**Project Abstract**

The Goal of this project was to develop a Temperature monitoring system which collects temperature data from various locations to monitor at one location. The system uses a Digital Thermometer to sense Temperature. Then the microcontroller which sensor is connected sends sensor readings to nearby wireless broadcasting device via RS485 and then the data is sent through 2.4GHz ISM (Industrial, Scientific and Medical) band to monitoring station.

The programming was done in Arduino Programming Language with Arduino Software (IDE) which is open source and can be expanded through C++ libraries.

**Specifications**

**Hardware Used**

1. **DS18B20 DIGITAL THERMOMETER.**

The DS18B20 communicates over a 1-Wire bus by definition requires that only one data line (and ground) for communication with Microcontroller. Each DS18B20 has a unique 64-bit serial code, Which Allows multiple DS18B20s to function on the same 1-Wire bus. THUS, it is simple to use one microprocessor to check many DS18B20s distributed over a large area.

* Unique 1-Wire® interface requires only one port pin for communication.
* Measures temperatures from –55°C to +125°C (–67°F to +257°F).
* ±0.5°C accuracy from –10°C to +85°C.
* Each device has a unique 64-bit address stored in an onboard ROM. allows for a huge number of sensors to be used on one data bus.
* Thermometer resolution is user-selectable from 9 to 12 bits.

1. **ATMega328P 8-Bit Microcontroller**

* Running at 16MHz with external Oscillator.
* 23 general purpose I/O lines.
* 2-wire serial interface.
* Serial programmable USART.
* 32KB ISP flash memory.
* Operates between 1.8-5.5 volts.

1. **Max485 RS-485 Transceiver IC**

* Data Rate 2.5 (Mbps).
* Up to 32 Devices on the BUS.
* Half-Duplex

1. **2.4G Wireless nRF24L01+ with PA and LNA**

The nRF24L01 integrates a complete 2.4GHz RF transceiver, RF synthesizer, and baseband logic including the Enhanced ShockBurst™ hardware protocol accelerator supporting a high-speed SPI interface for the application controller. There are also PA and LNA circuit on board, with the external antenna it can reach longer distance.

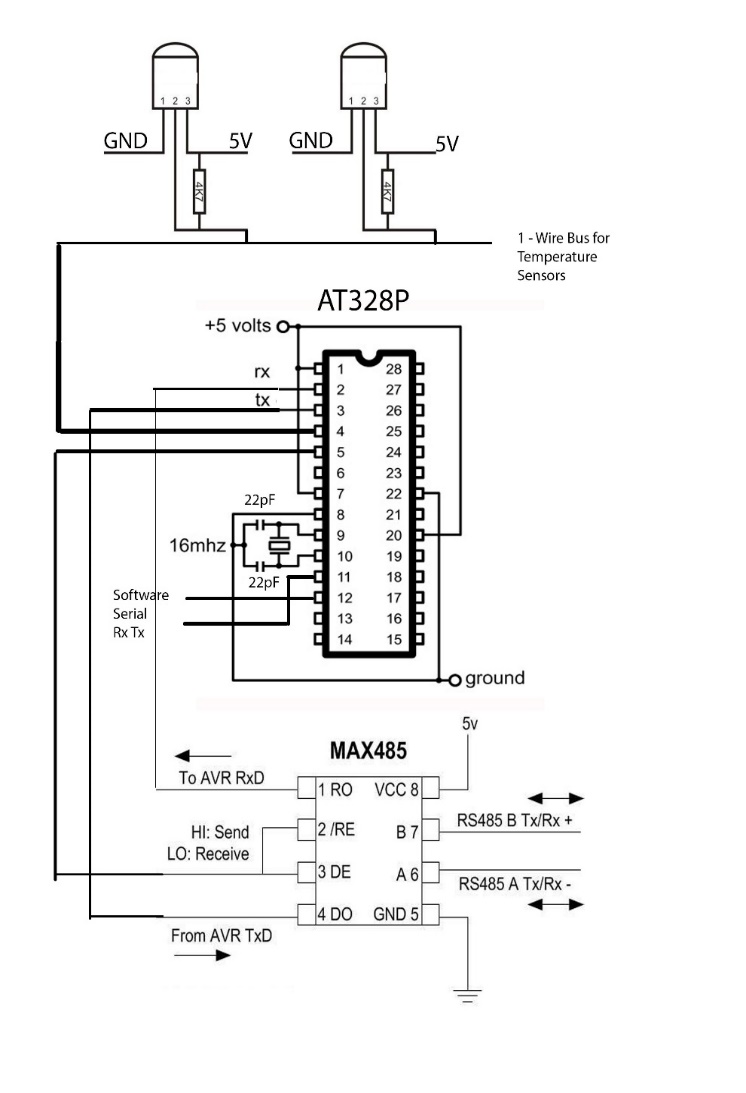
* Maximum output power +20 dBm.
* Voltage 3-3.6V (recommended 3.3V).
* Emission mode current (peak) 115 mA.
* Receive Mode Current (peak) 45 mA.
* Power-down mode current 4.2 uA.
* Operating temperature -20 to 70 ℃

**Schematics**

**Circuit diagrams**

I have used 3 circuits in this project prototype and few methods to communicate among microcontrollers. To connect sensors inside a room to microcontroller there is a 1-wire bus. To connect microcontrollers nearby I have used RS485 bus line. With that bus, collects data into a one point and from there the data will send wirelessly through nrf24l01 modules to monitoring station. By modifying the code, any of these circuits can connect to Temperature sensors with 1-wire and collect data.

