1. **INTRODUCTION**

**1.1 AIM OF THE PROJECT**

Wireless sensor network (WSN) refers to a group of spatially dispersed and dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location. WSNs measure environmental conditions like temperature, sound, pollution levels, humidity, wind, and so on.

This project helps to sense, notify and report the temperature changes in a metered location, even if there a slightest change in the temperature in two or more different places this will track the changes. In this project I used some modules which can sense the temperature with in 1KM range which costs less than ₹3,000.

**1.2 SCOPE OF THE PROJECT**

To sense, notify and report the temperature fluctuations in a metered location. We can help many factories, warehouses, market storage warehouses, etc., to monitor the temperature and in their place in order to maintain a constant temperature in their premises.

In recent years the advances in microchip design have given birth to the possibility of very lightweight sensor nodes fabrication. These sensor nodes can be deployed in a variety of monitoring applications such as in agricultural farmlands, health-care systems and disaster prone regions etc. Most often, the bottleneck of deploying wireless sensor networks lies on the management of the battery life. The limitation of the battery capacity in WSNs could make it expensive and difficult to deploy on a large scale. Thus, developing a protocol that can optimize the energy consumption has been the focused.

**1.3 DEFINITIONS, ACRONYMS**

* **ARDUINO:**

[Arduino](http://arduino.cc/) is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a [microcontroller](http://en.wikipedia.org/wiki/Microcontroller)) and a piece of [software](http://arduino.cc/en/Main/Software), or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. The Arduino platform has become quite popular with people just starting out with electronics, and for good reason.

Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.



Fig. 1.3.0

The Arduino hardware and software was designed for artists, designers, hobbyists, hackers, newbies, and anyone interested in creating interactive objects or environments. Arduino can interact with buttons, LEDs, motors, speakers, GPS units, cameras, the internet, and even your smart-phone or your TV! This flexibility combined with the fact that the Arduino software is free, the hardware boards are pretty cheap, and both the software and hardware are easy to learn has led to a large community of users who have contributed code and released instructions for a **huge** variety of Arduino-based projects.

* **NodeMCU**:

NodeMCU is an open source of Internet of things platform. It includes firmware, which runs on the ESP8266 Wi-Fi SoC from Embedded System, and hardware, which is based on the ESP-12 module. The term "NodeMCU" by default refers to the firmware rather than the dev kits. The firmware uses the C++ scripting language. It uses many open source projects.

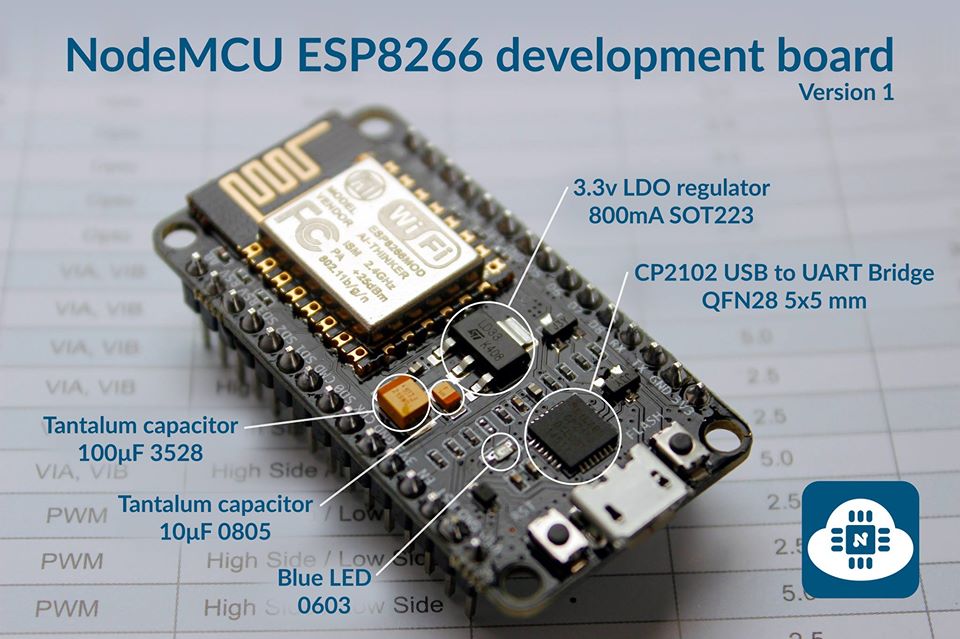
****

Fig. 1.3.1

* **ARDUINO UNO or NANO**:

