SRH University Heidelberg

Weather Station using Arduino Nano 33 BLE Sense and ESP32

Professional Technical Report

Author:	PANKAN, Anjali
Matriculation Number:	11016821
Email Address:	11016821@stud.hochschule-heidelberg.de
Author:	KANNANCHERY, Vishnu
Matriculation Number:	11017523
Email Address:	11017523@stud.hochschule-heidelberg.de
Supervisor:	Prof. Dr. Bernard Alfred

Contents

Chapter 1

Abstract

In this project, an implementation of weather forecast system using Arduino Nano 33 BLE sence and ESP32 module is discussed. The system consist of a remote station, which measures atmospheric conditions such as light, temperature, humidity, pressure, rain, wind speed and direction. The collected real time data is then transmitted to a base station which forecasts the weather. The remote base station is realized using Arduino Nano 33 BLE sence and the base station is implemented by using ESP32 module. In order to communicate between remote station and the base station, BLE is used. The remote station acts only as a sensing device which uses its inbuilt sensors. The base station runs the decision program to decide on the weather condition. Experiments shows that the proposed weather station forecasts the weather with adequate reliability.

Chapter 2

Introduction

Climate is an important part of human life. It is essential for the growth and wellbeing of every living thing in the world. Changes in atmospheric conditions due to emergence of industries and vehicles has huge impact on everyone's life [1]. It is to be noted that, knowing everyday weather condition is essential for improving the living conditions of every individual in the globe. It is also helpful for informing the people about emergency conditions of strong wind, heat wave or other crucial situations.

There are different methods of weather forecasting including the use of satellite [2] [3]. The advantage of using such system is the global coverage as it can cover over large areas. A drawback of such system is that it only gives value for present system and may not be precise depending on sensing area. By utilizing a local weather reporting station developed from miciconrollers can reduce the error in forecasting the weather. By combing the satellite and local weather station approaches it can give more information about the actual weather conditions.

There are many approaches to implementing a local weather station, such as [4] and [5]. A weather station is essentially a module which combines different sensors to collect data related to environment and atmospheric conditions. By using these sensors atmospheric conditions such as temperature, humidity, light, pressure, rain, wind etc. values can be measured directly. Once these values are available it can calculate the current weather condition by using some weather forecasting logic. The system can also send these values to a powerful base station to perform more complex calculations to extract important information. By storing the measured values in a long time storage, the weather station can forecast the future weather conditions.

2.0.1 Motivation

The motive of this project is to design and develop a real-time weather station using Arduino Nano 33 BLE Sense and ESP32. The system should contain two parts, a remote station which measures atmospheric conditions such as light, temperature, humidity, pressure, rain and wind, and a base station which performs the weather forecasting logic. The two stations should communicate with each other using a wireless technology such as Bluetooth. Finally, the effectiveness of the weather station needs to be tested with respect to real-time weather conditions.

Chapter 3

Hardware

3.1 Components

3.1.1 Arduino Nano 33 BLE Sense

The remote station of this weather station project is implemented by using a Arduino Nano 33 BLE Sense board. The main purpose of a remote station is to sense environmental conditions, collect them and send them to a base station.

Arduino Nano 33 BLE Sense is an evolution of the traditional Arduino Nano [6]. Comparing with Arduino Uno/Nano, Arduino Nano 33 BLE Sense consists of lot more powerful processor. The processor nRF52840 from Nordic Semiconductors is a 32-bit ARM® Cortex™-M4 CPU. The clock speed of the processor is 64 MHz. The Arduino Uno/Nano on the other hand, consists of processor ATmega328 with only clock speed of 16 MHz. This board has large program memory of 1MB, which is 32 times bigger than Arduino Uno. Similarly, the SRAM size is 256KB which is 128 times bigger than the previous board. Having these features it can be used for running large programs. Another important feature of Arduino Nano 33 BLE Sense is the Bluetooth® module, which can be paired via ultra low power consumption modes and NFC. The pin digram of Arduino Nano 33 BLE Sense is shown in Figure 3.1.

Arduino Nano 33 BLE Sense can be programmed by using Arduino IDE and ARM Mbed OS [7]. The main characteristic that makes this board unique from other Anrduino boards is the presence of different inbuilt sensors. It consists of sensors for sensing color, brightness, temperature, humidity, proximity, gesture control, motion, vibration, orientation and pressure. Below is a list of Arduino Nano 33 BLE Sense inbuilt sensors:

- HTS221 - Capacitive digital sensor : It is an ultra-compact sensor for measuring relative humidity as well as temperature.
- LPS22HB - MEMS nano pressure sensor : It is an ultra-compact sensor for measuring absolute pressure. It is piezoresistive and acts as a digital output barometer .
- MP34DT05-A - MEMS audio sensor : It is an ultra-compact, digital MEMS, omnidirectional, low-power microphone.

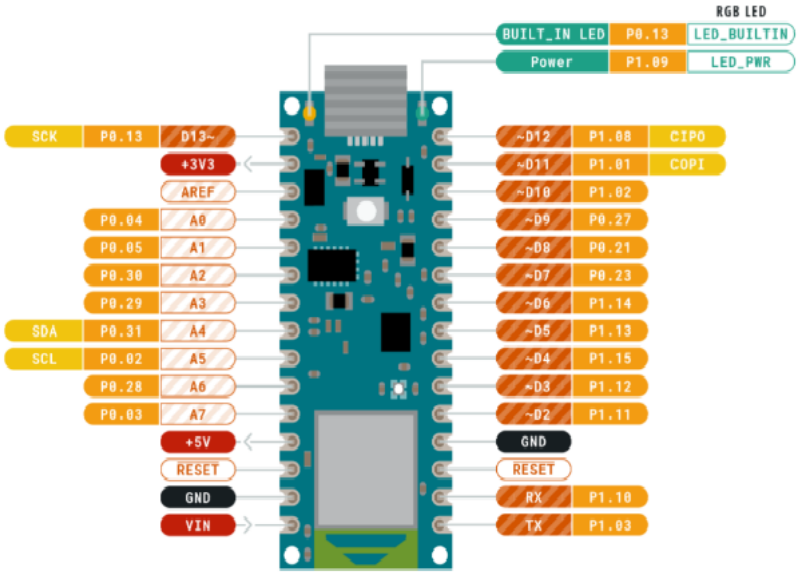

Figure 3.1: Pin Diagram of Arduino Nano 33 BLE Sense

MICROCONTROLLER	nRF52840
OPERATING VOLTAGE	3.3V
CLOCK SPEED	64MHz
CPU FLASH MEMORY	1MB (nRF52840)
SRAM	256KB (nRF52840)

Figure 3.2: Arduino Nano 33 BLE Sense Specification

- APDS-9960 - RGB, Digital Proximity, Ambient Light and Gesture Sensor : It is an ultra-compact single 8-pin sensor for measuring RGB, Digital Proximity, Ambient Light and Gesture.
- LSM9DS1 - iNEMO inertial module : It is a system-in-package consisting of 3D gyroscope, 3D accelerometer, 3D magnetometer.

Advantageof using Arduino Nano 33 BLE Sense:

Because of very large program memory and SRAM capacity compared to other Arduino boards, Arduino Nano 33 BLE Sense is quite suitable for running large programs. The presence of Bluetooth module makes it more suitable for applications which require wireless communication. More essentially, the inbuilt sensors makes this board a must to have component as it does not require extra sensors and so less costly. It is quite easy to use the inbuilt sensors of the board for environment sensing related applications than to use extra sensors. As a result, it is well suitable for implementing the remote station.

3.1.2 ESP32

The base station of this weather station project is realized by using ESP32 module. Base station is responsible for receiving sensor data from the remote station and process them to output weather condition.

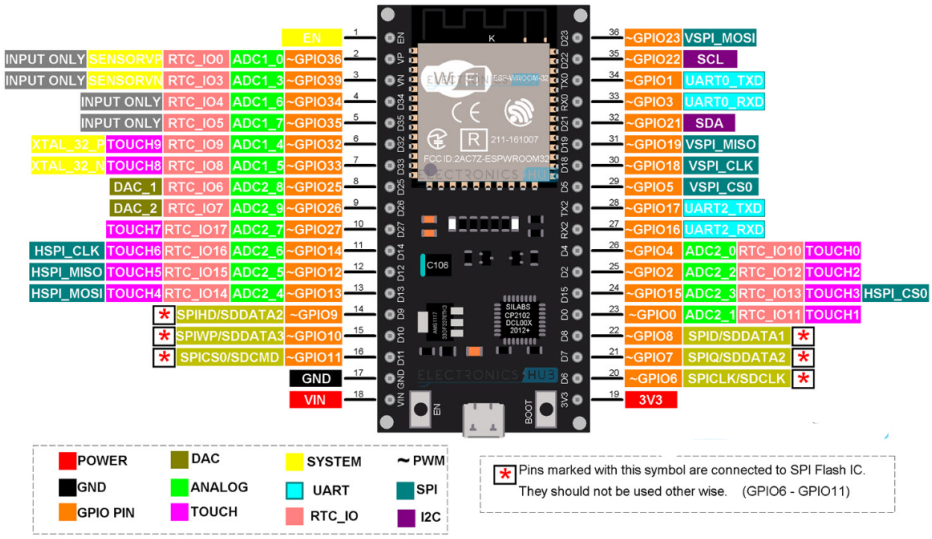

Figure 3.3: Pin Diagram of ESP32

MICROPROCESSOR	Tenscillica Xtensa LX6
OPERATING VOLTAGE	3.3V
CLOCK RATE	240MHz
CPU FLASH MEMORY	4 MB (nRF52840)
SRAM	512KB

Figure 3.4: ESP32 Specification

ESP32 is a powerful, generic, low-cost module that targets a wide range of applications. The module incorporates a Tensilica Xtensa LX6 [8] dual-core microprocessor with a maximum clock frequency of 240 MHz and SRAM of size 512KB. Compared to Arduino boards ESP32 has higher processing power and memory which makes it suitable for implementing base station. In this project the version of ESP32 used is ESP32-WROOM-32. The pin diagram of ESP32 is shown in Figure 3.3 [9].