1. Collecting data from sensors connected to 1-Wire bus and send that data into RS485 bus.



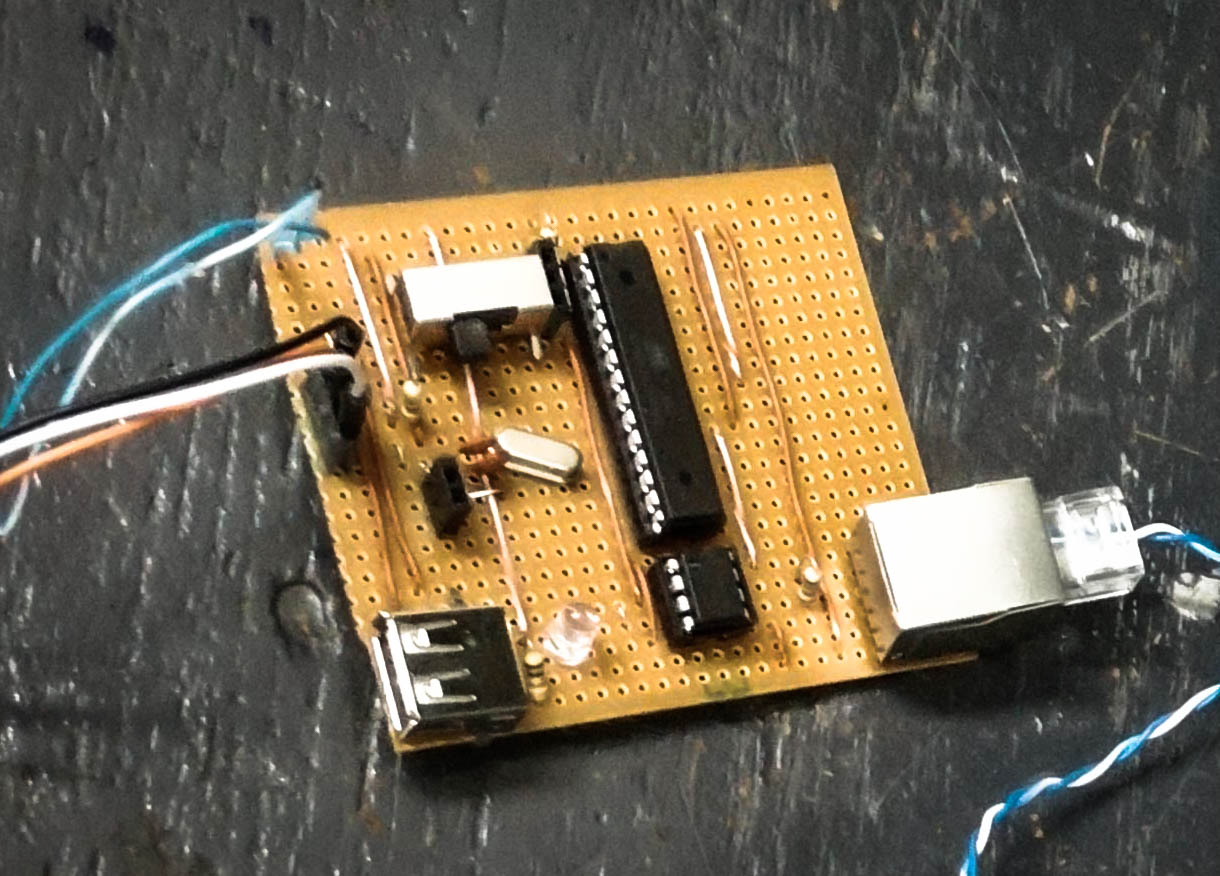
Type 01 Circuit Diagrame

Selection Switch for

Working mode and

Programming Mode (01)