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.x) or ATmega168 (Arduino Nano 2.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. The Arduino Nano can be powered via the Mini-B USB connection, 6-20V unregulated external power supply (pin 30), or 5V regulated external power supply (pin 27). The power source is automatically selected to the highest voltage source.

**Features**

* **Microcontroller:** Atmel ATmega168 or ATmega328
* **Operating Voltage (logic level):** 5 V
* **Input Voltage(recommended):** 7-12 V
* **Input Voltage(limits):** 6-20 V
* **Digital I/O Pins:** 14 (of which 6 provide PWM output)
* **Analog Input Pins:** 8
* **DC Current per I/O Pin:** 40 mA
* **Flash Memory:** 16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
* **SRAM:** 1 KB (ATmega168) or 2 KB (ATmega328)
* **EEPROM:** 512 bytes (ATmega168) or 1 KB (ATmega328)
* **Clock Speed:** 16 MHz

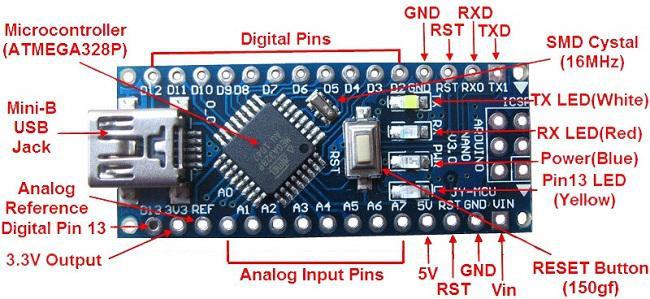


Fig 1.3.2

**Parts description of Arduino Nano 328P**

**Input and Output:**

There are totally 14 digital Pins and 8 Analog pins on your Nano board. Using them as input pins or drive loads we can use the digital pins to interface sensors by using them as output pins. A simple function like pinMode() and digitalWrite() can be used to control their operation. The operating voltage is 0V and 5V for digital pins. The analog pins can measure analog voltage from 0V to 5V using any of the 8 Analog pins using a simple function liken analogRead()

* **Serial Pins 0 (Rx) and 1 (Tx):** Rx and Tx pins are used to receive and transmit TTLserial data. They are connected with the corresponding ATmega328P USB to TTL serial chip.
* **External Interrupt Pins 2 and 3:** These pins can be configured to trigger an interrupt ona low value, a rising or falling edge, or a change in value.
* **PWM Pins 3, 5, 6, 9 and 11:** These pins provide an 8-bit PWM output by usinganalogWrite() function.
* **SPI Pins 10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK):** These pins are used for SPIcommunication.
* **In-built LED Pin 13:** This pin is connected with a built-in LED, when pin 13 is HIGH –LED is on and when pin 13 is LOW, its off.
* **I2C A4 (SDA) and A5 (SCA):** Used for IIC communication using Wire library.
* **AREF:** Used to provide reference voltage for analog inputs with analogReference()function.
* **Reset Pin:** Making this pin LOW, resets the microcontroller.
* **HC-12:**

HC-12 wireless serial port communication module is a new-generation multichannel embedded wireless data transmission module. Its wireless working frequency band is 433.4-473.0MHz, multiple channels can be set, with the stepping of 400 KHz, and there are totally 100 channels. The maximum transmitting power of module is 100mW (20dBm), the receiving sensitivity is -117dBm at baud rate of 5,000bps in the air, and the communication distance is 1,000m in open space.

