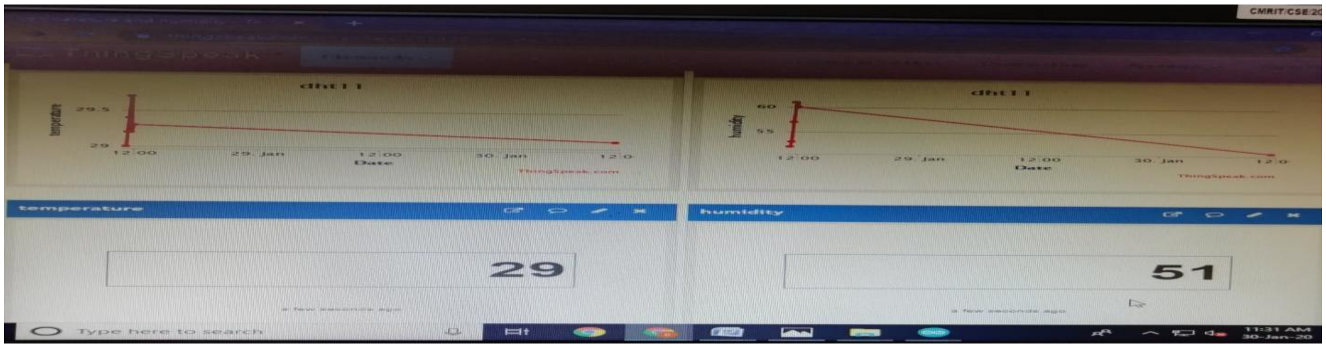


```
Waiting...
Temperature: 30.00 degrees Celcius, Humidity: 74.00%. Send to Thingspeak.
Waiting...
Temperature: 30.30 degrees Celcius, Humidity: 95.00%. Send to Thingspeak.
Waiting...
Temperature: 30.70 degrees Celcius, Humidity: 95.00%. Send to Thingspeak.
Waiting...
Temperature: 31.10 degrees Celcius, Humidity: 88.00%. Send to Thingspeak.
Waiting...
Temperature: 31.30 degrees Celcius, Humidity: 70.00%. Send to Thingspeak.
Waiting...
Temperature: 31.40 degrees Celcius, Humidity: 65.00%. Send to Thingspeak.
Waiting...
Temperature: 31.40 degrees Celcius, Humidity: 61.00%. Send to Thingspeak.
Waiting...
```

[✓] Autoscroll [ ] Show timestamp   Newline   115200 baud   Clear outp



Uploading the data in CLOUD:



Visualizing the output in Serial monitor:

WEEK-9

AIM: Write a program to create TCP Server on cloud using Arduino and Respond with humidity data to TCP Client when requested.

An IoT device can be made to communicate with a cloud or server using TCP/IP protocol without any hassle of network programming and network administration. In this project, an IoT device will be designed that could transmit sensor data to ThingSpeak Platform using the TCP/IP protocol.

The IoT device designed in this project is built using Arduino UNO. The Arduino is just a microcontroller board and cannot connect to an internet network on its own. For internet connectivity, the Arduino UNO is interfaced with ESP8266 module. The ESP8266 Wi-Fi Module is a self contained SOC with integrated TCP/IP protocol stack that can access to a Wi-Fi network. The ESP module allows the Arduino board to connect with a router and access internet network. The Arduino is programmed to communicate with the cloud platform.ThingSpeak over TCP/IP protocol. The Arduino can implement TCP/IP protocol by passing AT commands serially to the ESP8266 module.

The IoT device designed is a visitor counter as well as temperature and humidity monitor. For working as visitor counter, the IR sensors and Photodiodes are interfaced with the Arduino board. For working as temperature and humidity monitor, a DHT-11 sensor is interfaced with the Arduino board. The Arduino reads data from the sensors and send it to the ThingSpeak platform. A 0.6 inch 128 X 64 OLED is also interfaced with the Arduino which receives serial data from the board on I2C protocol and display the current temperature and humidity readings. The user can monitor the number of occupants in the house, temperature and humidity values from anywhere by accessing the ThingSpeak platform.

The Arduino board controls all the functionalities of the IoT device like counting visitors, reading temperature and humidity values from DHT-11 sensor, displaying temperature and humidity data on OLED, implementing TCP/IP protocol, connecting with ThingSpeak platform and sending data to the cloud server.

For this, the Arduino code is written and compiled using Arduino IDE.

Software Required –

• ThingSpeak server
• Arduino IDE

The IoT device that communicates with the ThingSpeak Cloud is built on Arduino UNO. The DHT-11 Sensor, IR sensor, Photodiodes, ESP8266 module and OLED module are interfaced with the Arduino board to make the IoT device.