Hardware Serial connection for Programming Microcontroller



Power Indicator LED

Max485 Transceiver IC

RJ 45 Connector

For RS 485 Bus

USB Interface for power supply and

Purposed serial debug through USB interface

Software Serial output for debugging purposes

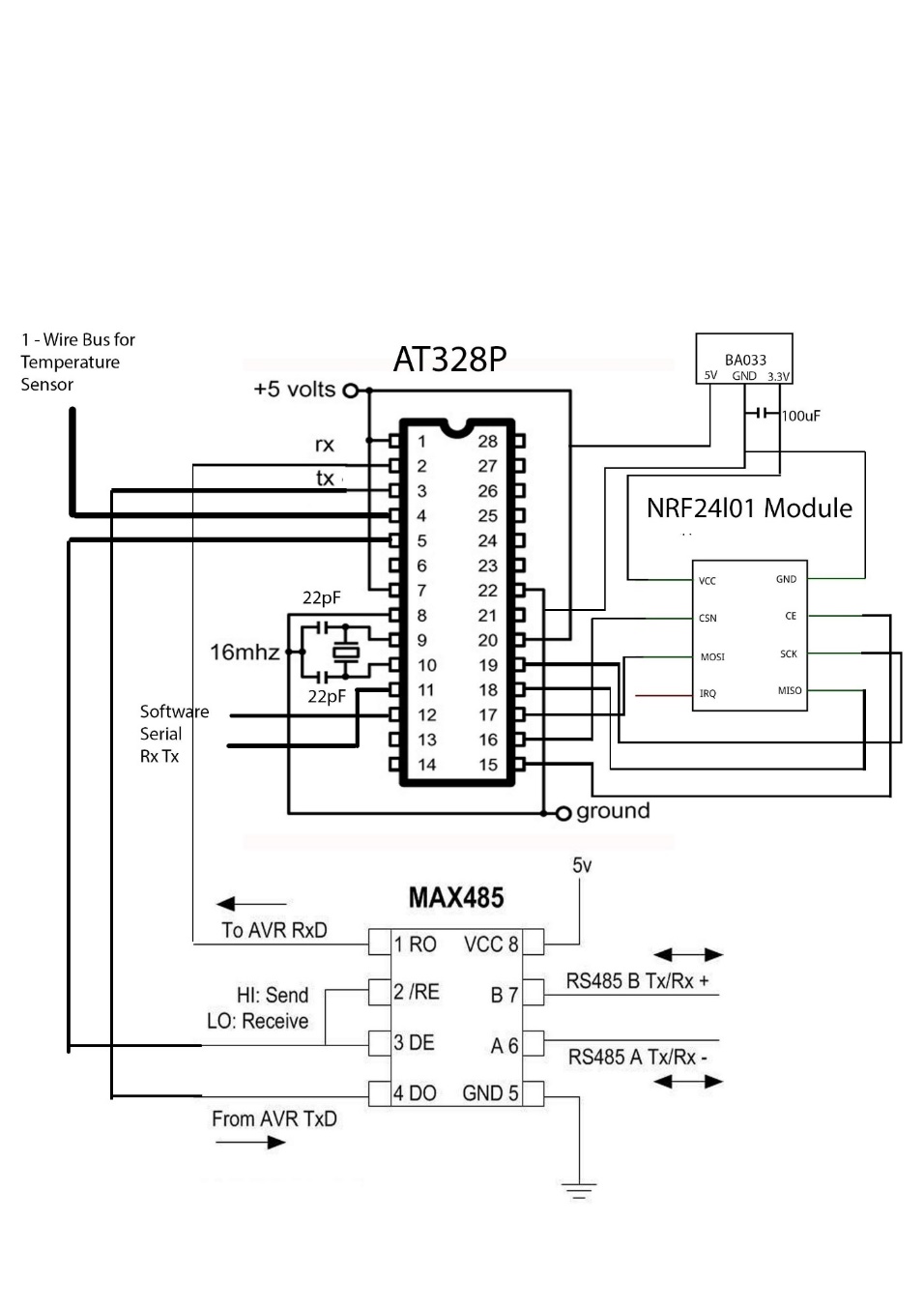
1 Wire Bus

Connection

ATMega328P Microcontroller

TYPE 01 CIRCUIT

1. Collecting data from nearby microcontrollers and send to the monitoring station.



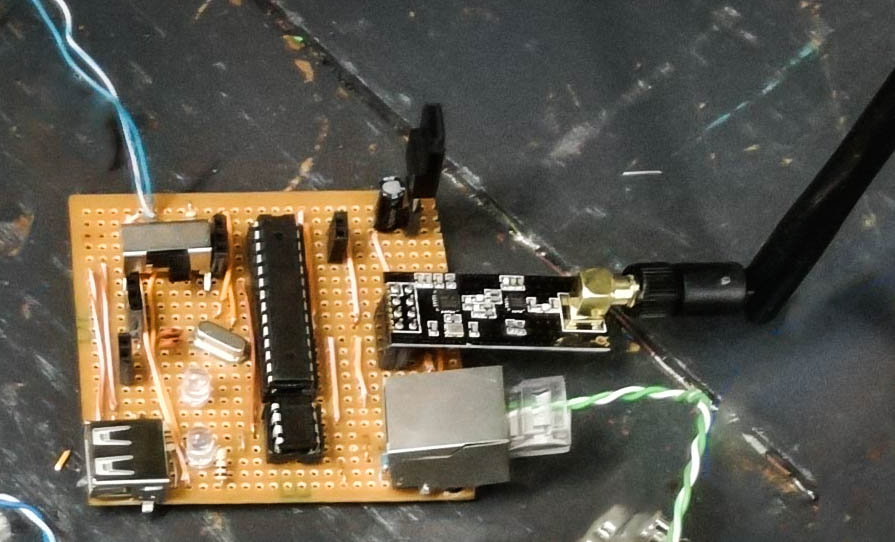
Type 02 Circuit Diagrame

3.3v Regulator for nRF24L01+

Wireless

Module

Hardware Serial connection for Programming Microcontroller

****

Max485 Transceiver IC

Software Serial output

1 Wire Bus

Connector

I2C Connector

nRF24L01+

Wireless

Module

RJ 45 Connector

For RS 485 Bus

USB Interface for power supply and

Purposed serial debug through USB interface

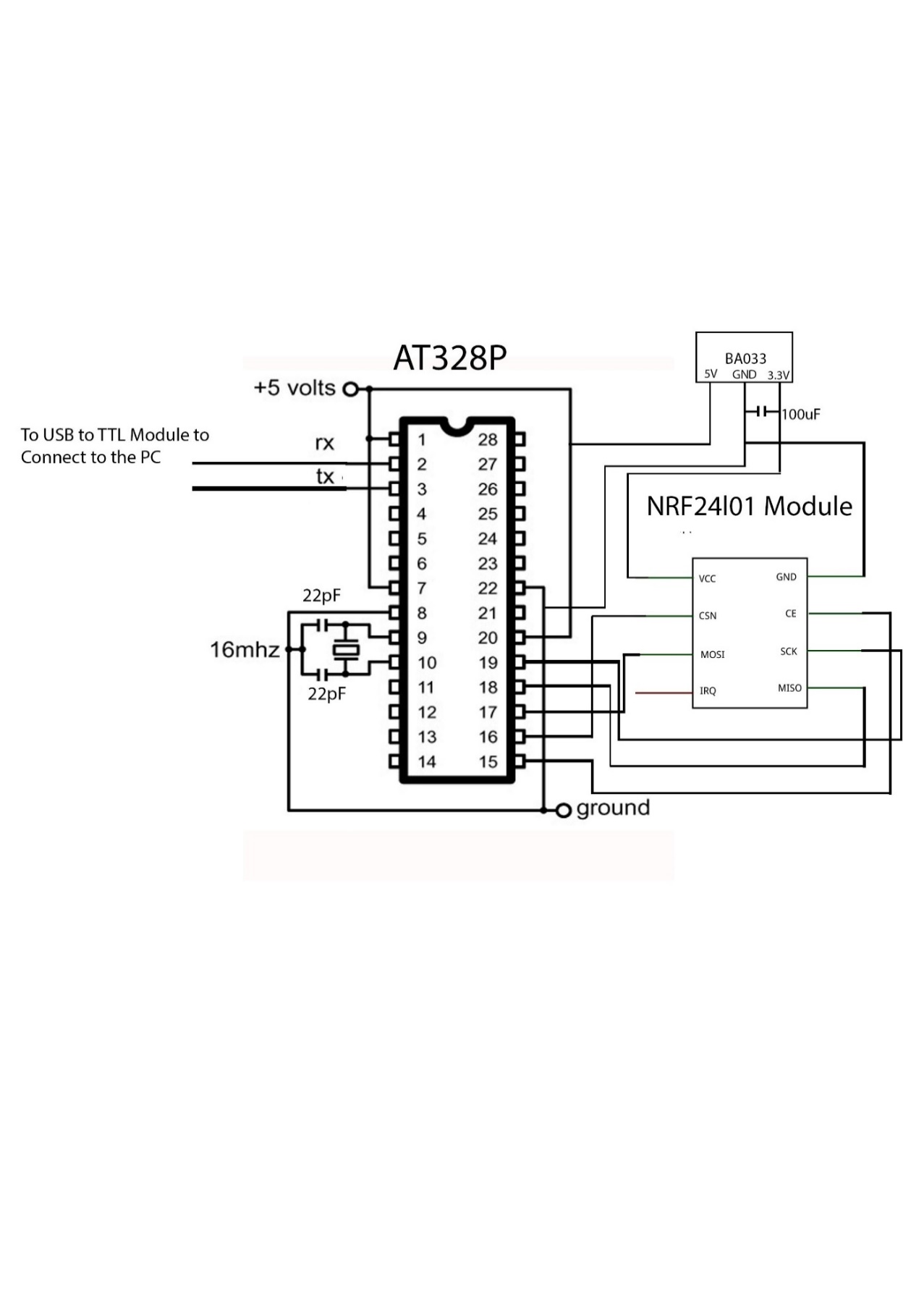
Selection Switch for

Working mode and

Programming

Mode (01)

1. Data Collecting Station.



Type 03 Circuit Diagrame

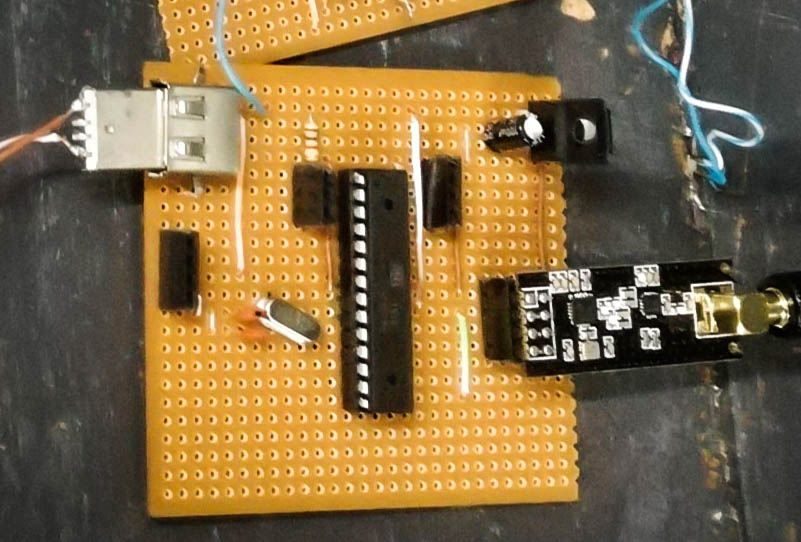
USB Interface for power supply and

Purposed direct PC connection without USB TTL Converter

3.3v Regulator for nRF24L01+

Wireless

Module

****

ATMega328P Microcontroller

nRF24L01+

Wireless

Module

I2C Connector

Hardware Serial connection for Programming Microcontroller and send Data to Computer

1 Wire Bus

Connector

TYPE 03 CIRCUIT

**Programming**

To program ATMega328P Microcontroller I used Arduino Programming Language with Arduino Software (IDE).I have used following Libraries to interface other sensors and modules to Microcontroller. For more information please refer to the links on references section. All These library files are open source softwares and free to use.