**Specifications:**

* Long-distance wireless transmission (1,000m in open space/baud rate 5,000bps in the air)
* Working frequency range (433.4-473.0MHz, up to 100 communication channels)
* Maximum 100mW (20dBm) transmitting power (8 gears of power can be set)
* Three working modes, adapting to different application situations
* Built-in MCU, performing communication with external device through serial port
* The number of bytes transmitted unlimited to one time

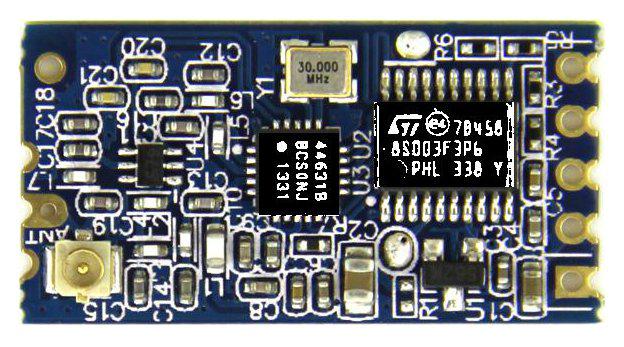


Fig. 1.3.3

* **DHT-11:**

The DHT11 is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers. The sensor can measure temperature from 0°C to 50°C and humidity from 20% to 90% with an accuracy of ±1°C and ±1%. So, I thought this will be the best sensor to include in my project.

**Specifications:**

* Long-di Operating Voltage: 3.5V to 5.5V
* Operating current: 0.3mA (measuring) 60uA (standby)
* Output: Serial data
* Temperature Range: 0°C to 50°C
* Humidity Range: 20% to 90%
* Resolution: Temperature and Humidity both are 16-bit
* Accuracy: ±1°C and ±1%

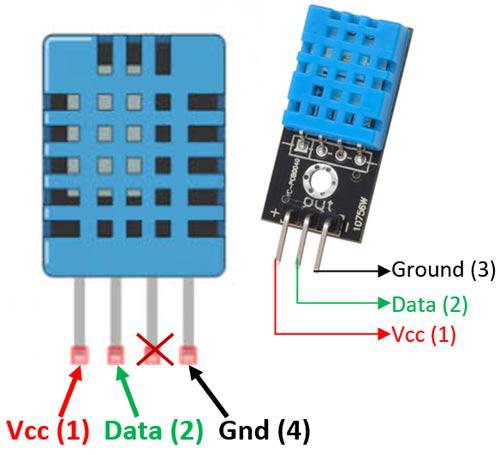


Fig. 1.3.4

* **LEACH PROTOCOL:**

The goal of LEACH is to lower the energy consumption required to create and maintain clusters in order to improve the lifetime of a wireless sensor network. LEACH is a hierarchical protocol in which most nodes transmit to cluster heads, and the cluster heads aggregate and compress the data and forward it to the base station (sink). Each node uses a stochastic algorithm at each round to determine whether it will become a cluster head in this round. LEACH assumes that each node has a radio powerful enough to directly reach the base station or the nearest cluster head, but that using this radio at full power all the time would waste energy.

Nodes that have been cluster heads cannot become cluster heads again for P rounds, where P is the desired percentage of cluster heads. Thereafter, each node has a 1/P probability of becoming a cluster head again. At the end of each round, each node that is not a cluster head selects the closest cluster head and joins that cluster. The cluster head then creates a schedule for each node in its cluster to transmit its data. All nodes that are not cluster heads only communicate with the cluster head in a TDMA (Time-Division multiple Access) fashion, according to the schedule created by the cluster head. They do so use the minimum energy needed to reach the cluster head, and only need to keep their radios on during their time slot. LEACH also uses CDMA so that each cluster uses a different set of CDMA codes, to minimize interference between clusters.

**2. SYSTEM ANALYSIS**

**2.1 PROPOSED SYSTEM**

Here in the proposed system we will have a NodeMCU ESP8266 attached to a HCU-12 and the NodeMCU ESP8266 is connected and this acts as a HEAD to the other two Arduino Nano are also connected to a DHT-11 and HCU-12 and connected to power source(<5V).