Arduino: It is an Atmega 328 based controller board which has 14 GPIO pins, 6 PWM pins, 6 Analog inputs and on board UART, SPI and TWI interfaces. The Atmega 328 is the sitting MCU on the Arduino board. There are two GPIO pins (external interrupt pins INT0 and INT1) of Arduino board used to interface photodiodes, TX and RX pins (pins 3 and 2 respectively)are used interface ESP module, SDA and SCL (pins 27 and 28 respectively) pins used to interface the OLED module.

OLED Module – The OLED module is used to display the temperature and humidity information as well as the count of the visitors (occupants) in the house. The module communicates with the Arduino board using I2C protocol. This is a two wire protocol. The module has four pins – SDA, SCL, VCC and GND. The VCC and Ground are connected to 5V DC and common ground respectively. The 5V DC can be supplied through a battery via 7805 voltage regulator IC. The SDA and SCL pins of the OLED module are connected to the SDA and SCL pins ((pins 27 and 28 respectively) of the Arduino board. The OLED are made up of carbon based organic materials. So unlike LCD displays, they do not require backlight and filters.

ESP8266 Module – The ESP8266 Wi-Fi Module is a self contained SOC with integrated TCP/IP protocol stack that can access to a Wi-Fi network. The ESP8266 is capable of either hosting an application or off loading all Wi-Fi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an AT command set firmware. The Chip Enable and VCC pins of the module are connected to the 3.3 V DC while Ground pin is connected to the common ground. The chip enable pin is connected to VCC via a 10K pull up resistor. The RESET pin is left not connected. The Tx and Rx pins of the module are connected to the RX and TX pins of the Arduino UNO. The GPIO-0 pin of the module is connected to VCC through a 10K pull up resistor.

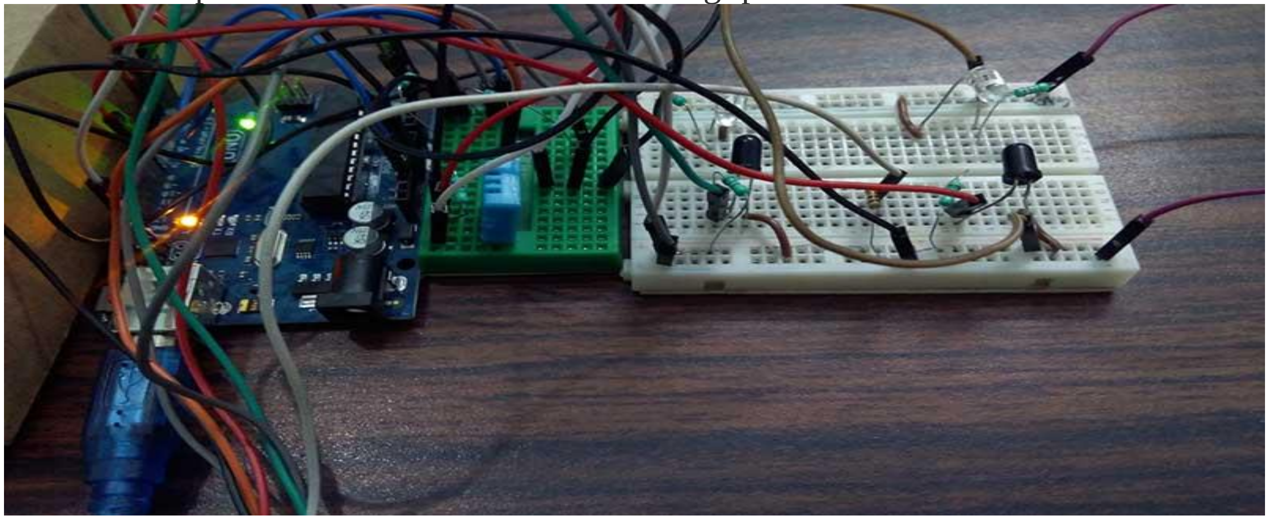
IR Sensors – An array of two IR LEDs is used in the circuit to detect entrance of any visitor. These IR LEDs are installed at the entrance of the house. An IR LED is a type of LED which emits light in the Infra-Red frequency range. The infra-red radiations are not visible to the human eye but can be seen by the lenses of a camera. Operationally, IR LEDs are not much different from normal LEDs. They also need a 3V DC for biasing and consume 20 mA current. They also need to be connected with pull-up resistor in a circuit. In the array, IR LEDs are connected with 220 ohm pull-up resistors.

Photodiodes – The photodiodes are used as IR receivers in the circuit. A photodiode is a type of diode which gets forward biased when light is incident on it. It has a high resistance when no light is falling on it. When the intensity of light incident on it increases, it starts getting forward biased and current starts flowing through it. So, when light is incident on it, its resistance decreases and there is less voltage drop across it. When light is not incident on it, its resistance increases and there is higher voltage drop across it. The photodiode looks exactly like an LED and may have a dark blue or black film on the outer casing. The photodiodes are used in reverse bias configuration in the circuit. An array of two photodiodes installed in line with IR transmitters is used in the circuit. The photodiodes are interfaced at external interrupt pins INT0 and INT1 of the Arduino board.

