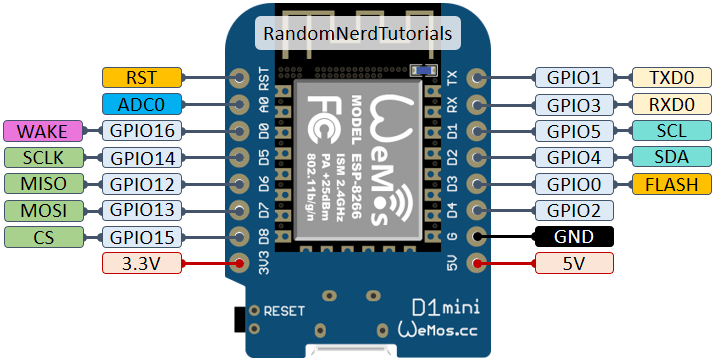
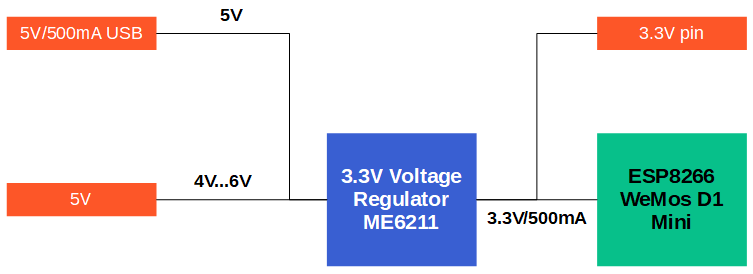
EnviroSense PICO