An important characteristic of ESP32 is the presence of integrated Wi-Fi along with dual mode Bluetooth. The Wi-Fi aims at applications that require large physical range or internet connectivity. The Bluetooth module helps for connect to other peripherals and receive or broadcast data. Like Arduino boards, ESP32 can be programmed with Arduino IDE by installing ESP32-Arduino core. The specifications of ESP32 series microcontroller that is used in this project is shown in Figure 3.3.

Advantage of using ESP32:

Since ESP32 has large processing power and memory it is well suited for implementing base station. By using the Bluetooth module, it can connect to remote station to receive sensor data. The inbuilt Wi-Fi can be used to connect to *NTP* server, over internet, to provide current time for deciding weather condition. Also, an OLED display can be easily attached to ESP32 to show output data.

3.1.3 OLED Display

In this weather station project, OLED is used to display the sensor data from the remote station a well as the foretasted weather. An example OLED display is shown in Figure 3.5. The name OLED stands for Organic Light Emitting Diode. OLED uses latest technology which consists of thin films of light emitting organic material.

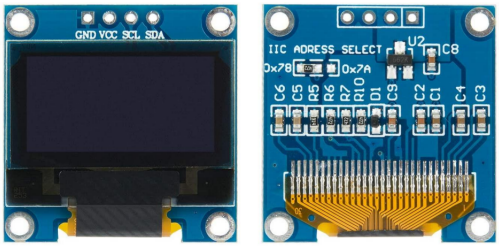

Figure 3.5: OLED Display Image

OPERATING VOLTAGE	3.3V	RESOLUTION	64 × 128 pixels
COMMUNICATION INTERFACE	4-wire SPI, I2C	DISPLAY SIZE	14.70 × 29.42mm
DISPLAY PANEL	OLED	PIXEL SIZE	0.15 × 0.15mm
DRIVER	SH1107	DIMENSIONS	30.0 × 39.5mm

Figure 3.6: OLED Display Specification

In this project a OLED display of 64X128 pixels resolution with 3.3V operating voltage has been used. The specification of OLED that used in this project is shown in Figure 3.6.

3.1.4 Rain Water Level Sensor

Rain water level sensor is a device which can used to detect the level or drop of water. In this weather station project, the rain water level sensor is used to detect the presence of rain. This sensor contains a series of wire which is attached to the surface of this device and is used to detect the level or presence of water droplets. The output of the water sensing is converted to analog signal which can easily be processed by Arduino.

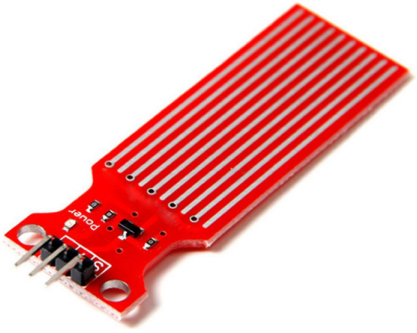

Figure 3.7: Rain Water Level Sensor Image

Operating Voltage and Current	DC 3-5V, less than 20mA
Sensor Type	Analog
Detection Area	40mm x 16mm
Dimension	60 x 19.5 x 2mm

Figure 3.8: Rain Water Level Sensor Specification

3.1.5 Wind Speed and Direction

The measuring of wind speed and direction have significant role in forecasting the weather. For measuring the behavior of wind anemometer device is used in most of the weather station. In this project simulating the working of anemometer, a DC motor has been used. The concept of calculating wind speed is based on the rotation of motor. According to [10] a single rotation of motor in a second is equivalent to 2.4km/hour. By reading the digital output value the wind speed can be estimated.

In order to simulate the direction of wind a $10K\Omega$ potentiometer is used. By placing the potentiometer node towards the south direction it is possible to determine the wind direction due to the rotation of the potentiometer.

3.2 Circuit Diagram

Figure 3.9 and 3.10 shows the circuit and connection digram of remote station.

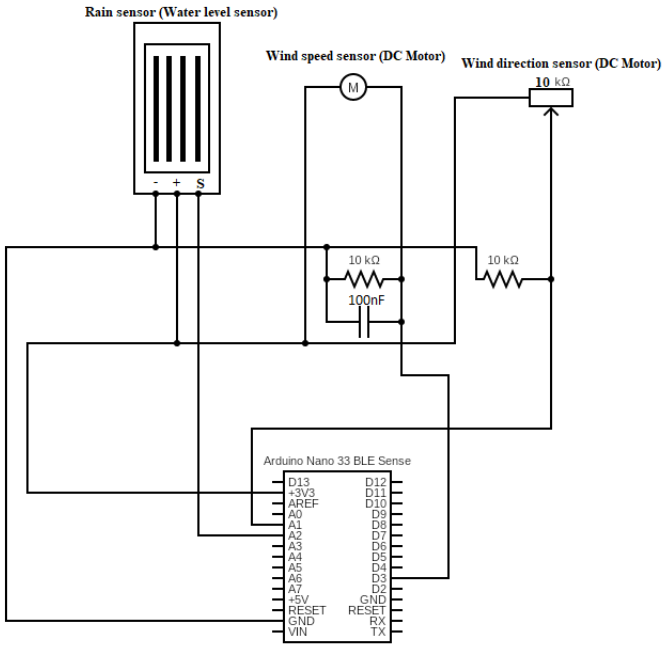

Figure 3.9: Circuit Diagram of Remote Station

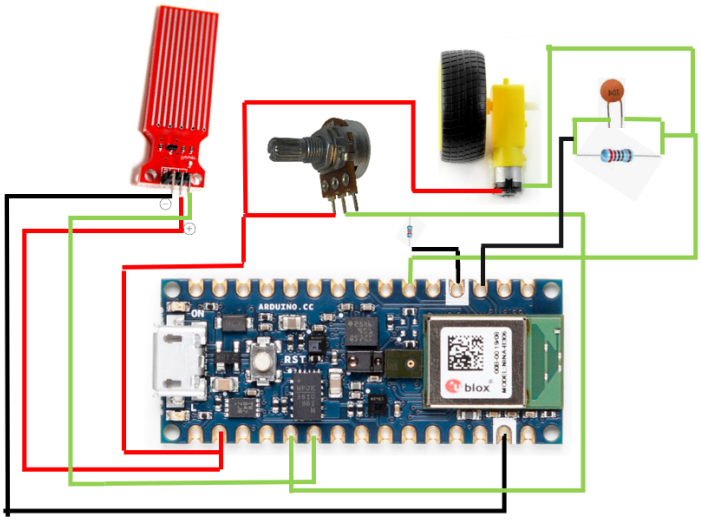

Figure 3.10: Connection Diagram of Remote Station

Figure 3.11 and 3.12 shows the circuit and connection digram of base station.