1. **Dallas Temperature Control Library**

I used this software library in the project to get temperature readings from DS18B20 thermometer to Microcontroller.

**Credits**

Miles Burton miles@mnetcs.com originally developed this library. Tim Newsome nuisance@casualhacker.net added support for multiple sensors on the same bus. Guil Barros [gfbarros@bappos.com] added getTempByAddress (v3.5) Rob Tillaart [rob.tillaart@gmail.com] added async modus (v3.7.0)

# **OneWire Library**

OneWire library lets access 1-wire devices made by Maxim/Dallas. We have to use this library with Dallas Temperature Control Library.

**Credits**

Jim Studt wrote OneWire in 2007, originally based on code by Derek Yerger.

## ICSC (Inter-Chip Serial Communications) Library

# This software library was used to archive communication between microcontrollers using RS485 bus system.

**Credits**

Majenko (https://github.com/MajenkoLibraries) originally developed this library.

1. **SoftwareSerial Library**

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino using software to replicate the functionality. This software library was used to get additional serial communication pins to get Debugging information.

**Credits**

The version of SoftwareSerial included in 1.0 and later is based on the NewSoftSerial library by Mikal Hart.

## SPI library

## This Software library was used to use Serial Peripheral Interface (SPI) to connect Wireless nRF24L01+ module to microcontroller.

**Credits**

## This software library developed in 2010 by Cristian Maglie (c.maglie@bug.st).

1. **nRF24L01 and RF24 libraries**

# These software libraries were used as driver for nRF24L01 2.4GHz Wireless Transceiver.

**Credits**

These software libraries wew developed in 2007 by Stefan Engelke (mbox@stefanengelke.de).

1. **Adafruit GFX and Adafruit SSD1306 libraries**

These software libraries were used as driver for OLED Display used in testing.

**Credits**

Developed by adafruit.com

**Codes**

1. **For Circuit Type 01**

Following code is written to read the temperature data from the sensor and the send that data through RS485 bus. This program is written to get data from 2 temperature sensors.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <ICSC.h>

#include <OneWire.h>

Define Library files used in code

#include <DallasTemperature.h>

#include <SoftwareSerial.h>

Data wire is plugged into port

4 on the Arduino

#define ONE\_WIRE\_BUS 4

#define TEMPERATURE\_PRECISION 12

Define value for Sensor precision.

OneWire oneWire(ONE\_WIRE\_BUS);

Setup an oneWire instance to communicate

with any OneWire devices.

float temp;

Define variables

char itemp;

int i = 0;

Pass our oneWire reference

to Dallas Temperature.

DallasTemperature sensors(&oneWire);

SoftwareSerial mySerial(10, 11); // RX, TX

Define Software Serial pins

/\* There are the addresses for the connected sensors. If sensor is

\* replaced, the address should change\*/

uint8\_t sen1[8] = {0x28, 0xEE, 0x6B, 0x56, 0x25, 0x16, 0x02, 0xCF};

Define sensor addresses

uint8\_t sen2[8] = {0x28, 0xEE, 0x02, 0x72, 0x25, 0x16, 0x02, 0x53};

void setup()

{

Define pins and baud rate for ICSC

Library

ICSC.begin(2, 115200,2);

mySerial.begin(9600);

Initialize software serial

mySerial.println("Device Started..");

Initialize Temperature sensor

Library

sensors.begin();

mySerial.print("Locating devices...");

Locate number of the sensors on the bus

mySerial.print("Found ");

mySerial.print(sensors.getDeviceCount(), DEC);

mySerial.println(" devices.");

mySerial.print("Parasite power is: ");

Report parasite power requirements

if (sensors.isParasitePowerMode()) Serial.println("ON");

else mySerial.println("OFF");

sensors.setResolution(sen1, TEMPERATURE\_PRECISION);

Set the resolution to 12 bit on Sensors

sensors.setResolution(sen2, TEMPERATURE\_PRECISION);

mySerial.print("Sensor 01 Resolution: ");

Verify the resolution of Sensors

mySerial.print(sensors.getResolution(sen1), DEC);

mySerial.println();

mySerial.print("Sensor 02 Resolution: ");

mySerial.print(sensors.getResolution(sen2), DEC);

mySerial.println();

void loop()

{

Serial.print("Requesting temperatures...");

sensors.requestTemperatures();

Request data from Sensors

Serial.println("DONE");

Sensor\_1\_temp();

Call functions

Sensor\_2\_temp();

delay(500);

500ms Delay

}

void Sensor\_1\_temp()

{

float tempC1 = sensors.getTempC(sen1);

Reading the data from sensor 01

if(tempC1 == -127){

ICSC.send(1, 'Q', 10, "z");

If the value equal to -127

Sends an error massage

mySerial.println("Senror fault");

}

else{

char itemp1 = (char) tempC1;

ICSC.send(1, 'Q', 10, (char \*)&itemp1);

Else send the data to master station on RS485 Bus and print on debug screen.

mySerial.print("Temperature Sent to Serial Bus");

mySerial.println("Temperature of Sensor 01 : ");

mySerial.print(tempC1);

mySerial.println(" C");

}

}

void Sensor\_2\_temp()

{

float tempC2 = sensors.getTempC(sen2);

if(tempC2 == -127){

ICSC.send(1, 'R', 10, "z");

mySerial.println("Senror fault");

Repeating same function on

Sensor 02

}

else{

char itemp2 = (char) tempC2;

ICSC.send(1, 'R', 10, (char \*)&itemp2);

mySerial.print("Temperature Sent to Serial Bus");

mySerial.println("Temperature of sensor 02 : ");

mySerial.print(tempC2);

mySerial.println(" C");

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **For circuit type 02**

With the following code, the data was collected to the microcontroller then sent to monitoring station wirelessly with nRF24L01 wireless modules. During the testing a I2C OLED display was attached for debugging and this code includes that part also. This code can handle data from 3 temperature sensors and number of sensors can be increased.

Here nRF24L01 uses "pipes" that connect from transmitter to receiver. Pipes have an address that needs to set. The Transmitter pipe must have the same address as the Receiver pipe. Later it's possible to use multiple "pipes" at once.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <ICSC.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#include <SoftwareSerial.h>

Define Library files used in code

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

#define OLED\_RESET 4

Define OLED reset pin

Adafruit\_SSD1306 display(OLED\_RESET);

SoftwareSerial mySerial(5, 6);

Define Software serial pins

#### Create a radio object and set the

#### Arduino pins to be used for CE and CS

RF24 radio(9, 10);

const byte rxAddr[6] = "00001";

Define address for pipe to use

int i=0;

int k=0;

int j=0;

int l=0;

int m=0;

int16\_t temp1;

Define variables

int16\_t temp2\_1;

int16\_t temp2\_2;

int16\_t temp2\_3;

int16\_t text[3];

void setup()

{

mySerial.begin(9600);

Initialize software serial

ICSC.begin(1, 115200,2);

Initialize the serial port and

internal structures.