# # ESP8266 D1 mini



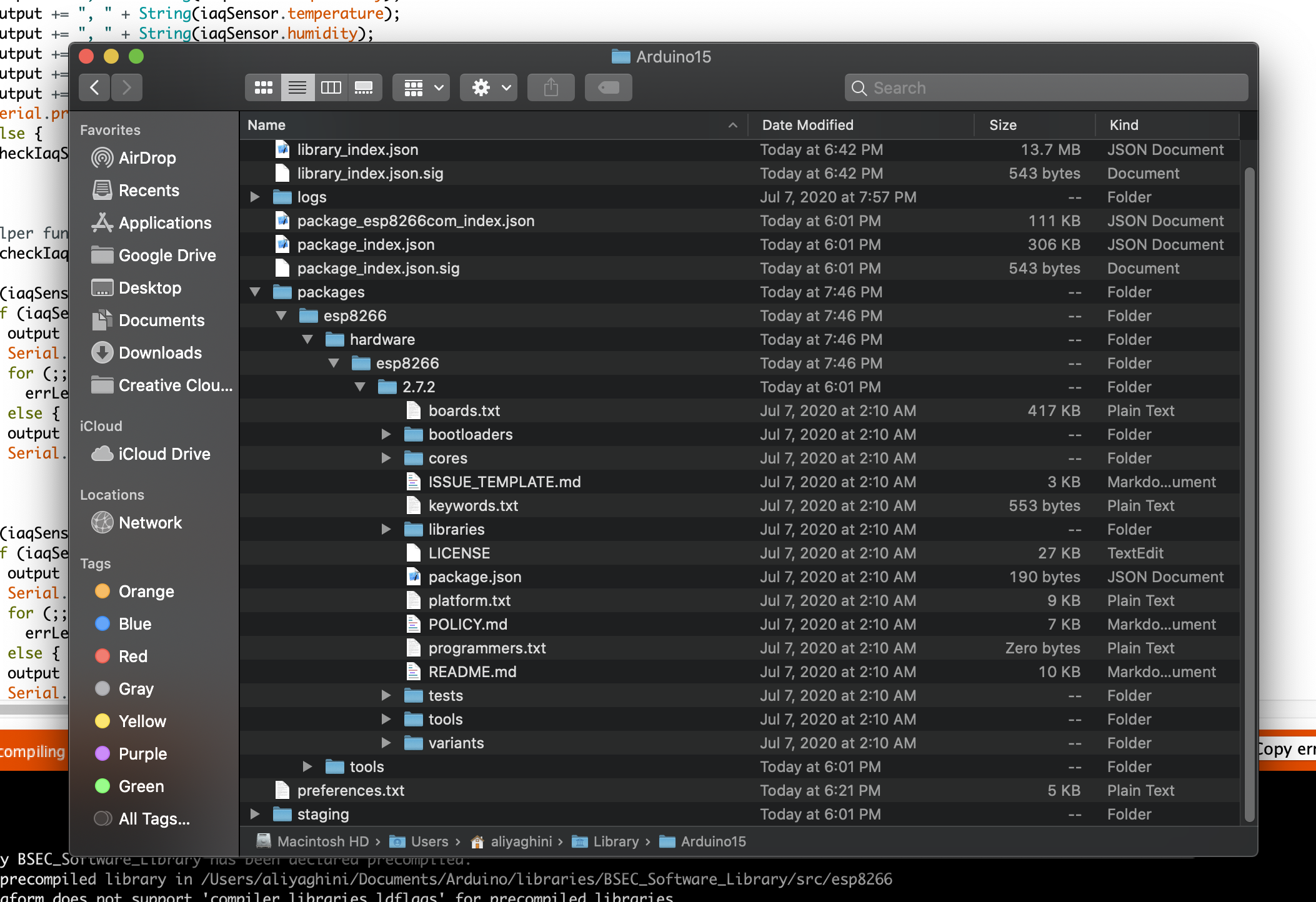


From: https://diyi0t.com/what-is-the-esp8266-pinout-for-different-boards/

# # BME680

Connected through I2C, SDA, SCL, 5v, G.

To edit platform.txt:



I faced many initial problems, so I switched to just an esp32 board for my test, and finally managed to get it working. In my case I had to search for the I2C address of the device and put it in this line:

iaqSensor.begin(0x77, Wire);

# # Display

I had a very hard time getting this display to work. There are couple of problems with it. To start with the PINOUT naming is wrong. This is how it should be:

LCD -- correct -where I am connecting to D1 mini

Scl -> (CLK) -> D5

Sda -> MOSI -> D7

Cs -> D3

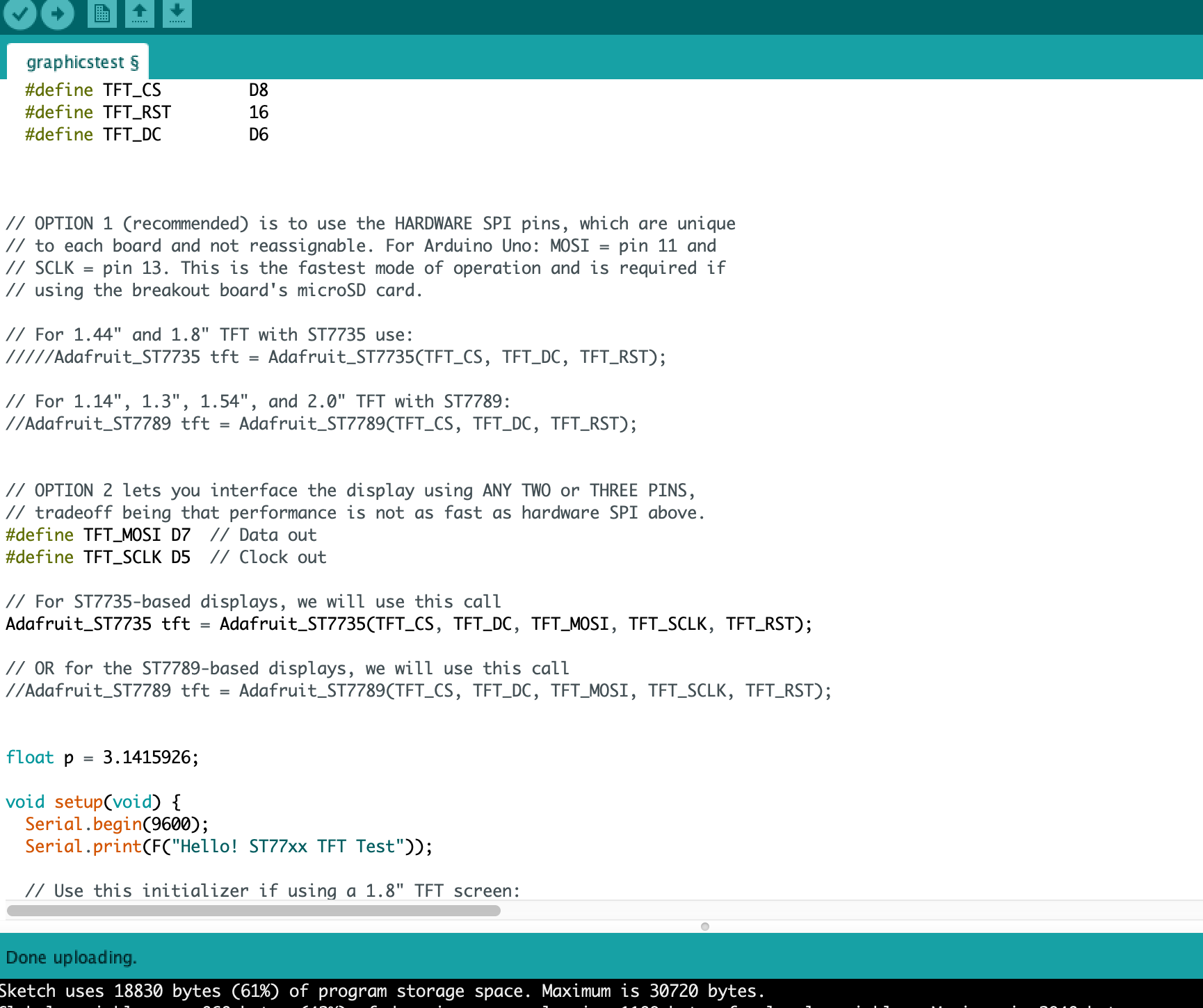
dc -> D4

res -> RX

The reset pin HAS TO BE CONNECTED!

I am using Adafruit library:

This is working so far from Adafruit library:



Code:

//#include <UTFT.h>

#include <ESP8266WiFi.h>

#include <ESP8266mDNS.h>

#include <WiFiUdp.h>

#include <ArduinoOTA.h>

#include <NTPClient.h>

#include <Credentials.h>

#include <Adafruit\_GFX.h> // Core graphics library

#include <Adafruit\_ST7735.h> // Hardware-specific library for ST7735

#include <SPI.h>

#include "bsec.h"

#include "PubSubClient.h" // Connect and publish to the MQTT broker

#ifndef STASSID

#define STASSID "your-ssid"

#define STAPSK "your-password"

#endif

const char\* ssid = mySSID;

const char\* password = myPASSWORD;

// BME680 Helper functions declarations

void checkIaqSensorStatus(void);

void errLeds(void);

// Create an object of the class Bsec

Bsec iaqSensor;

String BMEoutput;

String accuracyState;

String insideTemp;

String insideH;

String aqiText;

String saqi;

String eCO2;

int aqIndex;

unsigned long previousAirMillis = 10;

const unsigned long airInterval = 1 \* 60 \* 1000;

int i = 0;

//int j = 0;

//TFT LCD

#define TFT\_CS D3

#define TFT\_RST RX

#define TFT\_DC D4

#define TFT\_MOSI D7 // Data out

#define TFT\_SCLK D5 // Clock out

Adafruit\_ST7735 tft = Adafruit\_ST7735(TFT\_CS, TFT\_DC, TFT\_MOSI, TFT\_SCLK, TFT\_RST);

// Define NTP Client to get time

const long utcOffsetInSeconds = -25200;

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);

WiFiClient client;

String timeToDisplay = "waiting to receive the time from server";

// MQTT

const char\* mqtt\_server = "192.168.1.92"; // IP of the MQTT broker

const char\* humidity\_topic = "home/EnviroSensePico/humidity";

const char\* temperature\_topic = "home/EnviroSensePico/temperature";

const char\* aqi\_topic = "home/EnviroSensePico/aqi";

const char\* aqiS\_topic = "home/EnviroSensePico/staticaqi";

const char\* eCO2\_topic = "home/EnviroSensePico/equvalentCO2";

const char\* accuracy\_topic = "home/EnviroSensePico/accuracy";

//const char\* mqtt\_username = "cdavid"; // MQTT username

//const char\* mqtt\_password = "cdavid"; // MQTT password

const char\* clientID = "client\_EnviroSensePico"; // MQTT client ID

PubSubClient mqttClient(mqtt\_server, 1883, client);