DHT-11 Sensor – DHT-11 is a temperature and humidity sensor. The DHT11 sensor consists of two main components – one is Humidity sensing component and other is NTC temperature sensor (or Thermistor). The Thermistor is actually a variable resistor that changes its resistance with change in temperature. They both sense the temperature and humidity of area and give the output to the IC (which is placed on back side of sensor). The sensor has four pins – VCC, Ground, data Out and NC. The VCC and Ground pins are connected to the common VCC and Ground respectively. The Data Out pin of the sensor is connected to PD7 pin of the Arduino board via 10K pull-up resistor.

ThingSpeak Server – The ThingSpeak server is used to visualize the data received from the IoT device. The data is displayed in the form of graphs on the platform. The ThingSpeak generates the read and write API key. The Write API key is used to write the data to the channel and read API channel is used to allow other people to view private channel feeds and charts. The data can also be saved on the platform for future reference.

In order to configure the ThingSpeak platform to access the data on it, first an **account** must be created on the platform. Then a channel must be created for the data on that account. It can be done by navigating to **channel window** and creating a new channel. The required information must be filled in the given form at the website so that the needed fields are created. For this project, there must be created three fields – Total Persons, Temperature and Humidity. These fields can then be checked live on the server. After saving the channel settings, a Write API key is generated which must be noted down. This Write API key is used in the firmware code of the Arduino to access the private channel created on the ThingSpeak account.



As the circuit is powered on the Arduino board starts reading data from the IR receivers and the DHT-11 sensor. The IR receivers are connected in series with variable resistors between VCC and ground in reverse bias configuration forming a voltage divider circuit. The output from the IR receivers (photodiodes) are drawn from the junction of cathode terminals of the IR receiver. Under normal conditions, the light emitted by the IR transmitters is continuously received by the photodiodes. This keeps the digital logic output of the photodiodes to HIGH. When a person enters the house, the light from the IR transmitters is blocked and the logical output from the photodiodes is switched to LOW. The IR receivers are connected at the external interrupt pins of the Arduino, so an interrupt is generated at the INT0 and INT1 pin of the Arduino when a person enters or exits the house. There are two pairs or IR transmitter and receiver used. The sequence in which the interrupts are generated at the INT0 and INT1 pins indicate whether a person has entered or exit from the house. Accordingly, the count of the current occupants of the house is increased or decreased.

The DHT11 Temperature and Humidity Sensor is a digital sensor with inbuilt capacitive humidity sensor and Thermistor. It relays a real-time temperature and humidity reading every 2 seconds. The sensor operates on 3.5 to 5.5 V supply and can read temperature between 0° C and 50° C and relative humidity between 20% and 95%. The DHT11 detects water vapors by measuring the electrical resistance between the two electrodes. The humidity sensing component is a moisture holding substrate with electrodes applied to the surface. When water vapors are absorbed by the substrate, ions are released by the substrate which increases the conductivity between the electrodes. The change in resistance between the two electrodes is proportional to the relative humidity. Higher relative humidity decreases the resistance between the electrodes, while lower relative humidity increases the resistance between the electrodes.

The same data is passed to the ThingSpeak server every three seconds by accessing the server through Write API key. The data is passed over TCP/IP protocol by sending AT commands to the ESP module through serial communication. The data passed for all the three fields is updated every three seconds and is displayed in the form of graphs on the server.

CODE:

```
#include "U8glib.h"
#include "DHT.h"
U8GLIB_SSD1306_128X64 u8g(U8G_I2C_OPT_NONE|U8G_I2C_OPT_DEV_0);

const int PersonIn = 2;                //Variable to hold Incoming Entry of person

const int PersonOut = 3;               //Variable to hold Outgoing Entry of person
```