ICSC.registerCommand('P', &p\_received);

ICSC.registerCommand('Q', &q\_received);

Register a new commands

to be acted upon.

ICSC.registerCommand('R', &r\_received);

display.begin(SSD1306\_SWITCHCAPVCC, 0x3C);

display.clearDisplay();

display.setTextSize(1);

Initialize and set parameters

For OLED display

display.setTextColor(WHITE);

display.setCursor(0,0);

display.println("Display initialized..!");

display.display();

delay(500);

display.clearDisplay();

radio.begin();

Start up the actual radio module

with the "begin" method

radio.setRetries(15, 15);

radio.openWritingPipe(rxAddr);

Start pipe with given address

radio.stopListening();

Stop listening

}

void loop()

{

ICSC.process();

## Performing reception

i++;

k++;

l++;

## Incrementing values of

## variables

m++;

display.clearDisplay();

## Clearing display

display.setTextSize(1);

## Set display parameters

display.setTextColor(WHITE);

display.setCursor(0,0);

p\_print();

## Calling functions to print

## Data on display

q\_print();

r\_print();

mySerial.println("Running..");

## Write do serial debug interface

text[0]=temp1;

## Write data into an array

text[1]=temp2\_1;

text[2]=temp2\_2;

## Send data

radio.write(&text, sizeof(text));

delay(200);

## Delay of 200ms

display.setCursor(0,26);

if(j==0){

j=j+1;

}

else if(j==1){

display.println(".");

j=j+1;

}

## Shows a line of incrementing

## Dots. This was added to make

## Sure program is running in

## testing

else if(j==2){

display.println("..");

j=j+1;

}

else if(j==3){

display.println("...");

j=0;

}

else{j=0;}

display.display();

delay(250);

}

void p\_received(unsigned char src, char command, unsigned char len, char \*datap)

{

## This function is called when

## A packet came with the

## Corresponding Command ID

i = 0;

int16\_t \*value = (int16\_t \*)datap;

temp1 = \*value;

mySerial.println("P received data : ");

mySerial.println(temp1);

}

void p\_print(){

if(i>=10){

display.print("Station 1 fail");

## This function gives a error massage

## If no new data comes in 10 loops or

## Value 122 received. (122 received

## When There is a fault with sensor)

}

else if(temp1==122){

display.print("Station 1 sensor Fail ");

}

else{

display.print("Station 1 : ");

display.print(temp1);

mySerial.println("P temp : ");

mySerial.println(temp1);

display.println(" C");

}

}

void q\_print(){

display.setCursor(0,8);

if(k>=10){

display.print("Station 2 fail");

}

else if(temp2\_1 == 122){

display.print("S2 Sen 1 sensor Fail ");

}

else{

display.print("S02 sen 1 : ");

display.print(temp2\_1);

mySerial.println("Q temp : ");

mySerial.println(temp2\_1);

display.println(" C");

}

}

void r\_print(){

display.setCursor(0,16);

if(l>=10){

display.print("Station 2 fail");

}

else if(temp2\_2==122){

display.print("S2 Sen 2 sensor Fail ");

}

else{

display.print("S02 sen 2 : ");

display.print(temp2\_2);

display.println(" C");

}

}

void q\_received(unsigned char src, char command, unsigned char len, char \*dataq)

{

k = 0;

int16\_t \*value = (int16\_t \*)dataq;

temp2\_1 = \*value;

mySerial.println("Q received data : ");

mySerial.println(temp2\_1);

}

void r\_received(unsigned char src, char command, unsigned char len, char \*datar)

{

l = 0;

int16\_t \*value = (int16\_t \*)datar;

temp2\_2 = \*value;

mySerial.println("R received data : ");

mySerial.println(temp2\_2);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **For Circuit Type 03**

In this code the data was received from the nRF24L01 wireless modules and send to serial interface to display in computer. It is also possible to add a display to this unit also. After this it is possible to develop softwares to visualizing, analyzing and to store data for later use.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <SPI.h>

Define Library files used in code

#include <nRF24L01.h>

#include <RF24.h>

#### Create a radio object

RF24 radio(9, 10);

const byte rxAddr[6] = "00001";

Define address for pipe to use

void setup()

{

Start Serial connection

Serial.begin(9600);

Start up the actual radio module

with the "begin" method

radio.begin();

radio.openReadingPipe(0, rxAddr);

radio.startListening();

Start pipe with given address

}

void loop()

{

if (radio.available())

Check for available incoming

data from transmitter

{

int text[3];

radio.read(&text, sizeof(text));

Get the data payload

for (int j=0;j<3;j++){

if(j==0){

Serial.print("Station 1 Temperature : " );

Serial.println(text[j]);

}

else if(j==1){

Serial.print("Station 2 Sensor 1 Temperature : " );

Write data into serial monitor

Serial.println(text[j]);

}

else if(j==2){

Serial.print("Station 2 Sensor 2 Temperature : " );

Serial.println(text[j]);

Serial.println(); }

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Fault Detection**

There is a Serial data output for debugging purposes in every design and 2 types of failure detections. If a sensor stopped working we can get an alarm saying exactly which sensor shows the fault and if a microcontroller fails it will show which device is showing the error.

**Testing and Final Output**

While testing one error occurred due to the low smoothness of the supply voltage. The system was functioning properly but after 4, 5 minutes it stopped working. It was fixed by putting a capacitor across the voltage input.

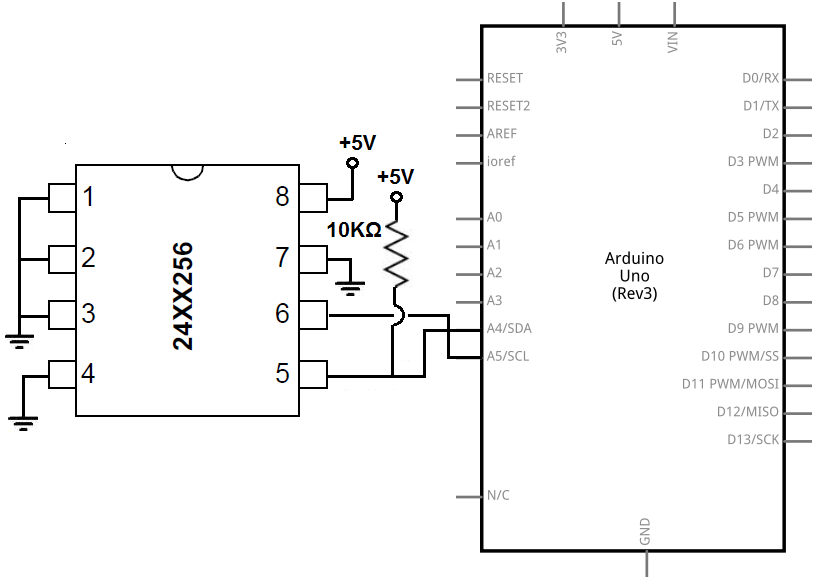
We tested the range of the wireless modules. Outdoor it worked for more than 500m range without any issue. But indoor the range was significantly reduced.