Overview:

* The NodeMCU ESP8266 is connected to a power source first, then the other two Arduino Nanos are connected simultaneously to the power source.
* We call NodeMCU ESP8266 as HEAD and Arduino Nanos as CLIENTS for easy understanding.
* First a HEAD starts transmitting the signal to the CLIENTS to transmit the TEMPERATURE in a hierarchical sequence.
* Now after receiving the signal from the HEAD to transmit the TEMPERATURE, the CLIENTs RETRANSMIT the TEMPERATURE to the HEAD in a hierarchical sequence.
* The HEAD receives the recorded TEMPERATURE from different test centers and calculates the average TEMPERATURE.
* If we are having more number of HEADS and CLIENTS to be served, we will have a main HEAD node which follows DEC (Deterministic Energy-Efficient Clustering) protocol, in which each HEAD has many CLIENTS retransmitting the data and after getting the data from CLIENTS the main HEAD sends the transmit signal to the local HEADS to send the Avg. TEMPERATURES and the main HEAD uploads the received data to a dynamic web page for storing the data and used for forecasting the weather in future.

**2.2 SYSTEM REQUIREMENTS**

**2.4.1 SOFTWARE REQUIREMENTS**

* Arduino IDE

**2.4.2 HARDWARE REQUIREMENTS**

* NodeMCU
* Arduino UNO
* Arduino NANO
* DHT-11
* HC-12
* Micro USB cable
* Power source
* Laptop