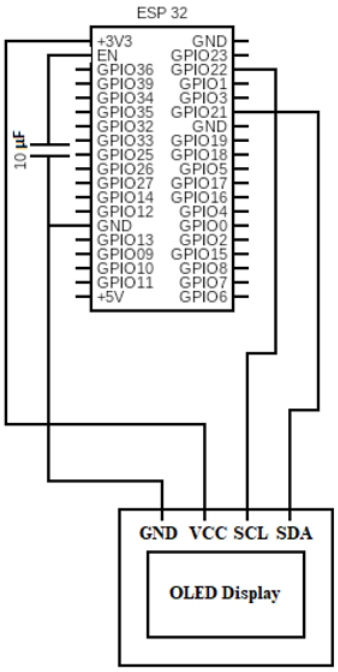

Figure 3.11: Circuit Diagram of Base Station

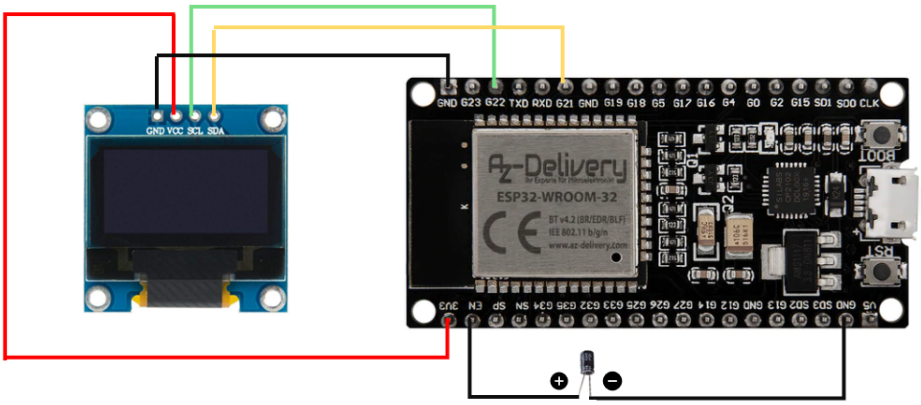

Figure 3.12: Connection Diagram of Base Station

3.3 Power Tree

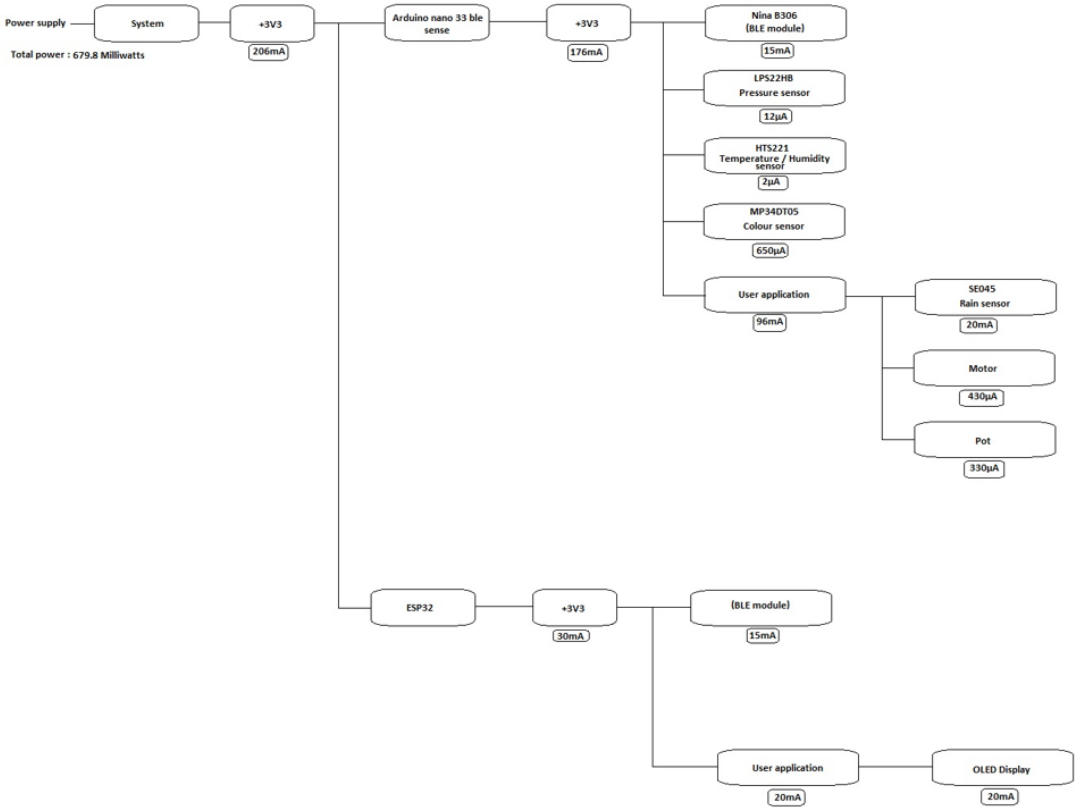

Figure 3.13: Power Tree

Chapter 4

Software

In order to develop firmware for this project, Arduino IDE is being used. The remote and base station applications are developed using C++ programming language.

4.1 System Implementation

4.1.1 Remote Station

The Arduino IDE programming interface gives a way to setup and loop through the application function. In the case of remote station, the setup function initializes the BLE module to start Bluetooth communication. In order to send weather data, a BLEService with name *weatherService* and id $180F$ and BLEStringCharacteristic with name *weatherChar* and id $2A19$ with options BLERead and BLENotify is created. The BLEService is then registered to start Bluetooth communication. Like the BLE module, other sensor modules such as APDS9960, HTS221 and LPS22HB are initialized to start sensing of colour, temperature, humidity and pressure respectively. The versions of the libraries used in this project is shown in Table 4.1.

Once BLE and sensor modules are initialized and started, the application continuously checks for connection from a nearby base station. When a connection is available, it calls sensor routines to read corresponding sensor values. Sensor data are collected only after every 200 milliseconds to avoid load on the device and to smooth data transfer. Once the sensor data is available it sends them to the base station. The flowchart of remote station operation is shown in Figure 4.1

Table 4.1: Remote Station Development Library Versions

Arduino Mbed OS Nano Boards	3.1.1
ArduinoBLE	1.3.1
Arduino_APDS9960	1.0.4
Arduino_HTS221	1.0.0
Arduino_LPS22HB	1.0.2

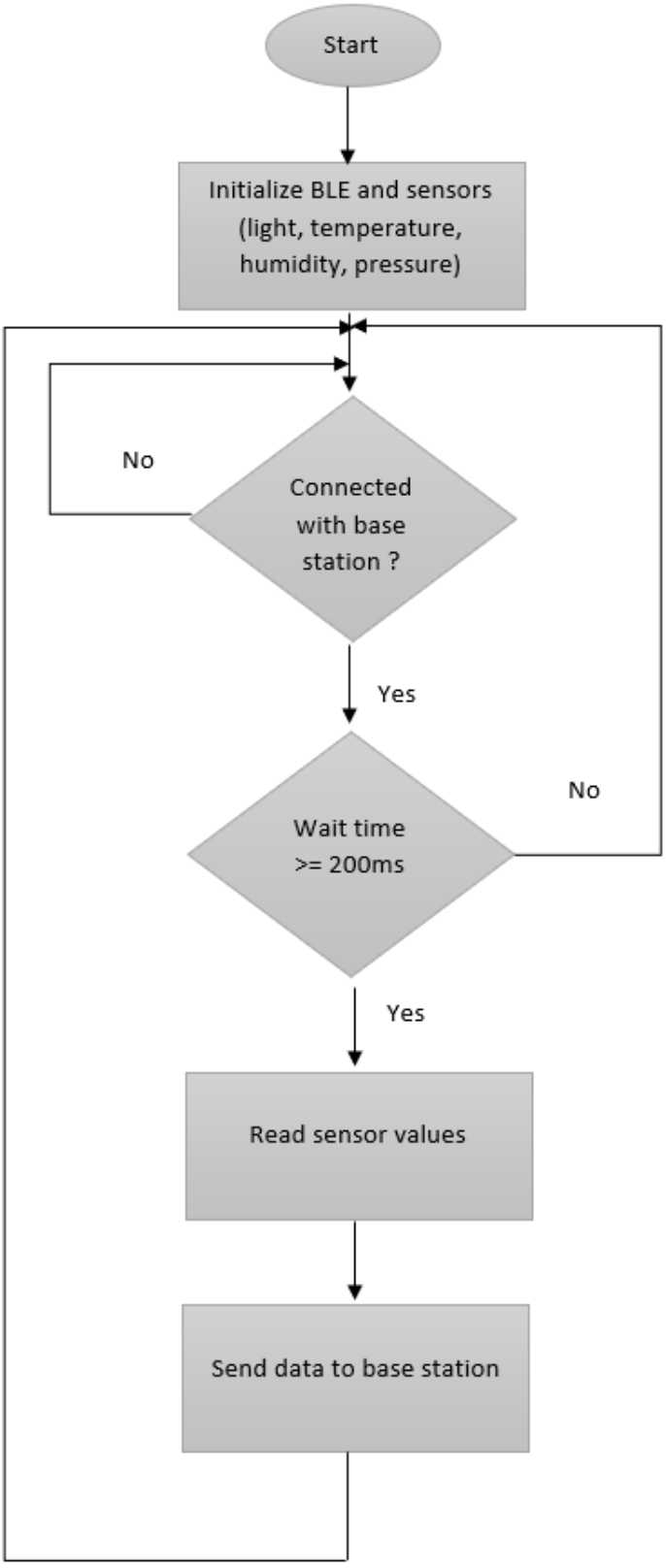

Figure 4.1: Flowchart of Remote Station

4.1.2 Base Station

The base station application in this project is more complex and large compared to the remote station application. The libraries used and their versions are shown in Table 4.2.

Table 4.2: Base Station Development Library Versions

esp32	1.0.6
Adafruit GFX Library	1.11.1
Adafruit SSD1306	2.5.3

Similar to the remote station, the base station also contains a setup phase and application loop. In the setup phase all devices and sensor modules are initialized. The BLEDevice setup has the same $serviceUUID$ and $charUUID$ as that of the remote station, so that it can connect to the remote station. The Wi-Fi module is initiated by providing an SSID and password as shown in Listing 4.1.

Listing 4.1: Wi-Fi connection code snippet

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
}
Serial.println("WIFI CONNECTED");
```

Once the Wi-Fi connection is completed, the real time clock value is determined, by connecting to NTP server, to decide on weather condition. Listing 4.2 shows the code snippet for initializing and getting real time clock value

Listing 4.2: Initilize and get real time clock

```
// NTP server details for time information.
const char* ntpServer       = "pool.ntp.org";
const long gmtOffset_sec    = 3600;
const int  daylightOffset_sec = 3600;

tm getLocalTime() {
   struct tm timeinfo;
   if(!getLocalTime(&timeinfo)){
      Serial.println("Failed to obtain time");
      return tm();
   }
   Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");

   return timeinfo;
}

// Init and get the time
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
```

```
getLocalTime();
```

After the clock value is available, the OLED display is initialized to display sensor and weather forecast information on it. The OLED is connected to the ESP32 as explained in the previous section.

Once all initializations are complete, the base station application listen on an indefinite loop to receive a connection from a remote station. When a connection is received, the base station reads the sensor values sent from the remotes station. If a connection is not available it scans for a new connection. Listing 4.3 shows the connection procedure.

Listing 4.3: connection to remote BLE server for sensor data