//Time interval Settings

long interval = 30000; // interval at which to send (milliseconds)

long previousMillis = interval; // will store last time data was sent

long dispInterval = 7000; // interval at which to change display (milliseconds)

long previousDispMillis = interval; // will store last time data was sent

int state = 0;

int displayState = 0;

void setup() {

tft.initR(INITR\_MINI160x80); // Init ST7735S mini display

TextWriter("Enviro Sense Pico v1", ST77XX\_GREEN, 3);

delay(1000);

TextWriter(" ALI YAGHINI", ST77XX\_BLUE, 3);

delay(1000);

TextWriter(" Wi-Fi ...", ST77XX\_WHITE, 2);

delay(1000);

WiFi.begin(ssid, password);

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

delay ( 500 );

//Serial.print ( "." );

}

TextWriter(" Updating Time", ST77XX\_WHITE, 2);

delay(1000);

timeClient.begin();

//BME680

TextWriter(" BME-680 Setup..", ST77XX\_WHITE, 2);

delay(1000);

Wire.begin();

iaqSensor.begin(0x77, Wire);

checkIaqSensorStatus();

bsec\_virtual\_sensor\_t sensorList[10] = {

BSEC\_OUTPUT\_RAW\_TEMPERATURE,

BSEC\_OUTPUT\_RAW\_PRESSURE,

BSEC\_OUTPUT\_RAW\_HUMIDITY,

BSEC\_OUTPUT\_RAW\_GAS,

BSEC\_OUTPUT\_IAQ,

BSEC\_OUTPUT\_STATIC\_IAQ,

BSEC\_OUTPUT\_CO2\_EQUIVALENT,

BSEC\_OUTPUT\_BREATH\_VOC\_EQUIVALENT,

BSEC\_OUTPUT\_SENSOR\_HEAT\_COMPENSATED\_TEMPERATURE,

BSEC\_OUTPUT\_SENSOR\_HEAT\_COMPENSATED\_HUMIDITY,

};

iaqSensor.updateSubscription(sensorList, 10, BSEC\_SAMPLE\_RATE\_LP);

checkIaqSensorStatus();

ArduinoOTA.begin();

}

void loop() {

ArduinoOTA.handle();

StateMachine();

}

void StateMachine() {

unsigned long currentMillis = millis();

switch (state) {

case 0: {

if (currentMillis - previousMillis > interval) {

previousMillis = currentMillis;

UpdateTime();

}

state++;

break;

}

case 1: {

if (iaqSensor.run()) {

updateBME();

}

else {

checkIaqSensorStatus();

}

state++;

break;

}

case 2: {

if (currentMillis - previousDispMillis > dispInterval) {

previousDispMillis = currentMillis;

DisplayManager();

}

state = 0;

break;

}

}

}

void UpdateTime() {

timeToDisplay = GetTime();

//TextWriter(timeToDisplay.c\_str(), ST77XX\_WHITE, 2);

//UpdateDisplayText();

}

String GetTime() {

timeClient.update();

// int hour = timeClient.getHours();

// int minute = timeClient.getMinutes();

// String sep = ":";

// String currentTime = hour + sep + minute;

String currentTime = timeClient.getFormattedTime();

currentTime.remove(5, 3);

return currentTime;

}

void updateBME() {

i++;

UpdateBMEText();

if (i >= 500) {

//DisplayManager();

SendMQTT();

i = 0;

// j = 0;

}

}

void UpdateBMEText() {

aqIndex = iaqSensor.iaq;

aqiText = String(aqIndex);

accuracyState = String(iaqSensor.iaqAccuracy);

insideTemp = String(iaqSensor.temperature);

insideH = String(iaqSensor.humidity);

saqi = String(iaqSensor.staticIaq);

eCO2 = String(iaqSensor.co2Equivalent);

}

void DisplayManager() {

AlertManager();

switch (displayState) {

case 0: {

DisplayTime();

displayState++;

break;

}

case 1: {

DisplayAqi();

displayState ++;

break;

}

case 2: {

DisplayTemp();

displayState = 0;

break;

}

}

}

void DisplayAqi() {

//tft.setCursor(0, 30);

tft.setCursor(1, 1);

tft.setTextColor(ST77XX\_BLUE);

tft.setTextSize(2);

tft.println(eCO2 + "ppm");

tft.setTextColor(ST77XX\_GREEN);

tft.setTextSize(4);

tft.println(aqiText);

tft.setTextColor(ST77XX\_YELLOW);

tft.setTextSize(2);

tft.println(insideH + "%");