**3. SYSTEM DESIGN**

**3.1 Architecture:**

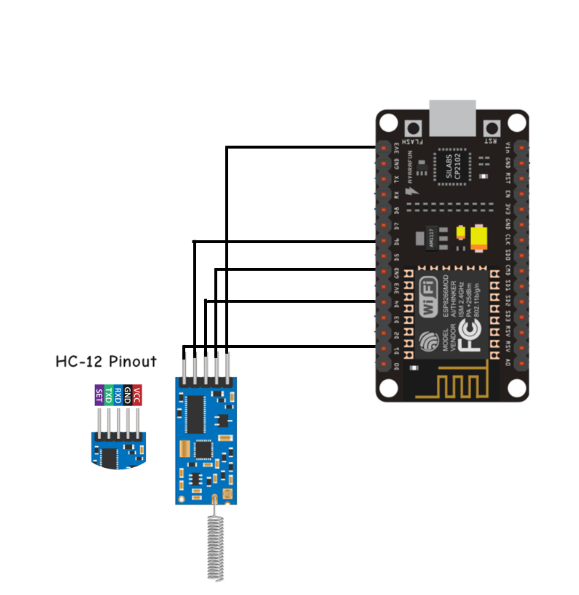


Fig. 3.1.1

HEAD

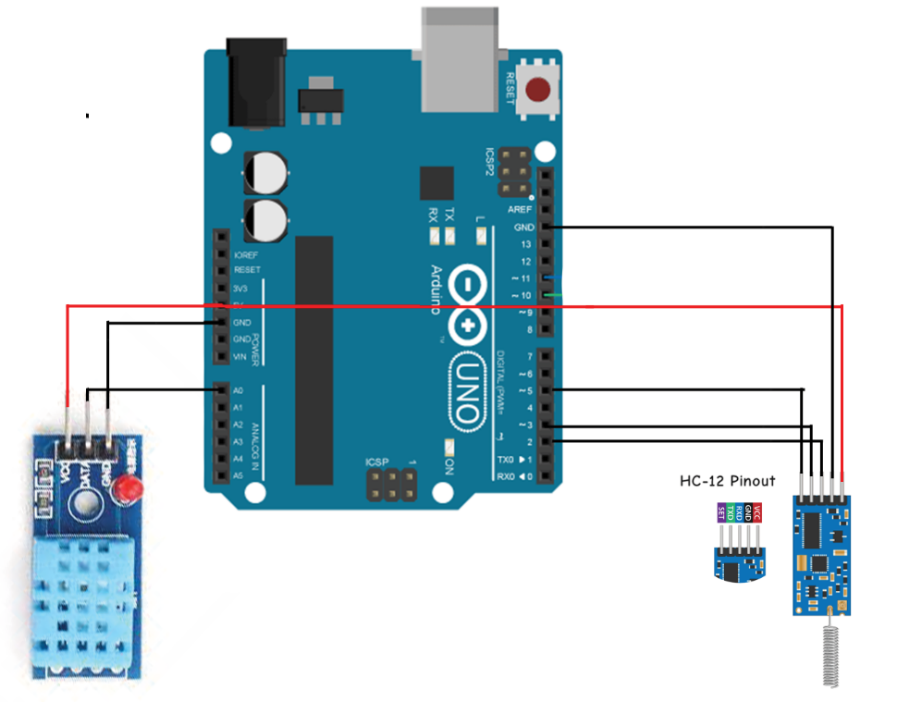


Fig. 3.1.2

CLIENT-1

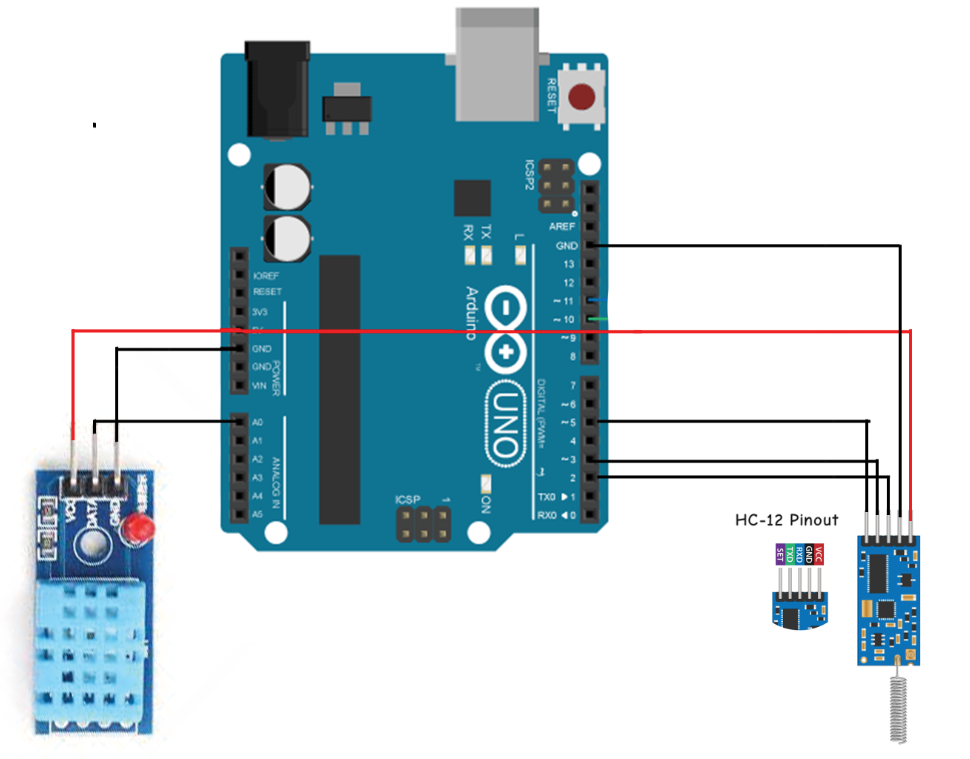


Fig. 3.1.3

CLIENT-2

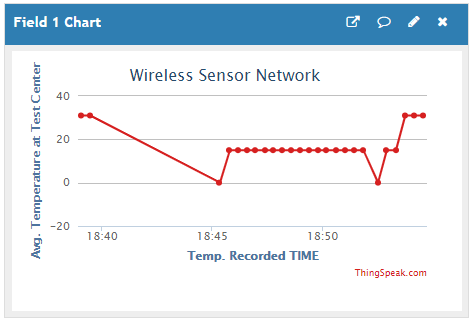


Fig.3.1.4

Values posted Dynamically

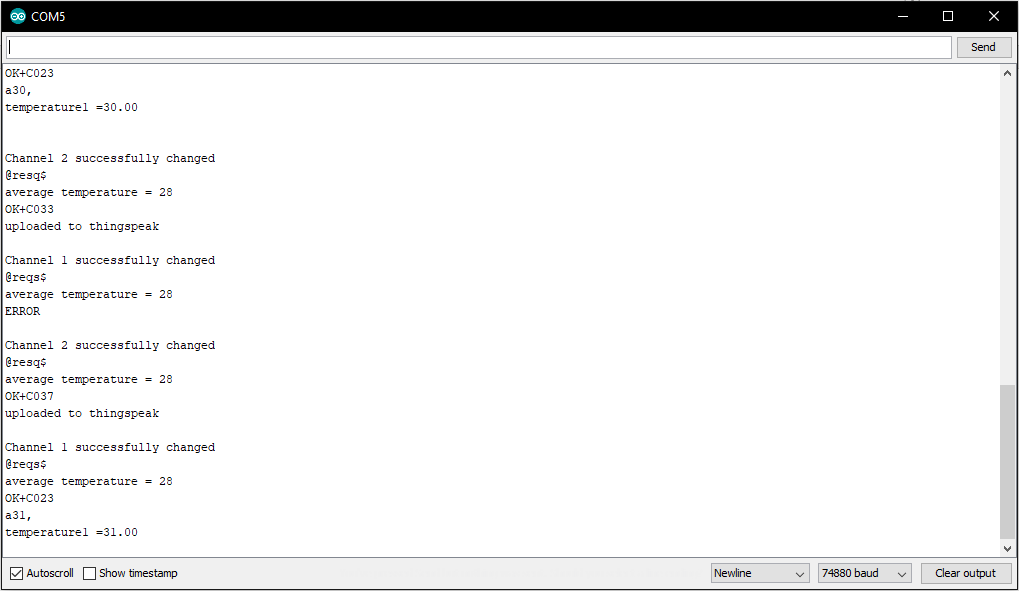