```
// Once connected to the remote BLE Server, read sensor data.
if (connected) {
   // Read the value of the characteristic, the sensor data.
   if(pRemoteCharacteristic->canRead()) {
      std::string value = pRemoteCharacteristic->readValue();
      sensorData = convertToIntArray(value);
      Serial.print("The sensor value was: ");
      Serial.println(value.c_str());
   }
} else if (doScan){
   BLEDevice::getScan()->start(0); // scan after disconnect
}
```

The sensor data collected from the remote station is then displayed on the OLED screen along with the current date and time.

An important function of the base station is the forecast of weather based on the received sensor data. A forecast() function determine the weather condition based on the time, temperature, RGB red and rain sensor values. The obtained condition is then displayed to a new screen of the OLED display. Different icons are declared to match the weather conditions and are displayed on the screen depending on the condition returned. Apart from RGB read, temperature, humidity, pressure and rain sensor values, wind speed and direction values are also displayed to the screen to show better weather condition.

Finally, the base station calculates one hour weather condition by storing temperature, RGB red and rain sensor values. These values are stored in an array of size 60, which means the values are stored in each minute so that one hour value will be available after each hour. Once the array is full, the values are added from the beginning. For determining the one hour forecast, average values are taken to determine the one hour forecast using the same forecast() function. Figure 4.2 shows the flowchart of base station.

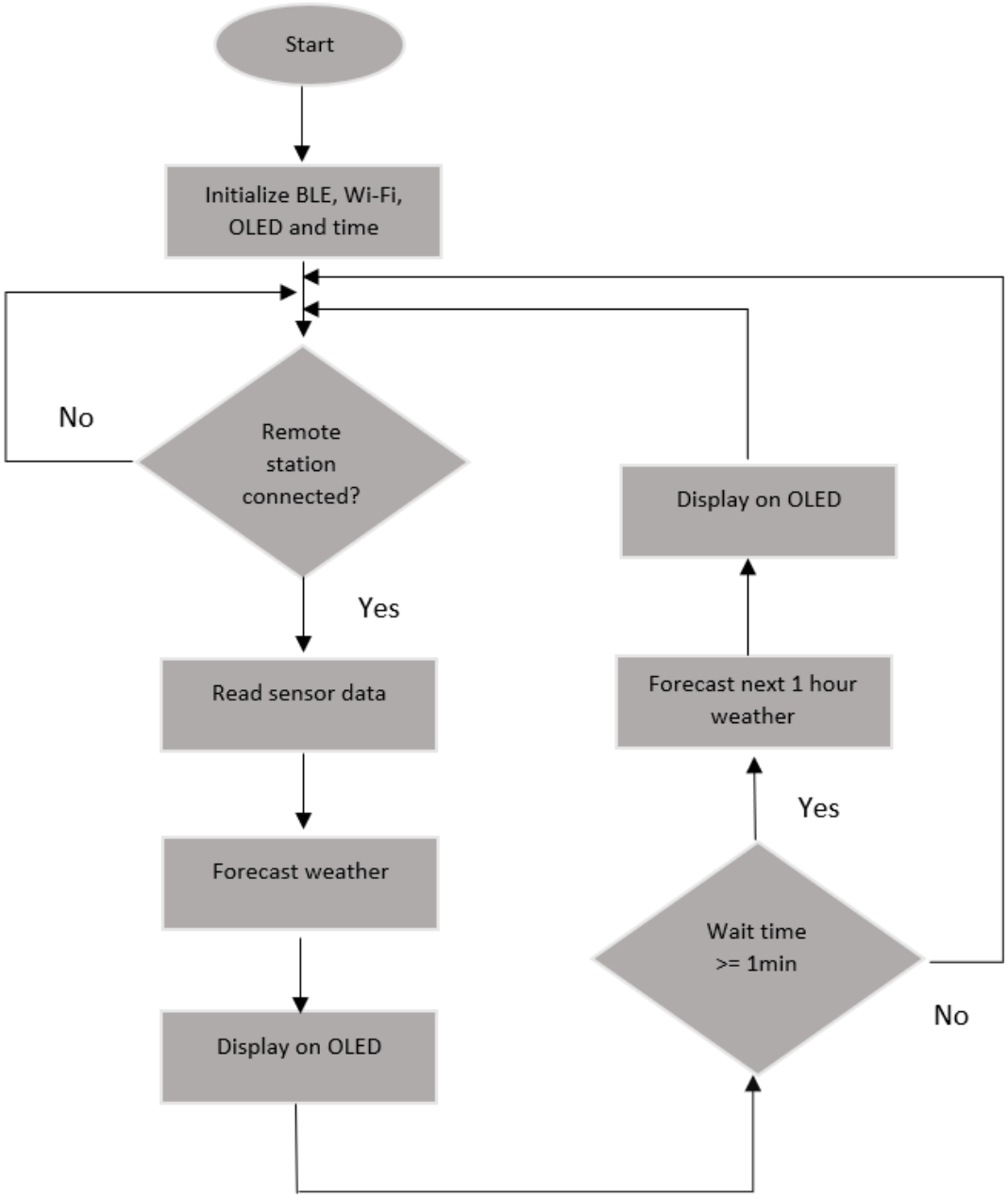

Figure 4.2: Flowchart of Base Station

Chapter 5

Result

Once the base station reads the sensor values and forecast the weather, it displays them to the OLED panel. The display consists of two screens as explained in the following sections:

5.1 Time and Sensor Value Display

In this screen the top line displays the real time clock value which is updated every second. The value contains the date followed by the current time. Depending on the current hour of the day either "Daytime" or "Night" is displayed in the same line. For the project, it is assumed that from morning 6 am to evening 6 pm has been considered as daytime and remaining time considered as night. Followed by the line with real time the screen displays received sensor values such as RGB red, temperature, humidity, pressure, rain and wind. Figure 5.1 shows an example display on the OLED of this screen.

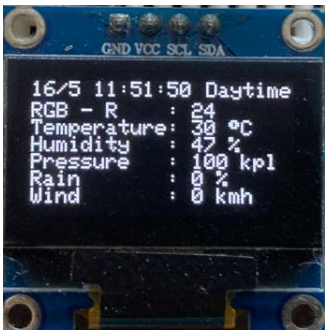

Figure 5.1: OLED Display with Time and Sensor Values

5.2 Weather Forecast Display

The second display of OLED is used to show the calculated current weather forecast along with next one hour weather forecast. Depending on the received RGB red, Temperature and Rain values different weather condition has been determined.

Following subsection discuss about variation of weather condition depending on the previously mentioned sensor values.

5.2.1 Determining Presence of Sun

The RGB Red or R value is a good indicator of presence of sun, as high value of R value indicates high brightness and low R value gives dull. As a result weather condition has been divided into four: sunny($R \geq 255$), partially cloudy ($R > 100$ and $R < 255$), mostly cloudy ($R > 0$ and $R < 100$) and cloudy ($R < 10$). It is to be noted that the R value can range from 0 to above 255 depending on the light condition. Figure 5.2 and 5.3 shows Suuny and Partially cloudy weather conditions at the top of the display.

Figure 5.2: Sunny Weather Condition

Figure 5.3: Partially Cloudy Weather Condition

5.2.2 Determining Presence of Rain and Snow

The rain sensor value (r) shows the presence or absence of rain. Depending on the sensor value received 3 conditions has been determined: Heavy rain($r > 500$), Rain ($0 < r < 500$) and No rain ($r = 0$). The above condition is true if the temperature is above 0^oC. If the temperature is 0^oC or below shows the presence of snow. The Figures 5.4 and 5.5 shows snow and rain conditions respectively.

Figure 5.4: Cloudy with Snow Weather Condition

Figure 5.5: Cloudy with Rain Weather Condition

5.2.3 Determining Wind Speed and Direction

Apart from sun and rain conditions presence of wind along with wind speed has been determined based on the wind speed(ws) and wind direction(wd) values. Depending on the wind speed values 3 condition has been assumed : Heavy Wind (ws $>$ 15km/h), Windy (0 $<$ ws $<$ 15km/h) and No wind (ws $=$ 0). Along with wind speed wind direction also determined based on the received wind direction value. The wind information is provided just below the current weather forecast display line. Figure 5.6 and 5.7 shows No wind and heavy wind condition towards the direction of North respectively.

Figure 5.6: No Wind Condition

Figure 5.7: Wind with Direction towards North

5.2.4 Next One Hour Weather Forecast

Figure 5.8: Next one hour Weather Forecast

One of the key thing in this project is calculation of weather for the next hour depending on the stored one hour value of stored sensor values. In order to do this, temperature, rain and RGB Red values are stored in each one minute and an average is calculated for these sensor values for forecasting. It is assumed that the average of last one hour gives an estimate of next one hour weather condition. The next one hour weather forecast is shown at the bottom of the OLED display as shown in the Figure 5.8 shows next one hour weather forecast at the bottom.

Chapter 6

Conclusion

In this project a Weather Station using Arduino Nano 33 BE Sense and ESP32 has been successfully implemented. The results shows both the remote station and the base station works perfectly as expected. It is found that Arduino Nano 33 BE Sense is well suitable for implementing remote station as weather forecast station. Due to its inbuilt sensors most of the atmospheric conditions such as temperature, pressure, humidity and light are sensed without much effort at low cost. The Bluetooth module available in this board is useful for communicating with a base station. In the case of base station ESP32 performed really well. As ESP32 supports large program and SRAM memory it is suitable for running large applications such as back-end server for processing sensor data.

The weather station implemented in this project shown to forecast the weather with adequate reliably. With respect to change in the RGB red value the system can approximately determine the presence of sunlight condition. The rain condition is also determined based on the rain water level sensor with better accuracy. Further, wind speed and direction are also provided to make the system more useful. Unlike other weather forecasting systems, the current forecaster does not consider temperature, humidity and pressure for weather condition determination. Although this system can be easily extended to include these sensor values to produce better results. Finally this weather station also calculates next one hour weather condition based on the average value of stored sensor values. Further consideration can be taken to extend this logic to incorporate complex machine learning techniques to produce better future weather forecasts.

Chapter 7

Appendix

7.1 Remote Station Code

```
#include <Arduino_APDS9960.h>
#include <Arduino_HTS221.h>
#include <Arduino_LPS22HB.h>
#include <ArduinoBLE.h>

BLEService weatherService("180F");
BLEStringCharacteristic weatherChar("2A19", BLERead | BLENotify, 30);
String oldSensorLevel = "0,0,0,0,0,0,0"; // last sensor readings
long previousMillis = 0; // last time the sensor level was read, in ms

int RainSensor = A2;

const int WindRecordTime = 3; //Measuring Time (Seconds)
const int windSensorPin = D3; //Interrupt Pin
int InterruptCounter;

int winddirection=A1;

void setup() {
  Serial.begin(9600);

  if (!APDS.begin()) {
    Serial.println("Error initializing APDS-9960 sensor.");
  }
  if (!HTS.begin()) {
    Serial.println("Failed to initialize humidity temperature sensor!");
    while (1);
  }
  if (!BARO.begin()) {
    Serial.println("Failed to initialize pressure sensor!");
    while (1);
  }
  if (!BLE.begin()) {
```

```
    Serial.println("starting BLE failed!");
    while (1);
  }

  BLE.setLocalName("WeatherMonitor");
  BLE.setAdvertisedService(weatherService); // add the service UUID
  weatherService.addCharacteristic(weatherChar); // add the battery
      level characteristic
  BLE.addService(weatherService); // Add the battery service
  weatherChar.writeValue(oldSensorLevel); // set initial value for this
      characteristic

  BLE.advertise();
  Serial.println("Bluetooth® device active, waiting for
      connections...");
}

void loop() {
  int R, T, H, P, r, ws, wd;
  BLEDevice central = BLE.central();
  if (central) {
    // connected to base station
    Serial.print("Connected to central: ");

    // print the base station BT address:
    Serial.println(central.address());