The system was working as I expected. I was able to get data from 3 sensors in 2 stations to monitoring station successfully. One thing I was not able to test as the range of the RS485 bus line.

**Developing Further more**

1. **Data Storing**

We can add a EEPROM like **24LC256 which can connect with I2C to Microcontroller to store temperature data temporary and save to a SD card periodically. The following diagram shows the connections between Arduino and EEPROM and we can use I2C\_EEPROM library by** robtillaart to communicate easily.



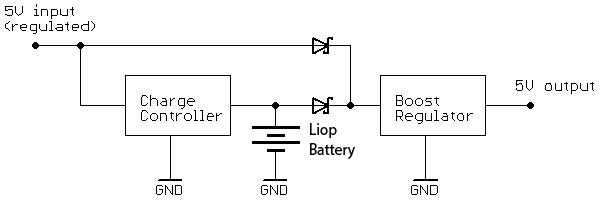
Connection Between Arduino and EEPROM

1. **USB to TTL IC to connect device directly with USB interface for debugging and Programming.**

We can add a USB to TTL converting IC like PL2303, FT232, CP2102 and CH340g to Circuit so it will be easy to do debugging or programming Microcontroller directly with device USB port.

1. **Adding UPS like Power Backup**

I have noticed that in a power failure it takes some time to start the generators and get the power back up. In that kind of case with a help of small power backup, we can run the system without stopping.



Suggested Power Backup

**Different Approach**

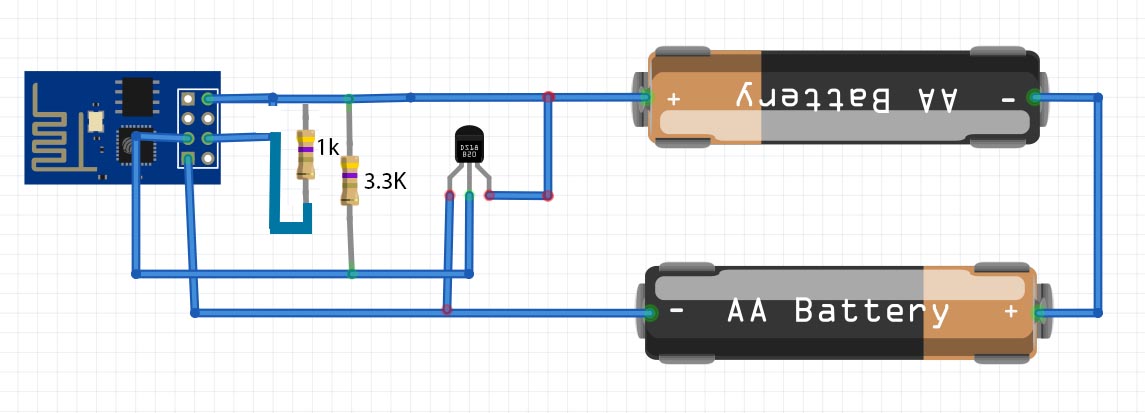
Instead of using RS485 and 2.4GHz Radio links we can use existing wifi network to send data to monitoring station. For this we can use instead of ATMega328P, ESP8266 by Espressif which is a Wifi transceiver with inbuilt microcontroller we can also program with Arduino IDE. There are many variants of ESP8266 but for our project we can use ESP 01. The monitoring station can be either a also a ESP8266 based design or a computer. To communicate with computer here we can use UDP(User Datagram Protocol).

We can use ESP8266 modules here in 2 methods.

1. ESP8266 to ESP8266 Connection through Wi-Fi.
2. Connect Esp8266 stations with another Wifi network and communicate.
3. **ESP8266 to ESP8266 Connection through Wi-Fi.**

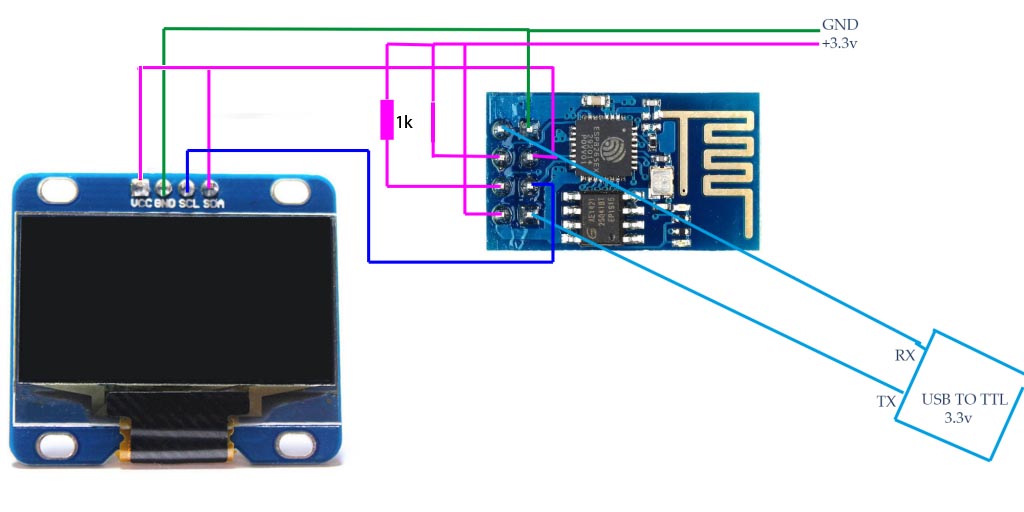
In this method one ESP8266 act as The Server and other one act as the Client. They are communicating through TCP protocol over Wifi.

Client Circuit Diagram as follows.

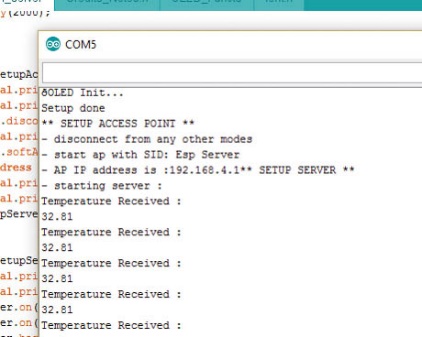
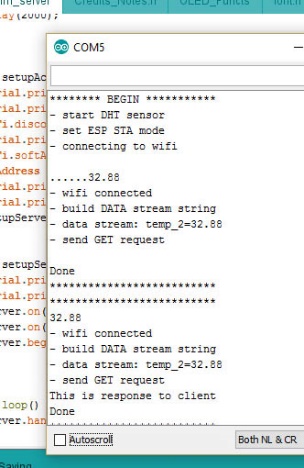


Here the Client get the temperature data from DS18B20 temperature sensor and send it to the Server.