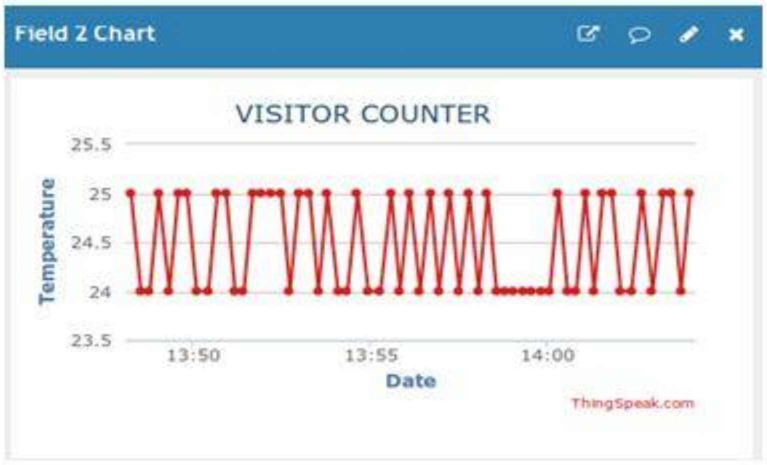
```
volatile unsigned int count = 0;         //Variable to hold the count value

int temperature;                         //Variable to hold Temperature

int humidity;                            //Variable to hold Humidity

String tempC;                            //Variable to hold the character value


char str[10];                            //Array Initialization to hold the String


// DHT11 or DHT22 pin and Type declaration

#define DHTPIN 7                         //DHT11 ouptput connected to Digital Pin 7

#define DHTTYPE DHT11                    //DHT TYPE-DHT 11 or 22

DHT dht(DHTPIN, DHTTYPE);                //DHTPIN and TYPE declaration


//ESP866-01 WIFI CONNECTION

#define SSID "NOT UR WIFI"               //"SSID-WiFiname"

#define PASS "marketing"                 //"password"

#define IP "184.106.153.149"             //thingspeak.com ip

String msg = "GET /update?key=03JHV63FG27FYL3R";      //API KEY of our channnel


//Initialise the variable in setup Function

void setup()

{

  Serial.begin(115200);             //Initialise Serial communication at 115200 bps

  Serial.println("AT");             //When baud rate matches, Print AT on Serial Monitor

  delay(5000);
  //Condition to check Whether Serial finds Ok response from ESP8266 or not

  //If It finds Ok response the call the connectWifi() Function

  if(Serial.find("OK"))

  {

    connectWiFi();

  }


  pinMode(PersonIn, INPUT);              //Initialise the Digital Pin-2 as Input

  pinMode(PersonOut, INPUT);              //Initialise the Digital Pin-3 as Input



  attachInterrupt(digitalPinToInterrupt (2), PERSONIN, FALLING);     //Attach Interrupt
handler for D2

  attachInterrupt(digitalPinToInterrupt  (3),  PERSONOUT,  FALLING);            //Attach
interrupt handler for D3


  //Initialize Timer1 Function

  timer1_init();



  //Display Welcome messgage on OLED

  OLED_Welcome_Message();

}


//Loop function

void loop()

{

  // check whether the flag bit is set
```