    // check the sensor level every 200ms
    // while the central is connected:
    while (central.connected()) {
      long currentMillis = millis();
      // if 200ms have passed, check the sensor level:
      if (currentMillis - previousMillis >= 200) {
        previousMillis = currentMillis;

        R = (int)get_colour_R();
        T = (int)get_temperature();
        H = (int)get_humidity();
        P = (int)get_pressure();
        r = (int)get_rain();
        ws = (int)get_wind();
        wd = (int)get_windDirection();

        updateSensorLevel(R, T, H, P, r, ws, wd);
        Serial.println("");
      }
    }
  }
}
```

```
void updateSensorLevel(int R, int T, int H, int P, int r, int ws, int wd)
    {
   String sensorLevel = String(R) + "," + String(T) + "," +
                   String(H) + "," + String(P) + "," +
                   String(r) + "," + String(ws) + "," +
                   String(wd);
   Serial.print("Sensor Level % is now: "); // print sensor readings
   Serial.println(sensorLevel);
   weatherChar.writeValue(sensorLevel); // update the weather level
       characteristic
}

// Get the RGB - R value.
int get_colour_R() {
   // check if a color reading is available
   while (! APDS.colorAvailable()) {
      delay(5);
   }
   int r, g, b;
   APDS.readColor(r, g, b);

   Serial.print("RGB - R : ");
   Serial.println(r);

   return r;
}

// Get the temperature value.
float get_temperature() {
   float temperature = HTS.readTemperature();

   Serial.print("Temperature = ");
   Serial.print(temperature);
   Serial.println(" °C");
   return temperature;

}

// Get the humidity value.
float get_humidity() {
   float humidity  = HTS.readHumidity();

   Serial.print("Humidity = ");
   Serial.print(humidity);
   Serial.println(" %");
   return humidity;
}
```

```
// Get the pressure value.
float get_pressure() {
   float pressure = BARO.readPressure();

   Serial.print("Pressure = ");
   Serial.print(pressure);
   Serial.println(" kPa");
   return pressure;
}

// Get the rain value.
int get_rain() {
   int value = analogRead(RainSensor);//Reads the Value of RainSensor.
   Serial.print("Rain - RainSensor value is :");
   Serial.println(value);
   return value;
}

// Get the wind speed value.
float get_wind() {
   //https://www.aeq-web.com/arduino-anemometer-wind-sensor/
   float WindSpeed = wind_meassure();
   Serial.print("Wind Speed: ");
   Serial.print(WindSpeed);    //Speed in km/h
   Serial.println(" km/h");
   return WindSpeed;
}


float wind_meassure() {
   InterruptCounter = 0;
   attachInterrupt(digitalPinToInterrupt(windSensorPin), countup, RISING);
   delay(1000 * WindRecordTime);
   detachInterrupt(digitalPinToInterrupt(windSensorPin));
   float WindSpeed = (float)InterruptCounter / (float)WindRecordTime *
       2.4;
   return WindSpeed;
}

void countup() {
   InterruptCounter++;
}

// Get the wind direction value.
int get_windDirection() {
   int news = analogRead(winddirection);//Reads the Value of LDR(light).

   Serial.print("winddirection value is :");//Prints the value of LDR to
       Serial Monitor.
```

```
   Serial.println(news);
   return news;
}
```

7.2 Base Station Code

```
#include <Wire.h>
#include <WiFi.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#include <stdio.h>
#include <algorithm>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

#include "time.h"
#include "BLEDevice.h"


// Global definitions.

#define OLED_SCREEN_WIDTH 128 // OLED width, in pixels
#define OLED_SCREEN_HEIGHT 64 // OLED height, in pixels

#define RED_COLOUR_MAX 255
#define RED_COLOUR_MED 100
#define RED_COLOUR_MIN 10

#define TEMP_MIN 0

#define RAIN_AREA 500

#define WIND_MAX 15
#define WIND_MIN 0

#define HOUR_SIX 6
#define HOUR_TWELVE 12
#define HOUR_EIGHTEEN 18
#define HOUR_TWENTYFOUR 24

// WiFi SSID and Password.
const char* ssid     = "test";
const char* password = "test";
```

```
// NTP server details for time information.
const char* ntpServer        = "pool.ntp.org";
const long gmtOffset_sec     = 3600;
const int  daylightOffset_sec = 3600;

// Remote station BLE service to connect to.
static BLEUUID serviceUUID("180F");
// The characteristic of the remote service.
static BLEUUID charUUID("2A19");

static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* device;

// Vector to save up-to-date sensor data.
std::vector<int> sensorData{0, 0, 0, 0, 0, 0, 0};

// Define icons for weather display.
//
    https://www.hackster.io/shubhamsantosh99/weather-monitoring-on-oled-c1d4a8
const unsigned char sunny [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xfc,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xfc, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xfc,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xf7, 0xfc, 0xff, 0x3f, 0xff, 0x80, 0xff, 0xff, 0xf3, 0xfc,
    0xff, 0x3f, 0xff, 0x80,
0xff, 0xff, 0xf3, 0xfc, 0xfe, 0x7f, 0xff, 0x80, 0xff, 0xff, 0xf9, 0xfc,
    0xfe, 0x7f, 0xff, 0x80,
0xff, 0xff, 0xf8, 0xfd, 0xfc, 0xff, 0xff, 0x80, 0xff, 0xff, 0xfc, 0xff,
    0xfc, 0xff, 0xff, 0x80,
0xff, 0xff, 0xfc, 0xff, 0xfd, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xf3, 0xff, 0xf0, 0x3f, 0xff, 0x7f, 0x80, 0xff, 0xf1, 0xff, 0x80,
    0x0f, 0xfc, 0x3f, 0x80,
0xff, 0xf8, 0x7f, 0x07, 0x83, 0xf8, 0x7f, 0x80, 0xff, 0xfc, 0x3e, 0x3f,
    0xe1, 0xe1, 0xff, 0x80,
0xff, 0xff, 0x1c, 0x7f, 0xf9, 0xe3, 0xff, 0x80, 0xff, 0xff, 0xf8, 0xff,
    0xfc, 0xff, 0xff, 0x80,
0xff, 0xff, 0xf9, 0xff, 0xfc, 0x7f, 0xff, 0x80, 0xff, 0xff, 0xf1, 0xff,
    0xfe, 0x7f, 0xff, 0x80,
```

```
0xff, 0xff, 0xf3, 0xff, 0xfe, 0x3f, 0xff, 0x80, 0xff, 0xff, 0xf3, 0xff,
    0xff, 0x3f, 0xff, 0x80,
0xff, 0xff, 0xf3, 0xff, 0xff, 0x3f, 0xff, 0x80, 0xff, 0xff, 0xe7, 0xff,
    0xff, 0x3f, 0xff, 0x80,
0xff, 0x80, 0x67, 0xff, 0xff, 0x38, 0x0f, 0x80, 0xff, 0x80, 0xe7, 0xff,
    0xff, 0x38, 0x0f, 0x80,
0xff, 0xff, 0xe7, 0xff, 0xff, 0x3f, 0xff, 0x80, 0xff, 0xff, 0xf3, 0xff,
    0xff, 0x3f, 0xff, 0x80,
0xff, 0xff, 0xf3, 0xff, 0xff, 0x3f, 0xff, 0x80, 0xff, 0xff, 0xf3, 0xff,
    0xfe, 0x7f, 0xff, 0x80,
0xff, 0xff, 0xf1, 0xff, 0xfe, 0x7f, 0xff, 0x80, 0xff, 0xff, 0xf9, 0xff,
    0xfc, 0x7f, 0xff, 0x80,
0xff, 0xff, 0xbc, 0xff, 0xf8, 0xef, 0xff, 0x80, 0xff, 0xff, 0x1c, 0x7f,
    0xf1, 0xe3, 0xff, 0x80,
0xff, 0xfc, 0x3e, 0x1f, 0xe3, 0xf0, 0xff, 0x80, 0xff, 0xf8, 0xff, 0x03,
    0x07, 0xf8, 0x7f, 0x80,
0xff, 0xf1, 0xff, 0xc0, 0x0f, 0xfe, 0x3f, 0x80, 0xff, 0xff, 0xff, 0xf8,
    0x7f, 0xff, 0x7f, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xfc, 0xff,
    0xf9, 0xff, 0xff, 0x80,
0xff, 0xff, 0xfc, 0xff, 0xfc, 0xff, 0xff, 0x80, 0xff, 0xff, 0xf9, 0xfc,
    0xfc, 0xff, 0xff, 0x80,
0xff, 0xff, 0xf9, 0xfc, 0xfe, 0x7f, 0xff, 0x80, 0xff, 0xff, 0xf3, 0xfc,
    0xfe, 0x7f, 0xff, 0x80,
0xff, 0xff, 0xf3, 0xfc, 0xff, 0x3f, 0xff, 0x80, 0xff, 0xff, 0xf7, 0xfc,
    0xff, 0x3f, 0xff, 0x80,
0xff, 0xff, 0xff, 0xfc, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xfc,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xfc, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80
};
const unsigned char partly_cloudy [] PROGMEM = {
0xff, 0xff, 0xf8, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xf8, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xf8, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xf8, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xf8, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xf8, 0xff,
    0xf7, 0xff, 0xff, 0x80,
0xfe, 0x7f, 0xf8, 0xff, 0xe3, 0xff, 0xff, 0x80, 0xfc, 0x3f, 0xff, 0xff,
    0xe1, 0xff, 0xff, 0x80,
0xfc, 0x1f, 0xff, 0xff, 0xc3, 0xff, 0xff, 0x80, 0xfe, 0x0f, 0xff, 0xff,
    0x87, 0xff, 0xff, 0x80,
0xff, 0x0f, 0xff, 0xff, 0x07, 0xff, 0xff, 0x80, 0xff, 0x8f, 0xc0, 0x3f,
    0x8f, 0xff, 0xff, 0x80,
0xff, 0xdf, 0x00, 0x0f, 0xdf, 0xff, 0xff, 0x80, 0xff, 0xfe, 0x00, 0x03,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xfc, 0x06, 0x01, 0xff, 0xff, 0xff, 0x80, 0xff, 0xf8, 0x3f, 0xc1,
    0xff, 0xff, 0xff, 0x80,
```