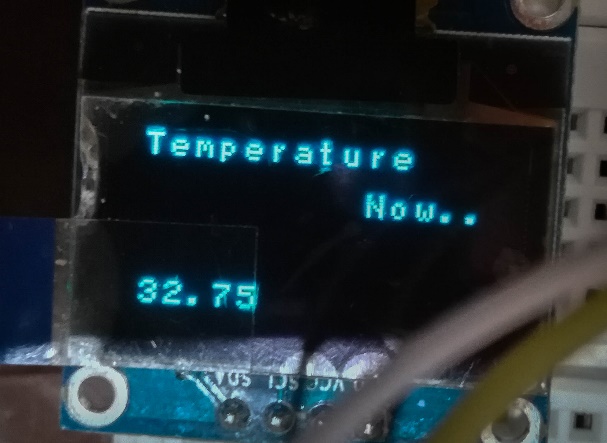
Server Diagram as Follows.



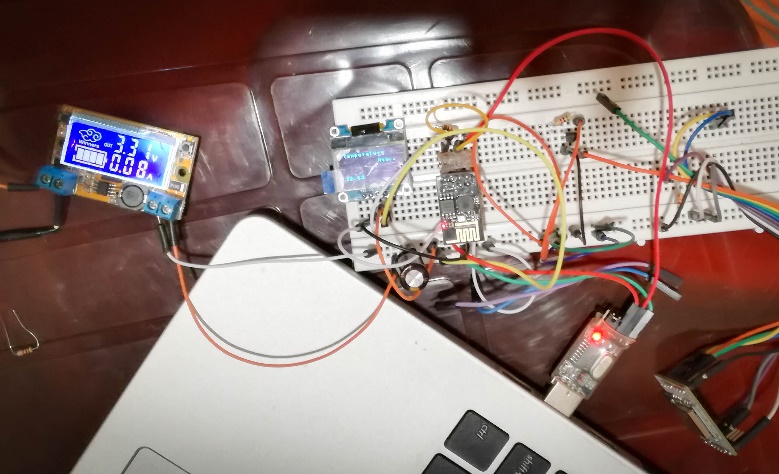
Here is the test setup and it was working as I expected.



Serial output From Server Serial Output from Client



OLED Display output



Test Setup

Code for the Server ESP8266

For this code to work, OLED\_Functs.ino and font.h files from following link should be also in same folder when compliling this code.

https://halckemy.s3.amazonaws.com/uploads/document/file/54091/ESP\_WiFiScan-Acquire\_2.zip

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include <Wire.h>

char buffer[20];

char charbuf[50];

char charbuf2[50];

extern "C" {

#include "user\_interface.h"

}

// Internet router credentials

const char\* ssid = "Esp Server";

const char\* password = "123123123";

ESP8266WebServer server(80);

void setup() {

Serial.begin(74880);

WiFi.mode(WIFI\_AP\_STA);

Wire.pins(0, 2); //on ESP-01.

Wire.begin();

StartUp\_OLED(); // Init Oled and fire up!

Serial.println("OLED Init...");

clear\_display();

sendStrXY(" Temperature ", 0, 1); // 16 Character max per line with font set

sendStrXY(" Monitoring project ", 2, 1);

sendStrXY(" By Imesh Sachinda ", 2, 1);

delay(4000);

Serial.println("Setup done");

setupAccessPoint();

}

// Handling the / root web page from my server

void handle\_index() {

server.send(200, "text/plain", "Get out from my server!");

}

// Handling the /feed page from my server

void handle\_feed() {

String t = server.arg("temp");

String h = server.arg("temp\_2");

server.send(200, "text/plain", "This is response to client");

//setupStMode(t, h);

Serial.println(t);

Serial.println(h);

t.toCharArray(charbuf,50);

h.toCharArray(charbuf2,50);

clear\_display();

sendStrXY(charbuf, 0, 1); // 16 Character max per line with font set

sendStrXY(charbuf2, 1, 1);

delay(2000);

}

void setupAccessPoint(){

Serial.println("\*\* SETUP ACCESS POINT \*\*");

Serial.println("- disconnect from any other modes");

WiFi.disconnect();

Serial.println("- start ap with SID: "+ String(ssid));

WiFi.softAP(ssid, password);

IPAddress myIP = WiFi.softAPIP();

Serial.print("- AP IP address is :");

Serial.print(myIP);

setupServer();

}

void setupServer(){

Serial.println("\*\* SETUP SERVER \*\*");

Serial.println("- starting server :");

server.on("/", handle\_index);

server.on("/feed", handle\_feed);

server.begin();

}void loop() {

server.handleClient();

}

Code for client ESP8266

In the hardware for testing I was using a ESP 12 module and this code is for that model. For any other models, Pin configuration should change as needed.

#include <OneWire.h>

#include <DallasTemperature.h>

#include <ESP8266WiFi.h>

#include <ESP8266HTTPClient.h>

#define ONE\_WIRE\_BUS D1

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature DS18B20(&oneWire);

char temperatureString[6];

float temperature;

float temp;

// AP Wi-Fi credentials

const char\* ssid = "Esp Server";

const char\* password = "123123123";

float h;

float t;

// Static network configuration

IPAddress ip(192, 168, 4, 4);

IPAddress gateway(192, 168, 4, 1);

IPAddress subnet(255, 255, 255, 0);

WiFiClient client;

void setup() {

ESP.eraseConfig();

WiFi.persistent(false);

Serial.begin(74880);

Serial.println();

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Serial.println("\*\*\*\*\*\*\*\* BEGIN \*\*\*\*\*\*\*\*\*\*\*");

Serial.println("- start Dallas Temperature sensor");

delay(500);

Serial.println("- set ESP STA mode");

WiFi.mode(WIFI\_STA);

Serial.println("- connecting to wifi");

WiFi.config(ip, gateway, subnet);

WiFi.begin(ssid, password);

Serial.println("");

while (WiFi.status() != WL\_CONNECTED) {

if(counter > 20){

Serial.println("- can't connect, going to sleep");

}

delay(500);

Serial.print(".");

counter++;

DS18B20.begin();

}

}

void sendHttpRequest() {

HTTPClient http;

http.begin(serverHost);

http.addHeader("Content-Type", "application/x-www-form-urlencoded");

http.POST(data);

http.writeToStream(&Serial);

http.end();

}

void buildDataStream() {

data = "temp\_2=";

data += temp;

Serial.println("- data stream: "+data);

}

float getTemperature() {

do {

DS18B20.requestTemperatures();

temp = DS18B20.getTempCByIndex(0);

delay(100);

} while (temp == 85.0 || temp == (-127.0));

return temp;

}

void loop() {

float temperature = getTemperature();

dtostrf(temp, 2, 2, temperatureString);

// send temperature to the serial console

Serial.println(temperatureString);

while (WiFi.status() != WL\_CONNECTED) {

WiFi.begin(ssid, password);

Serial.println("Trying to connect");

delay(100);

}

Serial.println("- wifi connected");

Serial.println("- build DATA stream string");

buildDataStream();

Serial.println("- send GET request");

sendHttpRequest();

Serial.println();

Serial.println("Done");

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

delay(1000);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Connect ESP8266 stations with another Wifi network and communicate**

In this method hardware connections are same as before and Code only changes. Here instead of one ESP8266 act as wifi Access point and other connects to that, both ESP8266 s connects to another wifi network and communicates through that network. For testing, I used my home wifi router and connected both ESP8266 modules to that. Then also I was able to get temperature data to receiving esp8266 successfully. Test setup was Working as I was expected.