Fig. 3.1.5

HEAD SERIAL MONITOR

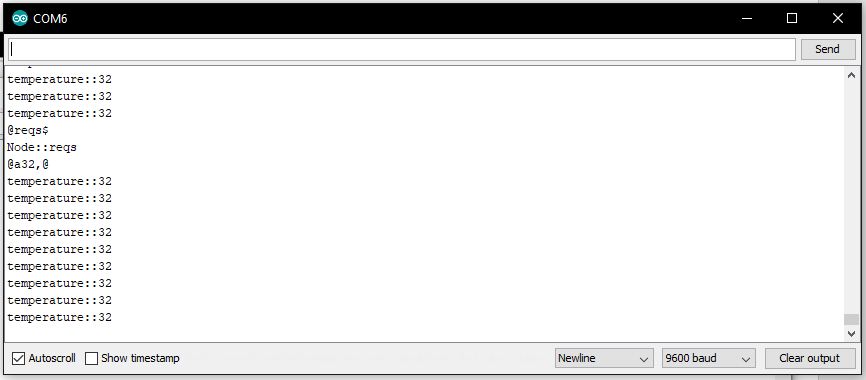


Fig. 3.1.6

CLIENT-1 SERIAL MONITOR

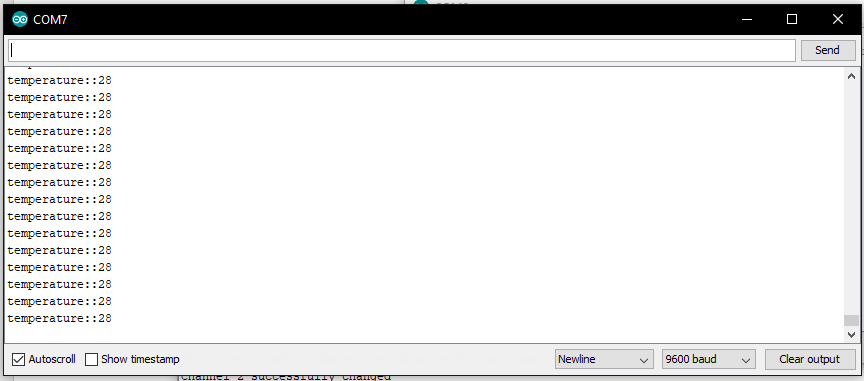


Fig. 3.1.7

CLIENT-2 SERIAL MONITOR

**4. SYSTEM IMPLEMENTATION & TESTING**

**4.1 Sample code**:

**4.1.1 Starting HEAD and transmitting signal to CLIENTS to send data:**

#define setPin 5

Ticker ticker;