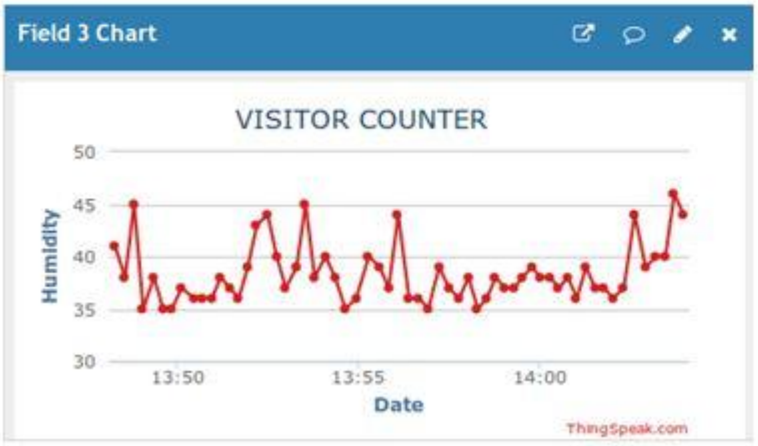
```
    // if set, it means that there has been a compare match
    // and the timer has been cleared
    // use this opportunity to update the data on OLED and Thingspeak
  if (TIFR1 & (1 << OCF1A))
  {
    //Get the temperature and Humidity value from DHT11 Sensor
      Temperature_and_Humidity_Test();


    //Update the Temperature, Humidity and Total Person on Server(Thingspeak)
      update_Data_To_Server();


      //Display the Temperature, Humidity and Total Person on OLED
      Display_Data_on_OLED();
    }
    // wait! we have one more thing to do
    // clear the flag bit manually since there is no ISR to execute
    // clear it by writing '1' to it (as per the datasheet)
    TIFR1 |= (1 << OCF1A);
}

/*
* Function Name - OLED_Welcome_messgage
* To diplay the text when OLED starts
* Input Paraameter - void
* Return - void
*/
void OLED_Welcome_Message(void)
{
  u8g_uint_t x = 0;
  u8g_uint_t y = 20;
  const char *s = "VISIOR COUNTER!";
  //Draw the string "VISITOR COUNTER!" on OLED with given reference points
  u8g.firstPage();
  do
  {
    u8g.setFont(u8g_font_unifont);
    u8g.drawStr( x, y,s );
  }
  while( u8g.nextPage() );
}

/*
* External Interrupt Function INT0- PERSONIN
* To count the person coming in the room
*/
void PERSONIN()
{
  count++;
}

/*
* External Interrupt Function INT1- PERSONOUT()
* To count the person coming out of the room
*/
void PERSONOUT()
{
  count--;
  if (count <= 0)
  count = 0;
}

/*
*Function Name - Temperature_and_Humidity_Test()
*Read the Temperature and Humidity value from DHT sensor
*/
void Temperature_and_Humidity_Test()
{
  //Read the Temperature and Humidity from DHT sensor
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();
  char buffer[10];
  //There is a useful c function called dtostrf() which will convert a float to a char array so it
can then be printed easily.

  //The format is: dtostrf(floatvar, StringLengthIncDecimalPoint, numVarsAfterDecimal,
charbuf);

  tempC = dtostrf(temperature, 4, 1, buffer);

}


/*

*Function Name= Display_Data_on_OLED()

*To display the data on OLED with paticular format

*/

void Display_Data_on_OLED()

{

  //To Display on OLED

  u8g.firstPage();

  do

  {

    //Set the font of display

    u8g.setFont(u8g_font_helvB08);



    //Draw the string "Total Person" with specified reference points(x,y)

    u8g.drawStr( 0, 15, "TOTAL Person:");

    //set the count value to OLED specied points

    u8g.drawStr( 80, 15, dtostrf(count, 5, 2, str));
    //Draw the string "Humidity" with specified reference points(x,y)
    u8g.drawStr( 0, 30, "Humidity:");
    //set the Humidity value to OLED specied points
    u8g.drawStr( 80, 30, dtostrf(humidity, 5, 2, str));
    u8g.drawStr( 120, 30, "%");

    //Draw the string "Temperature" with specified reference points(x,y)
    u8g.drawStr( 0, 45, "Temperature:");
    //set the Temperature value to OLED specied points
    u8g.drawStr( 80, 45, dtostrf(temperature, 5, 2, str));
    u8g.drawStr( 120, 45, "260C");
  }
  while( u8g.nextPage() );
}

/*
* Function Name- update_Data_To_Server()
* Establish TCP connection and send the updated Data to Server(Thingspeak)
*/
void update_Data_To_Server()
{
   //Specify which connection channel you wish to connect on ( 0 - 4 ),
   //the protocol type (TCP/UDP),the IP address (or domain if you have DNS access)
```

```
//and the port number using the CIPSTART command:

String cmd = "AT+CIPSTART="TCP","";