}

void DisplayTime() {

tft.setCursor(0, 30);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(4);

tft.println(" " + timeToDisplay);

tft.setTextColor(ST77XX\_MAGENTA);

tft.setTextSize(2);

tft.println(insideTemp + "c");

}

void DisplayTemp() {

tft.setCursor(0, 30);

tft.setTextColor(ST77XX\_MAGENTA);

tft.setTextSize(4);

tft.println(insideTemp + "c");

}

void AlertManager() {

if (aqIndex <= 200) {

tft.fillScreen(ST77XX\_BLACK);

} else {

tft.fillScreen(ST77XX\_RED);

}

}

void SendMQTT() {

// Connect to MQTT Broker

// client.connect returns a boolean value to let us know if the connection was successful.

// If the connection is failing, make sure you are using the correct MQTT Username and Password (Setup Earlier in the Instructable)

if (mqttClient.connect(clientID)) {

SendMQTTMessages(humidity\_topic, String(insideH));

SendMQTTMessages(aqi\_topic, String(aqIndex));

SendMQTTMessages(accuracy\_topic, String(accuracyState));

SendMQTTMessages(aqiS\_topic, String(saqi));

//SendMQTTMessages(aqiS\_topic, saqi));

SendMQTTMessages(eCO2\_topic, String(eCO2));

SendMQTTMessages(temperature\_topic, String(insideTemp));

mqttClient.disconnect(); // disconnect from the MQTT broker

//errLeds();

}

else {

//Serial.println("Connection to MQTT Broker failed...");

}

}

void SendMQTTMessages(const char\* topic, String parameter) {

if (!mqttClient.publish(topic, parameter.c\_str())) {

// Again, client.publish will return a boolean value depending on whether it succeeded or not.

// If the message failed to send, we will try again, as the connection may have broken.

//Serial.println("Temperature failed to send. Reconnecting to MQTT Broker and trying again");

mqttClient.connect(clientID);

delay(10); // This delay ensures that client.publish doesn't clash with the client.connect call

mqttClient.publish(topic, parameter.c\_str());

}

}

void TextWriter(String text, uint16\_t color, int size) {

tft.fillScreen(ST77XX\_BLACK);

tft.invertDisplay(true);

tft.setRotation(1);

tft.setCursor(0, 0);

tft.setTextSize(size);

tft.setTextColor(color);

tft.setTextWrap(true);

tft.print(text);

}

void rotateText() {

for (uint8\_t i = 0; i < 4; i++) {

tft.fillScreen(ST77XX\_BLACK);

Serial.println(tft.getRotation(), DEC);

tft.setCursor(0, 30);

tft.setTextColor(ST77XX\_RED);

tft.setTextSize(1);

tft.println("Hello World!");

tft.setTextColor(ST77XX\_YELLOW);

tft.setTextSize(2);

tft.println("Hello World!");

tft.setTextColor(ST77XX\_GREEN);

tft.setTextSize(3);

tft.println("Hello World!");

tft.setTextColor(ST77XX\_BLUE);

tft.setTextSize(4);

tft.print(1234.567);

delay(1000);

tft.setRotation(tft.getRotation() + 1);

}

}

// BME680 Helper function definitions

void checkIaqSensorStatus(void)

{

if (iaqSensor.status != BSEC\_OK) {

if (iaqSensor.status < BSEC\_OK) {

//output = "BSEC error code : " + String(iaqSensor.status);

//Serial.println(output);

for (;;)

errLeds(); /\* Halt in case of failure \*/

} else {

//output = "BSEC warning code : " + String(iaqSensor.status);

//Serial.println(output);

}

}

if (iaqSensor.bme680Status != BME680\_OK) {

if (iaqSensor.bme680Status < BME680\_OK) {

//output = "BME680 error code : " + String(iaqSensor.bme680Status);

//Serial.println(output);

for (;;)

errLeds(); /\* Halt in case of failure \*/

} else {

//output = "BME680 warning code : " + String(iaqSensor.bme680Status);

//Serial.println(output);

}

}

}

//BME680 Reated

void errLeds(void)

{

pinMode(LED\_BUILTIN, OUTPUT);

digitalWrite(LED\_BUILTIN, HIGH);

delay(100);

digitalWrite(LED\_BUILTIN, LOW);

delay(100);

}