SoftwareSerial HC12(12, 2); // HC-12 TX Pin, HC-12 RX Pin

String node,rd,cmd1,cmd2,c,d;

float t1,t2;

volatile int q,l;

WiFiClient client;

unsigned long myChannelNumber = 694090; //Your Channel Number (Without Brackets)

const char \* myWriteAPIKey = "JY5OXWK1T1SFPLOL"; //Your Write API Key

void setup() {

ticker.attach(1,isrFunc);

Serial.begin(74880); // Open serial port to computer

HC12.begin(9600); // Open serial port to HC12

pinMode(setPin, OUTPUT);

WiFi.begin(ssid, password);

ThingSpeak.begin(client);

}

void loop() {

val = (t1+ t2)/2 ;

if(q==6){

channel1();

delay(200);

Serial.print("average temperature = ");

Serial.println(val);

}

if(q==12)

{

channel2();

delay(200);

Serial.print("average temperature = ");

Serial.println(val);

}

if (q==18)

{

ThingSpeak.writeField(myChannelNumber, 1,val, myWriteAPIKey); //Update in ThingSpeak

Serial.println("uploaded to thingspeak");

q=0;

}

while (HC12.available()>0) { // If HC-12 has data

// Serial.write(HC12.read());

delay(500);

rd = HC12.readStringUntil(',@');

delay(500);

// Serial.write("rd==");

rd.trim();

Serial.println(rd);

node=rd.substring(0,1);

//Serial.print("Node::");

// Serial.println(node);

if (node =="a" )

{

c = rd.substring(1, 3);

// Send the data to Serial monitor

delay(500);

t1 = c.toFloat();

delay(500);

Serial.print("temperature1 =");

Serial.println(t1);

}

if (node =="b" )

{

d = rd.substring(1, 3);

delay(500);

// Send the data to Serial monitor

t2 = d.toFloat();

delay(500);

Serial.print("temperature2 ="); Serial.println(t2);

}

}

}

**4.1.2 Requesting to transmit from CHANNEL-1 or CLIENT-1:**

//CHANNEL-1

void channel1() {

digitalWrite(setPin, LOW); // Set HC-12 into AT Command mode

delay(100); // Wait for the HC-12 to enter AT Command mode

HC12.print("AT+C023"); // Send AT Command to HC-12

delay(200);

Serial.println("");

Serial.println("Channel 1 successfully changed");

digitalWrite(setPin, HIGH); // Serial port to HC12

cmd1 = "@reqs$";

delay(200);

HC12.print(cmd1);

Serial.println(cmd1);

delay(500);

}

**4.1.3 Requesting to transmit from CHANNEL-2 or CLIENT-2:**

//CHANNEL-2

void channel2() {

digitalWrite(setPin, LOW); // Set HC-12 into AT Command mode

delay(100); // Wait for the HC-12 to enter AT Command mode

HC12.print("AT+C033"); // Send AT Command to HC-12

delay(200);

Serial.println("");

Serial.println("Channel 2 successfully changed");

digitalWrite(setPin, HIGH);

delay(100);

cmd2 = "@resq$";

delay(200);

HC12.print(cmd2);

Serial.println(cmd2);

delay(500);

}

**4.1.4 Data from CLIENTS to HEAD:**

4.1.4.1 Transmitting data from CLIENT-1 to HEAD:

int t=01;

DHT dht(DHTPIN, DHT11);

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

dht.begin();// Open serial port to computer

HC12.begin(9600);

pinMode(setPin, OUTPUT);

digitalWrite(setPin, LOW); // Set HC-12 into AT Command mode

delay(100); // Wait for the HC-12 to enter AT Command mode

HC12.println("AT+C023"); // Send AT Command to HC-12 ("AT+C001")

delay(200);

if (HC12.available()>0) { // If HC-12 has data (the AT Command response)