```
0xff, 0xf0, 0x7f, 0xe0, 0xff, 0xff, 0xff, 0x80, 0xff, 0xf0, 0xff, 0xf0,
    0x7f, 0xff, 0xff, 0x80,
0xff, 0xe1, 0xff, 0xf8, 0x7f, 0xff, 0xff, 0x80, 0xff, 0xe1, 0xff, 0xfc,
    0x7f, 0xff, 0xff, 0x80,
0xff, 0xe3, 0xff, 0xfc, 0x3f, 0xff, 0xff, 0x80, 0xff, 0xe3, 0xff, 0xfc,
    0x3f, 0xff, 0xff, 0x80,
0x01, 0xc3, 0xff, 0x00, 0x3f, 0xff, 0xff, 0x80, 0x01, 0xc3, 0xf8, 0x00,
    0x3f, 0xff, 0xff, 0x80,
0x01, 0xc3, 0xe0, 0x00, 0x1f, 0xff, 0xff, 0x80, 0x01, 0xe3, 0xc0, 0x00,
    0x07, 0xff, 0xff, 0x80,
0xff, 0xe3, 0x80, 0xfe, 0x03, 0xff, 0xff, 0x80, 0xff, 0xe3, 0x03, 0xff,
    0x83, 0xff, 0xff, 0x80,
0xff, 0xe0, 0x0f, 0xff, 0xc1, 0xff, 0xff, 0x80, 0xff, 0xf0, 0x1f, 0xff,
    0xe0, 0xff, 0xff, 0x80,
0xff, 0xf0, 0x3f, 0xff, 0xf0, 0xff, 0xff, 0x80, 0xff, 0xf8, 0x3f, 0xff,
    0xf8, 0x7f, 0xff, 0x80,
0xff, 0xf8, 0x7f, 0xff, 0xfc, 0x7f, 0xff, 0x80, 0xff, 0x88, 0x7f, 0xff,
    0xfc, 0x27, 0xff, 0x80,
0xfc, 0x00, 0xff, 0xff, 0xfc, 0x00, 0x7f, 0x80, 0xf8, 0x00, 0xff, 0xff,
    0xfe, 0x00, 0x3f, 0x80,
0xf0, 0x00, 0xff, 0xff, 0xfe, 0x00, 0x1f, 0x80, 0xe0, 0x70, 0xff, 0xff,
    0xfe, 0x3c, 0x0f, 0x80,
0xc1, 0xfc, 0xff, 0xff, 0xfe, 0x7f, 0x07, 0x80, 0x83, 0xff, 0xff, 0xff,
    0xff, 0xff, 0x87, 0x80,
0x87, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc3, 0x80, 0x8f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xc3, 0x80,
0x8f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe3, 0x80, 0x0f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xe3, 0x80,
0x0f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe3, 0x80, 0x0f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xe3, 0x80,
0x0f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe3, 0x80, 0x8f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xe3, 0x80,
0x8f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc3, 0x80, 0x87, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xc7, 0x80,
0xc3, 0xff, 0xff, 0xff, 0xff, 0xff, 0x87, 0x80, 0xe1, 0xff, 0xff, 0xff,
    0xff, 0xff, 0x0f, 0x80,
0xe0, 0x7f, 0xff, 0xff, 0xff, 0xfc, 0x1f, 0x80, 0xf0, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x3f, 0x80,
0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0x80, 0xff, 0x80, 0x00, 0x00,
    0x00, 0x03, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80
};
const unsigned char sunny_cloudy_rain [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xef, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xe7,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xe7, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0x9f, 0xe7,
    0xfb, 0xff, 0xff, 0x80,
```

```
0xff, 0xff, 0x9f, 0xe7, 0xf3, 0xff, 0xff, 0x80, 0xff, 0xff, 0xcf, 0xe7,
    0xf3, 0xff, 0xff, 0x80,
0xff, 0xff, 0xcf, 0xef, 0xe7, 0xff, 0xff, 0x80, 0xff, 0xff, 0xe7, 0xff,
    0xe7, 0xff, 0xff, 0x80,
0xff, 0xff, 0xe7, 0xff, 0xef, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0x01, 0xff, 0xff, 0xff, 0x80, 0xff, 0xef, 0xfc, 0x00,
    0x7f, 0xf7, 0xff, 0x80,
0xff, 0xc7, 0xf8, 0x7c, 0x1f, 0xc7, 0xff, 0x80, 0xff, 0xe1, 0xf1, 0xff,
    0x0f, 0x8f, 0xff, 0x80,
0xff, 0xf9, 0xe3, 0xff, 0xcf, 0x1f, 0xff, 0x80, 0xff, 0xfd, 0xc7, 0xff,
    0xe7, 0x7f, 0xff, 0x80,
0xff, 0xff, 0xcf, 0xff, 0xe3, 0xff, 0xff, 0x80, 0xff, 0xff, 0x8f, 0xff,
    0xf3, 0xff, 0xff, 0x80,
0xff, 0xff, 0x9f, 0xff, 0xf1, 0xff, 0xff, 0x80, 0xff, 0xff, 0x9f, 0xff,
    0xf9, 0xff, 0xff, 0x80,
0xff, 0xff, 0x3f, 0xff, 0xf9, 0xff, 0xff, 0x80, 0xff, 0xff, 0x3f, 0xff,
    0x01, 0xff, 0xff, 0x80,
0xff, 0xff, 0x3f, 0xfc, 0x01, 0xff, 0xff, 0x80, 0xff, 0x07, 0x3f, 0xf0,
    0x20, 0xc0, 0xff, 0x80,
0xff, 0x07, 0x3f, 0xe1, 0xfc, 0x61, 0xff, 0x80, 0xff, 0xff, 0x3e, 0x03,
    0xfe, 0x3f, 0xff, 0x80,
0xff, 0xff, 0x38, 0x07, 0xff, 0x1f, 0xff, 0x80, 0xff, 0xff, 0x80, 0x0f,
    0xff, 0x9f, 0xff, 0x80,
0xff, 0xff, 0x83, 0xff, 0xff, 0xcf, 0xff, 0x80, 0xff, 0xff, 0x87, 0xff,
    0xff, 0xc7, 0xff, 0x80,
0xff, 0xff, 0x8f, 0xff, 0xff, 0xc3, 0xff, 0x80, 0xff, 0xff, 0x9f, 0xff,
    0xff, 0xf1, 0xff, 0x80,
0xff, 0xf9, 0x9f, 0xff, 0xff, 0xf8, 0xff, 0x80, 0xff, 0xf1, 0xbf, 0xff,
    0xff, 0xfc, 0xff, 0x80,
0xff, 0xc3, 0x3f, 0xff, 0xff, 0xfe, 0x7f, 0x80, 0xff, 0xcf, 0x3f, 0xff,
    0xff, 0xfe, 0x7f, 0x80,
0xff, 0xff, 0x3f, 0xff, 0xff, 0xfe, 0x7f, 0x80, 0xff, 0xff, 0x3f, 0xff,
    0xff, 0xfe, 0x7f, 0x80,
0xff, 0xff, 0x9f, 0xff, 0xff, 0xfe, 0x7f, 0x80, 0xff, 0xff, 0x9f, 0xff,
    0xff, 0xfe, 0x7f, 0x80,
0xff, 0xff, 0x8f, 0xff, 0xff, 0xfc, 0xff, 0x80, 0xff, 0xff, 0xcf, 0xff,
    0xff, 0xfc, 0xff, 0x80,
0xff, 0xff, 0xe3, 0xff, 0xff, 0xf9, 0xff, 0x80, 0xff, 0xff, 0xf0, 0x7f,
    0xff, 0xc3, 0xff, 0x80,
0xff, 0xff, 0xf8, 0x3f, 0xff, 0x87, 0xff, 0x80, 0xff, 0xff, 0xff, 0x7f,
    0xff, 0xdf, 0xff, 0x80,
0xff, 0xff, 0xff, 0xfb, 0xfd, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xdf, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
```

```
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80
};
const unsigned char cloudy [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xfe,
    0x07, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xf8, 0x00, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xf0,
    0x00, 0x7f, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xe0, 0x00, 0x3f, 0xff, 0xf0, 0xff, 0xff, 0xff, 0x80,
    0x00, 0x1f, 0xff, 0xf0,
0xff, 0xff, 0xff, 0x00, 0x00, 0x1f, 0xff, 0xf0, 0xff, 0xff, 0xff, 0x00,
    0x00, 0x0f, 0xff, 0xf0,
0xff, 0xff, 0xfe, 0x00, 0x00, 0x0f, 0xff, 0xf0, 0xff, 0xff, 0xf0, 0x00,
    0x00, 0x07, 0xff, 0xf0,
0xff, 0xff, 0xc0, 0x00, 0x00, 0x03, 0xff, 0xf0, 0xff, 0xff, 0xc0, 0x00,
    0x00, 0x00, 0x3f, 0xf0,
0xff, 0xff, 0x80, 0x00, 0x00, 0x00, 0x1f, 0xf0, 0xff, 0xff, 0x80, 0x00,
    0x00, 0x00, 0x0f, 0xf0,
0xff, 0xff, 0x80, 0x00, 0x00, 0x00, 0x07, 0xf0, 0xff, 0xff, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xf0,
0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x03, 0xf0, 0xff, 0xff, 0x00, 0x00,
    0x00, 0x00, 0x03, 0xf0,
0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x03, 0xf0, 0xff, 0xff, 0x00, 0x00,
    0x00, 0x00, 0x03, 0xf0,
0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x03, 0xf0, 0xff, 0xff, 0x80, 0x00,
    0x00, 0x00, 0x03, 0xf0,
0xff, 0xff, 0x80, 0x00, 0x00, 0x00, 0x07, 0xf0, 0xff, 0xf8, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xf0,
0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xf0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x00, 0x1f, 0xf0,
0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xf0, 0xff, 0x80, 0x00, 0x00,
    0x00, 0x00, 0x7f, 0xf0,
0xff, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xf0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x03, 0xff, 0xf0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xf0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x03, 0xff, 0xf0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xf0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x03, 0xff, 0xf0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xf0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x07, 0xff, 0xf0,
0xff, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xf0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x07, 0xff, 0xf0,
0xff, 0x80, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xf0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x1f, 0xff, 0xf0,
```