cmd += IP;

cmd += "",80";

Serial.println(cmd);


if(Serial.find("Error"))

 {

   return;

 }

cmd = msg ;

cmd += "&field1=";                    //Creates field1 for Total Person in the Room

cmd += String(count);                 //Upload count data to thingspeak

cmd += "&field2=";                  //Creates field2 for Temperature

cmd += tempC;                       //Upload Temperature data to Thingspeak

cmd += "&field3=";                      //Creates field3 for Humidity

cmd += String(humidity);              //Upload Humidity data to Thingpeak

cmd += "rnrn";

//Send Data Length

Serial.print("AT+CIPSEND=");

Serial.println(cmd.length());
```

```
  //If Everything works ok, then we get ">"
  if(Serial.find(">"))
  {
    //Serial monitor prints the Get command with the API key and fields speacified
    Serial.print(cmd);
  }
  else
  {
   //close the TCP connection
    Serial.println("AT+CIPCLOSE");
    Temperature_and_Humidity_Test();
    count;
  }
}

/*
*Function Name- connectWifi()
*It establishes the connection between the ESP8266 and Wifi
*Input paramerter-void
*Return type- boolean
*/
boolean connectWiFi()
{
  //Enable the module to act as both Station and an access point
  Serial.println("AT+CWMODE=1");
  delay(2000);

  //Join the WIFI access point
  String cmd="AT+CWJAP=\"";
  cmd+=SSID;
  cmd+="\",\"";
  cmd+=PASS;
  cmd+="\"";

  //Print the Wifi SSID and Password on Serial monitor
  Serial.println(cmd);
  delay(5000);

  if(Serial.find("OK"))
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

/*


Field 2 Chart
VISITOR COUNTER
Temperature
25.5
25
24.5
24
23.5
13:50
13:55
14:00
Date
ThingSpeak.com

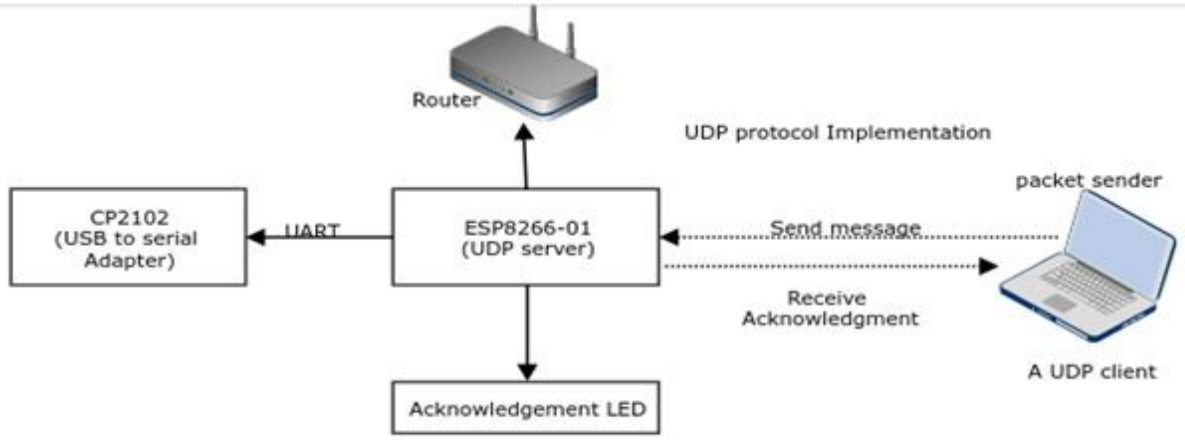

Field 3 Chart
VISITOR COUNTER
Humidity
50
45
40
35
30
13:50
13:55
14:00
Date
ThingSpeak.com

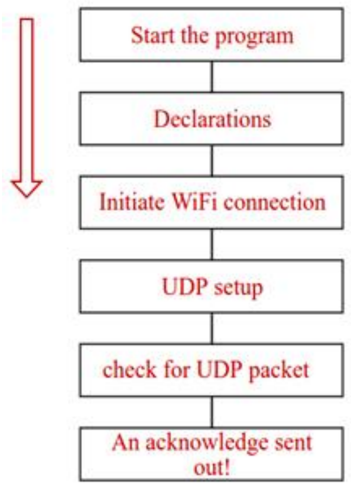
WEEK-10

AIM: Write a program to create UDP Server on cloud using Arduino and Respond with humidity data to UDP Client when requested.

The UDP protocol has a small overhead of 8 bytes which makes it more suitable for use in the Internet of Things. In this project, the application of UDP protocol in IoT will be demonstrated. In this project, an ESP8266 Wi-Fi modem will be configured as UDP server and a laptop will be used as UDP Client. Both Client and server will be co-located communicating through same Wi-Fi router so, the ESP board will act as a local serverThe ESP module working as server checks for the UDP packet received from the client on a particular port. When a valid packet is arrived, an acknowledge packet is sent back to the client to the same port it has been sent out. In response to receiving packet and sending acknowledgement, the ESP modem switches on an LED as visual indication of successful Client-Server Communication. This application is very useful as it demonstrates server/client communication over UDP protocol.

It can be said that the IOT device designed in this experiment is a simple UDP server with LED indicator. It is designed by interfacing an LED with the ESP-8266 Wi-Fi module. The Wi-Fi module as well as LED light are powered continuously with the help of a USB to Serial Converter. The Wi-Fi module needs to loaded with a firmware that could receive data over UDP protocol and respond with an acknowledgement. The Arduino UNO is used to flash the firmware code on the ESP8266 module. The ESP module can also be flashed with code using a FTDI converter like CP2102. The firmware itself is written in the Arduino IDE.