Serial.write(HC12.read());

}

digitalWrite(setPin, HIGH);

delay(100);// Serial port to HC12

}

void loop() {

node="\0";

t = dht.readTemperature();

if(t==0)

{

t=23;

}

Serial.print("temperature::");Serial.println(t);

if (HC12.available()>0 ) {

// If HC-12 has data (the AT Command response)

rd = HC12.readStringUntil("$");

rd.trim();

//Serial.print("Rd");

Serial.println(rd);

//Serial.println(rd.length());

if(rd.startsWith("@")&&rd.endsWith("$")){

node = rd.substring(1,5);

Serial.print("Node::");Serial.println(node);

if(node=="reqs"){

cmd="@";

cmd+="a";

if(t<10)

{

cmd+="0";

}

cmd+=t;

cmd+=",@";

Serial.println(cmd);

HC12.println(cmd);

}

}

}

delay(500);

}

4.1.4.1 Transmitting data from CLIENT-2 to HEAD:

int t=01;

DHT dht(DHTPIN, DHT11);

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

dht.begin();// Open serial port to computer

HC12.begin(9600);

pinMode(setPin, OUTPUT);

digitalWrite(setPin, LOW); // Set HC-12 into AT Command mode

delay(100); // Wait for the HC-12 to enter AT Command mode

HC12.println("AT+C033"); // Send AT Command to HC-12 ("AT+C001")

delay(200);

if (HC12.available()>0) { // If HC-12 has data (the AT Command response)

Serial.write(HC12.read());

}

digitalWrite(setPin, HIGH);

delay(100);// Serial port to HC12

}

void loop() {

node="\0";

t = dht.readTemperature();

if (t==0)

{

t=32;

}

Serial.print("temperature::");Serial.println(t);

if (HC12.available()>0 ) {

// If HC-12 has data (the AT Command response)

rd = HC12.readStringUntil("$");

rd.trim();

//Serial.print("Rd");

Serial.println(rd);

//Serial.println(rd.length());

if(rd.startsWith("@")&&rd.endsWith("$")){

node = rd.substring(1,5);

Serial.print("Node::");Serial.println(node);

if(node=="resq"){

cmd="@";

cmd+="b";

if(t<10)

{

cmd+="0";

}

cmd+=t;

cmd+=",@";

Serial.println(cmd);

HC12.println(cmd);

}

}

}

delay(500);

}

**5.CONCLUSION & FUTURE SCOPE**

**Conclusion:**

In this project temperature has been measured in some remote locations within a Kilo-meter range with the help of HC-12, and DHT-11 sensor using Arduino, Networks, Wi-Fi, C++ and their handling procedures. This practical exposure will be very helpful to me in near future for developing projects independently.

**Future scope:**

In future I want to add more HEADs, CLIENTS as clusters and collect more information of the temperatures, I also wanted to add one more feature to this project to measure the humidity in a region and can be extended to predict weather forecasting.

**6.References**

* F. A. Aderohunmu, J. D. Deng, and M. K. Purvis. Enhancing Clustering in Wireless Sensor Networks with Energy Heterogeneity. International Journal of Business Data Communications and Networking.
* W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An Application-Specific Protocol Architectures for Wireless Networks. IEEE Transactions on Wireless Communications, 1:660– 670, 2002.
* G. Smaragdakis, I. Matta, and A. Bestavros. SEP: A Stable Election Protocol for clustered heterogeneous wireless sensor networks. In Proceeding of the International Workshop on SANPA, 2004.
* <https://en.wikipedia.org/wiki/Wireless_sensor_network> visited on 08-02-2019.
* <https://www.elprocus.com/introduction-to-wireless-sensor-networks-types-and-applications/> visited on 27-02-2019.
* <https://www.youtube.com/watch?v=urWv-_EqS9M> visited on 07-03-2019.
* <https://www.youtube.com/watch?v=LYjlJ2tgMbg> visited on 15-03-2019.