Code for the Server ESP8266

In this code Server ESP8266 tries to request a fix IP. This IP address should put in client Code to send data.

For this code to work, OLED\_Functs.ino and font.h files from following link should be also in same folder when compliling this code.

https://halckemy.s3.amazonaws.com/uploads/document/file/54091/ESP\_WiFiScan-Acquire\_2.zip

#include <ESP8266WiFi.h>

#include <Wire.h>

char ssid[] = "i2s"; // SSID of your home WiFi

char pass[] = "123123123"; // password of your home WiFi

WiFiServer server(80);

IPAddress ip(192, 168, 43, 201); // IP address of the server

IPAddress gateway(192,168,43,1); // gateway of your network

IPAddress subnet(255,255,255,0); // subnet mask of your network

void setup() {

Serial.begin(115200); // only for debug

Wire.pins(0, 2); //on ESP-01.

Wire.begin();

StartUp\_OLED(); // Init Oled and fire up!

Serial.println("OLED Init...");

clear\_display();

sendStrXY(" Temperature ", 0, 1); // 16 Character max per line with font set

sendStrXY(" Monitoring project ", 2, 1);

sendStrXY(" By Imesh Sachinda ", 2, 1);

delay(2000);

WiFi.config(ip, gateway, subnet); // forces to use the fix IP

WiFi.begin(ssid, pass); // connects to the WiFi router

clear\_display();

sendStrXY(" Connecting to wifi..", 2, 1);

while (WiFi.status() != WL\_CONNECTED) {

Serial.print(".");

delay(500);

}

server.begin(); // starts the server

Serial.println("Connected to wifi");

Serial.print("Status: "); Serial.println(WiFi.status()); // some parameters from the network

Serial.print("IP: "); Serial.println(WiFi.localIP());

Serial.print("Subnet: "); Serial.println(WiFi.subnetMask());

Serial.print("Gateway: "); Serial.println(WiFi.gatewayIP());

Serial.print("SSID: "); Serial.println(WiFi.SSID());

Serial.print("Signal: "); Serial.println(WiFi.RSSI());

Serial.print("Networks: "); Serial.println(WiFi.scanNetworks());

clear\_display();

sendStrXY(" Connected..", 0, 1);

delay(2000);

//pinMode(ledPin, OUTPUT);

}

void loop () {

WiFiClient client = server.available();

if (client) {

if (client.connected()) {

//digitalWrite(ledPin, LOW); // to show the communication only (inverted logic)

Serial.println(".");

String request = client.readStringUntil('\r'); // receives the message from the client

Serial.print("From client.Temperatue : "); Serial.println(request);

clear\_display();

sendStrXY(" Temperatue ", 0, 1);

char charbuf[50];

request.toCharArray(charbuf,50);

sendStrXY(charbuf, 2, 1);

client.flush();

client.println("Hi client! No, I am listening.\r"); // sends the answer to the client

//digitalWrite(ledPin, HIGH);

}

client.stop(); // tarminates the connection with the client

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code for client ESP8266**

In this code the client connects to the temperature sensor and send the temperature data to server.

#include <OneWire.h>

#include <ESP8266WiFi.h>

#include <DallasTemperature.h>

#define ONE\_WIRE\_BUS D1

float temp;

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature DS18B20(&oneWire);

char temperatureString[6];

//byte ledPin = 2;

char ssid[] = "i2s"; // SSID of your home WiFi

char pass[] = "123123123"; // password of your home WiFi

unsigned long askTimer = 0;

IPAddress server(192,168,43,201); // the fix IP address of the server

WiFiClient client;

void setup() {

Serial.begin(115200); // only for debug

WiFi.begin(ssid, pass); // connects to the WiFi router

while (WiFi.status() != WL\_CONNECTED) {

Serial.print(".");

delay(500);

}

Serial.println("Connected to wifi");

Serial.print("Status: "); Serial.println(WiFi.status()); // Network parameters

Serial.print("IP: "); Serial.println(WiFi.localIP());

Serial.print("Subnet: "); Serial.println(WiFi.subnetMask());

Serial.print("Gateway: "); Serial.println(WiFi.gatewayIP());

Serial.print("SSID: "); Serial.println(WiFi.SSID());

Serial.print("Signal: "); Serial.println(WiFi.RSSI());

//pinMode(ledPin, OUTPUT);

DS18B20.begin();

}

float getTemperature() {

do {

DS18B20.requestTemperatures();

temp = DS18B20.getTempCByIndex(0);

delay(100);

} while (temp == 85.0 || temp == (-127.0));

return temp;

}

void loop () {

float temperature = getTemperature();

dtostrf(temperature, 2, 2, temperatureString);

// send temperature to the serial console

Serial.println(temperatureString);

client.connect(server, 80); // Connection to the server

//digitalWrite(ledPin, LOW); // to show the communication only (inverted logic)

Serial.println(".");

client.println(temperatureString); // sends the message to the server

String answer = client.readStringUntil('\r'); // receives the answer from the sever

Serial.println("from server: " + answer);

client.flush();

//digitalWrite(ledPin, HIGH);

delay(2000); // client will trigger the communication after two seconds

}

**References**

For Library Files

* https://www.milesburton.com/Dallas\_Temperature\_Control\_Library
* https://github.com/milesburton/Arduino-Temperature-Control-Library
* https://www.pjrc.com/teensy/td\_libs\_OneWire.html
* https://github.com/MajenkoLibraries/ICSC
* https://github.com/arduino/Arduino/blob/master/hardware/arduino/avr/libraries/SoftwareSerial/src/SoftwareSerial.h
* https://www.arduino.cc/en/Reference/SPI
* https://github.com/maniacbug/RF24/blob/master/nRF24L01.h
* http://maniacbug.github.io/RF24/
* https://learn.adafruit.com/monochrome-oled-breakouts/overview

More Information about Libraries used

* https://arduino-info.wikispaces.com/Nrf24L01-2.4GHz-HowTo
* https://sourceforge.net/p/arduino-icsc/wiki/Home/
* http://www.elecfreaks.com/wiki/index.php?title=2.4G\_Wireless\_nRF24L01p\_with\_PA\_and\_LNA#Specification

Sources for Specifications

* https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
* https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf
* http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P
* http://www.hobbytronics.co.uk/arduino-external-eeprom
* https://thearduinoandme.wordpress.com/tutorials/esp8266-send-receive-binary-data/
* http://playground.arduino.cc/Main/LibraryForI2CEEPROM
* https://www.hackster.io/rayburne/esp8266-01-using-arduino-ide-67a124
* https://vaasa.hacklab.fi/2016/02/06/esp8266-on-nodemcu-board-ds18b20-arduinoide-thingspeak/
* http://www.geekstips.com/two-esp8266-communication-talk-each-other/