```
0xff, 0xe0, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0
};
const unsigned char drizzle [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xfc, 0x01,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xf8, 0x00, 0x3f, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xc0, 0x00,
    0x07, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xc0, 0x00, 0x07, 0xff, 0xff, 0xe0, 0xff, 0xfe, 0x07, 0xff,
    0x80, 0xff, 0xff, 0xe0,
0xff, 0xfc, 0x1f, 0xff, 0xe0, 0x7f, 0xff, 0xe0, 0xff, 0xf8, 0x3f, 0xff,
    0xf8, 0x7f, 0xff, 0xe0,
0xff, 0xf8, 0x3f, 0xff, 0xf8, 0x7f, 0xff, 0xe0, 0xff, 0xf0, 0x3f, 0xff,
    0xfc, 0x7f, 0xff, 0xe0,
0xff, 0xe0, 0x3f, 0xff, 0xfe, 0x01, 0xff, 0xe0, 0xff, 0xe0, 0xff, 0xff,
    0xfe, 0x00, 0x1f, 0xe0,
0xff, 0xe0, 0xff, 0xff, 0xff, 0x00, 0x0f, 0xe0, 0xff, 0x81, 0xff, 0xff,
    0xff, 0x00, 0x07, 0xe0,
0xff, 0x81, 0xff, 0xff, 0xff, 0xfe, 0x07, 0xe0, 0xff, 0x81, 0xff, 0xff,
    0xff, 0xff, 0xc3, 0xe0,
0xff, 0x81, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xe0, 0xfe, 0x01, 0xff, 0xff,
    0xff, 0xff, 0xf0, 0xe0,
0xfc, 0x01, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x60, 0xe0, 0x03, 0xff, 0xff,
    0xff, 0xff, 0xf8, 0x20,
0xc1, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x20, 0x83, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xfc, 0x00,
0x83, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x03, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xfc, 0x00,
0x07, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x07, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xfc, 0x00,
0x1f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x1f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xfc, 0x00,
0x1f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x20, 0x1f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xf8, 0x20,
0x07, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x60, 0x03, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xf8, 0x60,
0x83, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xe0, 0xc1, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xc3, 0xe0,
0xe0, 0x7f, 0xff, 0xff, 0xff, 0xf8, 0x07, 0xe0, 0xf8, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xe0,
```

```
0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xe0, 0xfc, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x3f, 0xe0,
0xff, 0xe0, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xf1, 0xff, 0x1f,
    0xfb, 0xff, 0x0f, 0xe0,
0xff, 0x80, 0xfc, 0x0f, 0xe1, 0xfe, 0x0f, 0xe0, 0xff, 0x80, 0xfc, 0x0f,
    0xc1, 0xfc, 0x0f, 0xe0,
0xff, 0x01, 0xf8, 0x1f, 0x81, 0xf8, 0x0f, 0xe0, 0xfe, 0x03, 0xf0, 0x3f,
    0x07, 0xe0, 0x3f, 0xe0,
0xfc, 0x1f, 0xc3, 0xf8, 0x07, 0x80, 0xff, 0xe0, 0xfc, 0x7f, 0xc3, 0xf8,
    0x1f, 0x81, 0xff, 0xe0,
0xfc, 0x7f, 0xc3, 0xf8, 0x3f, 0x87, 0xff, 0xe0, 0xfc, 0xff, 0xc7, 0xfc,
    0x7f, 0x8f, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0
};
const unsigned char mist [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0x3f, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xfe, 0x1f, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xcc, 0x1c, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0x80, 0x00, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xc0, 0x00, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xc0, 0x00, 0x1f, 0xe0,
0xff, 0xff, 0xfc, 0x01, 0xe0, 0x00, 0x0f, 0xe0, 0xff, 0xff, 0xf8, 0x00,
    0x60, 0x00, 0x0f, 0xe0,
0xff, 0xff, 0xe0, 0x00, 0x30, 0x00, 0x1f, 0xe0, 0xff, 0xff, 0xc0, 0x00,
    0x00, 0x00, 0x1f, 0xe0,
0xff, 0xff, 0xc0, 0x00, 0x08, 0x00, 0x1f, 0xe0, 0xff, 0xff, 0x80, 0x00,
    0x00, 0x00, 0x07, 0xe0,
0xff, 0xff, 0x00, 0x00, 0x04, 0x00, 0x07, 0xe0, 0xff, 0xff, 0x00, 0x00,
    0x02, 0x00, 0x07, 0xe0,
0xff, 0xff, 0x00, 0x00, 0x02, 0x00, 0x0f, 0xe0, 0xff, 0xff, 0x00, 0x00,
    0x02, 0x00, 0x3f, 0xe0,
0xff, 0xfe, 0x00, 0x00, 0x03, 0x00, 0x1f, 0xe0, 0xff, 0xf0, 0x00, 0x00,
    0x03, 0x80, 0x0f, 0xe0,
0xff, 0xc0, 0x00, 0x00, 0x00, 0x60, 0x0f, 0xe0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x0c, 0x0f, 0xe0,
0xff, 0x80, 0x00, 0x00, 0x00, 0x07, 0xff, 0xe0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x01, 0xff, 0xe0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xe0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xff, 0xe0,
```

```
0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xe0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xff, 0xe0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xe0, 0xfe, 0x00, 0xbf, 0xff,
    0x9e, 0x00, 0xff, 0xe0,
0xfe, 0x01, 0xff, 0xff, 0xff, 0x80, 0xff, 0xe0, 0xfe, 0x06, 0x00, 0x00,
    0x00, 0xc0, 0xff, 0xe0,
0xff, 0x06, 0x00, 0x00, 0x00, 0x40, 0xff, 0xe0, 0xff, 0x06, 0x00, 0x00,
    0x00, 0xc0, 0xff, 0xe0,
0xff, 0x83, 0x00, 0x00, 0x00, 0xc1, 0xff, 0xe0, 0xff, 0xc7, 0xff, 0xff,
    0xff, 0x81, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0x83, 0xff, 0xe0, 0xff, 0x80, 0x00, 0x00,
    0x60, 0xc7, 0xff, 0xe0,
0xff, 0x00, 0x00, 0x00, 0x40, 0xff, 0xff, 0xe0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0xff, 0xff, 0xe0,
0xff, 0x80, 0x00, 0x00, 0xe0, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xf0, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xe0, 0xff, 0xe0, 0x00, 0x00,
    0x07, 0xff, 0xff, 0xe0,
0xff, 0xe0, 0x00, 0x00, 0x07, 0xff, 0xff, 0xe0, 0xff, 0xf0, 0x00, 0x00,
    0x0f, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0
};
const unsigned char rain [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xc0, 0x3f, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0x00,
    0x0f, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xf8, 0x00, 0x01, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xf0, 0x00,
    0x00, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xf0, 0x00, 0x00, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xe0, 0x00,
    0x00, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xe0, 0x00, 0x00, 0x7f, 0xff, 0xe0, 0xff, 0xff, 0xc0, 0x00,
    0x00, 0x3f, 0xff, 0xe0,
0xff, 0xff, 0x80, 0x00, 0x00, 0x3f, 0xff, 0xe0, 0xff, 0xff, 0x80, 0x00,
    0x00, 0x3f, 0xff, 0xe0,
0xff, 0xff, 0x80, 0x00, 0x00, 0x1f, 0xff, 0xe0, 0xff, 0xf8, 0x00, 0x00,
    0x00, 0x1f, 0xff, 0xe0,
0xff, 0xe0, 0x00, 0x00, 0x00, 0x07, 0xff, 0xe0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x00, 0xff, 0xe0,
0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xe0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x00, 0x7f, 0xe0,
0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xe0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x0f, 0xe0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xe0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xe0,
```

```
0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0xfc, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x03, 0xe0,
0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0xfc, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x03, 0xe0,
0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0xfc, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x03, 0xe0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0xfe, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x03, 0xe0,
0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xe0, 0xff, 0x80, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xe0,
0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xe0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x00, 0x2f, 0xe0,
0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xe0, 0xff, 0xf0, 0x00, 0x00,
    0x00, 0x00, 0x7f, 0xe0,
0xff, 0xfe, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xe7, 0xfd, 0xff, 0x3f, 0xff, 0xe0, 0xff, 0xff, 0xe7, 0xf9,
    0xff, 0x3f, 0xff, 0xe0,
0xff, 0xff, 0xe1, 0xf8, 0xfe, 0x1f, 0xff, 0xe0, 0xff, 0xff, 0xc1, 0xf8,
    0xfe, 0x1f, 0xff, 0xe0,
0xff, 0xff, 0x87, 0xf1, 0xfc, 0x3f, 0xff, 0xe0, 0xff, 0xff, 0x07, 0xe1,
    0xf0, 0x3f, 0xff, 0xe0,
0xff, 0xff, 0x0f, 0xe3, 0xe0, 0x7f, 0xff, 0xe0, 0xff, 0xfe, 0x0f, 0xc3,
    0xe0, 0x7f, 0xff, 0xe0,
0xff, 0xfc, 0x1f, 0xc7, 0xc0, 0xff, 0xff, 0xe0, 0xff, 0xfe, 0x3f, 0xc7,
    0xe1, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0
};
const unsigned char sleet [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xcf,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xfc, 0x00, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xfc, 0x00,
    0xbf, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xfc, 0x00, 0x3f, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xf8, 0x00,
    0x0f, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xf0, 0x00, 0x07, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xf0, 0x00,
    0x07, 0xff, 0xff, 0xf0,
```