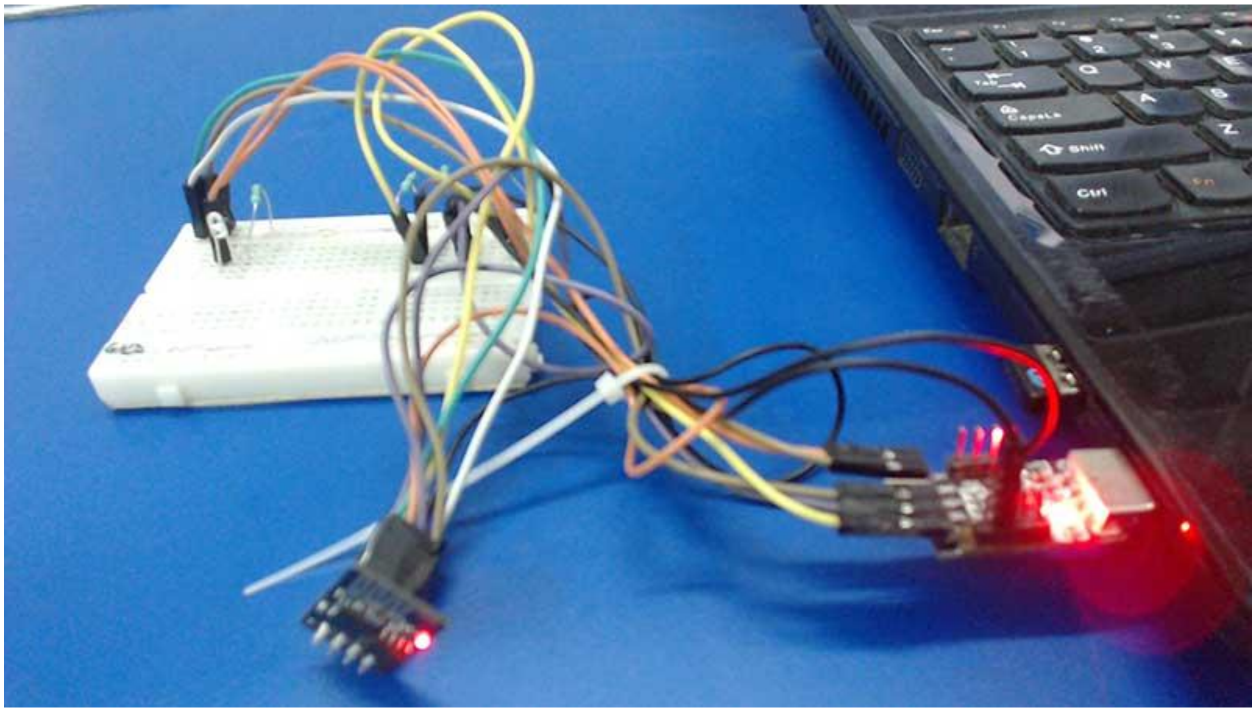
Software Required –

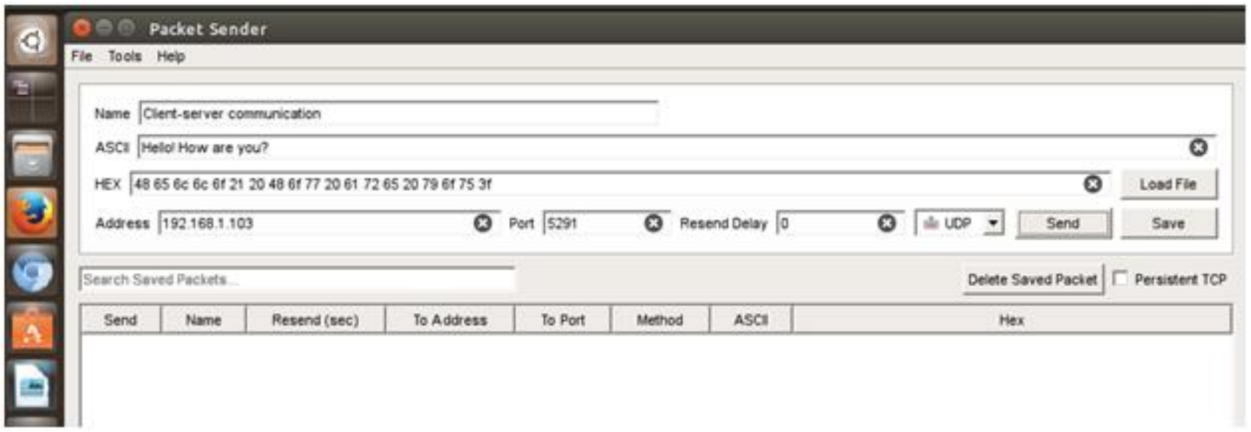
• ThingSpeak server

• Arduino IDE

ESP8266 board needs to be loaded with the firmware code. In this tutorial, the firmware code is written using Arduino IDE. It is loaded to the ESP8266 board using the Arduino UNO. A generic ESP8266 board is used in this project. This board does not have any bootstrapping resistors, no voltage regulator, no reset circuit and no USB-serial adapter. The ESP8266 module operates on 3.3 V power supply with current greater than or equal to 250 mA. So, CP2102 USB to serial adapter is used to provide 3.3 V voltage with enough current to run ESP8266 reliably in every situation.
The ESP8266 Wi-Fi Module is a self contained SOC with integrated TCP/IP protocol stack that can access to a Wi-Fi network. The ESP8266 is capable of either hosting an application or off loading all Wi-Fi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an AT command set firmware.

The Chip Enable and VCC pins of the module are connected to the 3.3 V DC while Ground pin is connected to the common ground. The chip enable pin is connected to VCC via a 10K pull up resistor. The RESET pin is connected to the ground via a tactile switch where the pin is supplied VCC through a 10K pull up resistor by default. The Tx and Rx pins of the module are connected to the RX and TX pins of the Arduino UNO. The GPIO-0 pin of the module is connected to ground via a tactile switch where the pin is supplied VCC through a 10K pull up resistor by default. Write the firmware code in the Arduino IDE and connect the Arduino board with the PC via USB cable. Open Arduino IDE and go to Tools->Port and select the Arduino board (Arduino UNO). It may look like /dev/ttyABM0 (Arduino/Genuino Uno). Select the correct port name. The port name can be different in different IDE setups. Then Open Serial monitor in the Arduino IDE by navigating to Tools->Serial Monitor and set the baud rate to 115200 bauds per second. Pass 'AT' and 'AT+GMR' commands to test the connection between the Arduino and ESP-01 module. Try different settings for the 'Line ending' option of the serial monitor like Both NL & CR. Try different combinations until the ESP module starts interacting correctly with the serial monitor.





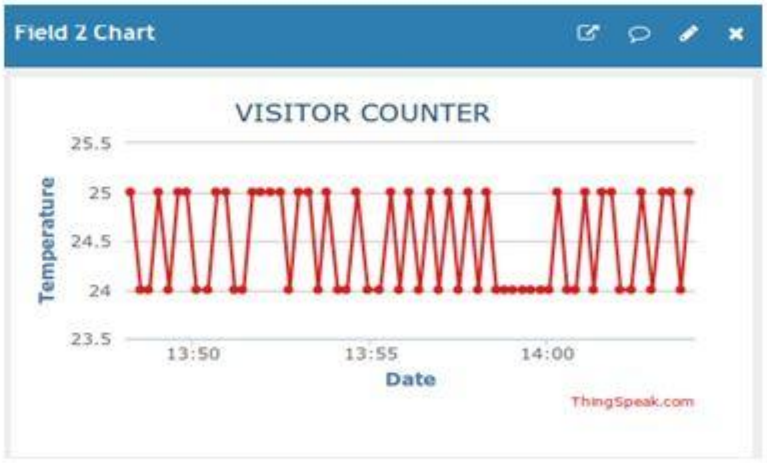
Name: Client-server communication
ASCII: Hello! How are you?
HEX: 48 65 6c 6c 6f 21 20 48 6f 77 20 61 72 65 20 79 6f 75 3f
Address: 192.168.1.103 Port: 5291 Resend Delay: 0 UDP
Persistent TCP: ☐

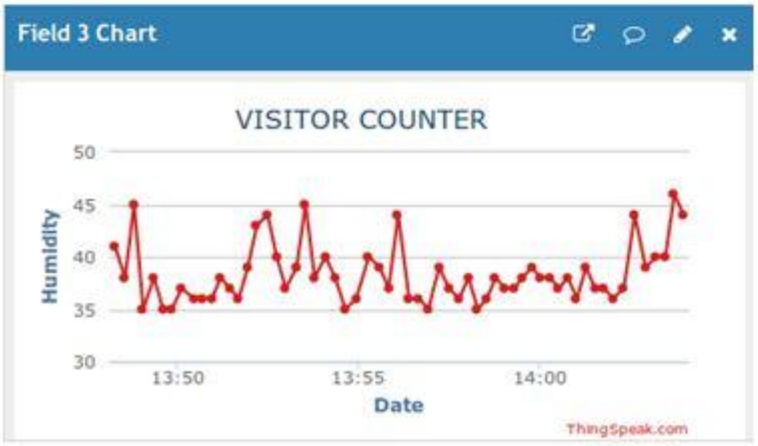
Send	Name	Resend (sec)	To Address	To Port	Method	ASCII	Hex

The ESP modem act as UDP server in this setup. It is loaded with a firmware that can receive data on UDP protocol and in response send back an acknowledgement on the same port. The Laptop which is acting as UDP Client sends the datagram using packet sender application. The firmware code on the ESP modem light up an LED after successful communication with the UDP Client.

In the firmware code of the ESP8266 based UDP server, the Wi-Fi connection is initiated with the available router and the UDP connection is setup using standard library functions. A library named WifiUDP.h is imported in the ESP code for managing the UDP connection and receiving data over it. Once the UDP packet is successfully received, the code sends an acknowledgement on the same port and set the LED connected pin to HIGH.

```
#include "DHT.h"

int status_led = 2;

WiFiUDP Udp;

unsigned int localUdpPort = 5219;  // local port to listen on

char ReceivedMessage[255];  // buffer for incoming packets

char  Acknowledge[] = "Received your message: ";  // a reply string to send back

void setup()

{

  Serial.begin(115200);

  pinMode(status_led, OUTPUT);

  Serial.println();

  Serial.printf("Connecting to %s ", ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)

  {

    delay(500);

    Serial.print(".");

  }

  Serial.println("Wi-Fi connected!");

  Udp.begin(localUdpPort);
  Serial.printf("I am listening at IP %s, UDP port %dn", WiFi.localIP().toString().c_str(), localUdpPort);

}

void loop()

{

  int packetSize = Udp.parsePacket();

  if (packetSize)

  {

    // receive incoming UDP message

Serial.printf("Received %d message from %s, port %dn", packetSize, Udp.remoteIP().toString().c_str(), Udp.remotePort());

    int len = Udp.read(ReceivedMessage, 255);

    if (len > 0)

    {

      ReceivedMessage[len] = 0;

    }

    Serial.printf("UDP message: %sn", ReceivedMessage);

    digitalWrite(status_led, HIGH);

    delay(1000);

    digitalWrite(status_led, LOW);

    // send back a reply, to the IP address and port we got the packet from

    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());

    Udp.write(Acknowledge);

    Udp.endPacket();
  }
```


Field 2 Chart
VISITOR COUNTER
25.5
25
24.5
24
23.5
Temperature
13:50
13:55
14:00
Date
ThingSpeak.com


Field 3 Chart
VISITOR COUNTER
50
45
40
35
30
Humidity
13:50
13:55
14:00
Date
ThingSpeak.com