```
0xff, 0xff, 0xc0, 0x00, 0x07, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xc0, 0x00,
    0x07, 0xff, 0xff, 0xf0,
0xff, 0xff, 0x80, 0x00, 0x07, 0xff, 0xff, 0xf0, 0xff, 0xfc, 0x00, 0x00,
    0x07, 0xff, 0xff, 0xf0,
0xff, 0xf8, 0x00, 0x00, 0x07, 0xff, 0xff, 0xf0, 0xff, 0xf8, 0x00, 0x00,
    0x00, 0x3f, 0xff, 0xf0,
0xff, 0xd0, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xf0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x0f, 0xff, 0xf0,
0xff, 0xc0, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xf0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x0f, 0xff, 0xf0,
0xff, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xf0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x03, 0xff, 0xf0,
0xff, 0x00, 0x70, 0x00, 0x00, 0x03, 0xff, 0xf0, 0xff, 0x00, 0x70, 0x00,
    0x00, 0x03, 0xff, 0xf0,
0xff, 0x03, 0xfc, 0x00, 0x00, 0x03, 0xff, 0xf0, 0xff, 0x47, 0x0c, 0x00,
    0x00, 0x0f, 0xff, 0xf0,
0xff, 0xc4, 0x04, 0x00, 0x00, 0x0f, 0xff, 0xf0, 0xff, 0xf4, 0x0c, 0x00,
    0x00, 0x0f, 0xff, 0xf0,
0xff, 0xf4, 0x7c, 0x40, 0x00, 0x0f, 0xff, 0xf0, 0xff, 0xfc, 0x7f, 0xff,
    0xff, 0x9f, 0xff, 0xf0,
0xff, 0xfc, 0x00, 0x00, 0x00, 0xdf, 0xff, 0xf0, 0xff, 0xfc, 0x00, 0x00,
    0x00, 0xff, 0xff, 0xf0,
0xff, 0xff, 0x00, 0x00, 0x00, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xfd, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xfc, 0x00, 0x00, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xfc, 0x00,
    0x00, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xfc, 0x00, 0x00, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xfc, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xfc, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xfc, 0x1f,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xfc, 0x1f, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xfc, 0x1f,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xfe, 0x1f, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0
};
const unsigned char snow [] PROGMEM = {
0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xf0,
    0x7f, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xe0, 0x7f, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xe0,
    0x3f, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xf0, 0x7f, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xf0,
    0x7f, 0xff, 0xff, 0x80,
```

```
0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xb0,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xe0, 0xff, 0x00, 0x07, 0xf0, 0x7f, 0x80, 0xff, 0xc0, 0xf3, 0x00,
    0x0f, 0xf0, 0x7f, 0x80,
0xff, 0xc0, 0xf3, 0x00, 0x0c, 0xf0, 0x3f, 0x80, 0xff, 0xc0, 0xe3, 0xc0,
    0x1c, 0xf0, 0x3f, 0x80,
0xff, 0xe0, 0x03, 0xe0, 0x3c, 0x60, 0x3f, 0x80, 0xff, 0xf0, 0x03, 0xf0,
    0x7c, 0x00, 0x3f, 0x80,
0xff, 0xfe, 0x03, 0xf0, 0xfc, 0x03, 0xff, 0x80, 0xff, 0xff, 0x03, 0xf0,
    0xfc, 0x07, 0xff, 0x80,
0xff, 0xfe, 0x01, 0xf0, 0xfc, 0x07, 0xff, 0x80, 0xff, 0xfc, 0x00, 0xf0,
    0xfc, 0x03, 0xff, 0x80,
0xff, 0xf0, 0x00, 0xf0, 0xf8, 0x00, 0xff, 0x80, 0xff, 0xf0, 0x00, 0x70,
    0xf0, 0x00, 0xff, 0x80,
0xff, 0xff, 0xf0, 0x30, 0xe0, 0x1f, 0xff, 0x80, 0xff, 0xef, 0xf8, 0x10,
    0xe0, 0xff, 0xff, 0x80,
0xff, 0xc3, 0xec, 0x00, 0x01, 0xfe, 0x7f, 0x80, 0xff, 0xe3, 0xfe, 0x00,
    0x03, 0xfc, 0x7f, 0x80,
0xff, 0xe1, 0xff, 0x00, 0x03, 0xfc, 0x7d, 0x80, 0xf8, 0xe0, 0x7f, 0x00,
    0x07, 0xf0, 0xff, 0x80,
0xf0, 0x00, 0x00, 0x00, 0x0f, 0xf0, 0xf0, 0x80, 0xf0, 0x00, 0x00, 0x00,
    0x03, 0xc0, 0x60, 0x80,
0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xf0, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x80,
0xf0, 0xe0, 0x5e, 0x00, 0x00, 0x00, 0x00, 0x80, 0xf0, 0xe1, 0xfe, 0x00,
    0x00, 0x00, 0x21, 0x80,
0xff, 0xe3, 0xfc, 0x00, 0x0f, 0xf0, 0x77, 0x80, 0xff, 0xc3, 0xfc, 0x00,
    0x0f, 0xf8, 0x7f, 0x80,
0xff, 0xe7, 0xf8, 0x00, 0x07, 0xfc, 0x7f, 0x80, 0xff, 0xff, 0xf0, 0x30,
    0x03, 0xfc, 0x7f, 0x80,
0xff, 0xff, 0xf0, 0x70, 0xc3, 0xff, 0x7f, 0x80, 0xff, 0xf1, 0x80, 0xf0,
    0xc1, 0xff, 0xff, 0x80,
0xff, 0xf0, 0x01, 0xf0, 0xc0, 0x00, 0xff, 0x80, 0xff, 0xf0, 0x03, 0xe0,
    0xe0, 0x01, 0xff, 0x80,
0xff, 0xfe, 0x07, 0xe0, 0xf0, 0x03, 0xff, 0x80, 0xff, 0xfc, 0x07, 0xe0,
    0xf8, 0x07, 0xff, 0x80,
0xff, 0xfc, 0x07, 0xf0, 0xfc, 0x0f, 0xff, 0x80, 0xff, 0xf8, 0x03, 0xe0,
    0x7c, 0x0f, 0xff, 0x80,
0xff, 0xe0, 0x43, 0xe0, 0xfc, 0x00, 0xff, 0x80, 0xff, 0xc0, 0xe3, 0xc0,
    0xfc, 0x40, 0x3f, 0x80,
0xff, 0xc0, 0xe3, 0x00, 0x3c, 0x60, 0x7f, 0x80, 0xff, 0xc0, 0xf6, 0x00,
    0x1c, 0xf0, 0x3f, 0x80,
0xff, 0xe0, 0xfc, 0x21, 0x0d, 0xe0, 0x7f, 0x80, 0xff, 0xe0, 0xfc, 0x33,
    0xcf, 0xf0, 0xff, 0x80,
0xff, 0xff, 0xfb, 0xf3, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xe1,
    0xfd, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xc0,
    0xff, 0xff, 0xff, 0x80,
```

```
0xff, 0xff, 0xff, 0xc0, 0x7f, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xc0,
    0xff, 0xff, 0xff, 0x80,
0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xff, 0x80, 0xff, 0xff, 0xff, 0xf9,
    0xff, 0xff, 0xff, 0x80
};

const unsigned char unknown [] PROGMEM = {
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xc0, 0x1f, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0x00,
    0x0f, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xfe, 0x00, 0x03, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xf8, 0x00,
    0x01, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xf0, 0x00, 0x00, 0x7f, 0xff, 0xe0, 0xff, 0xff, 0xe0, 0x00,
    0x00, 0x7f, 0xff, 0xe0,
0xff, 0xff, 0xe0, 0x00, 0x00, 0x3f, 0xff, 0xe0, 0xff, 0xff, 0xe0, 0x00,
    0x00, 0x3f, 0xff, 0xe0,
0xff, 0xff, 0xc0, 0x00, 0x00, 0x3f, 0xff, 0xe0, 0xff, 0xff, 0xc0, 0x00,
    0x00, 0x3f, 0xff, 0xe0,
0xff, 0xff, 0xc0, 0x00, 0x00, 0x3f, 0xff, 0xe0, 0xff, 0xfe, 0x00, 0x00,
    0x00, 0x1f, 0xff, 0xe0,
0xff, 0xf8, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xe0, 0xff, 0xf0, 0x00, 0x00,
    0x00, 0x01, 0xff, 0xe0,
0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xe0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x00, 0x3f, 0xe0,
0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xe0, 0xff, 0x80, 0x00, 0x00,
    0x00, 0x00, 0x0f, 0xe0,
0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xe0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xe0,
0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xe0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xe0,
0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xe0, 0xff, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xe0,
0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x07, 0xe0, 0xff, 0x80, 0x00, 0x00,
    0x00, 0x00, 0x07, 0xe0,
0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xe0, 0xff, 0xc0, 0x00, 0x00,
    0x00, 0x00, 0x0f, 0xe0,
0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xe0, 0xff, 0xf0, 0x00, 0x00,
    0x00, 0x00, 0x3f, 0xe0,
0xff, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xe0, 0xff, 0xff, 0x80, 0x00,
    0x00, 0x0f, 0xff, 0xe0,
0xff, 0xff, 0x80, 0x00, 0x00, 0x0f, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xf1, 0xfc,
    0x7f, 0x1f, 0xff, 0xe0,
```

```
0xff, 0xff, 0xe1, 0xf8, 0x7e, 0x1f, 0xff, 0xe0, 0xff, 0xff, 0xe3, 0xf8,
    0x7e, 0x1f, 0xff, 0xe0,
0xff, 0xff, 0xc7, 0xf0, 0xfc, 0x3f, 0xff, 0xe0, 0xff, 0xff, 0x87, 0xe1,
    0xf8, 0x7f, 0xff, 0xe0,
0xff, 0xff, 0x8f, 0xe3, 0xf0, 0xff, 0xff, 0xe0, 0xff, 0xff, 0x0f, 0xc3,
    0xf0, 0xff, 0xff, 0xe0,
0xff, 0xff, 0x1f, 0xc7, 0xf1, 0xff, 0xff, 0xe0, 0xff, 0xff, 0x1f, 0xc7,
    0xf1, 0xff, 0xff, 0xe0,
0xff, 0xff, 0x3f, 0xcf, 0xf3, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xe0
};

// Weather icon list.
//.............................................0......1.............2.....
const unsigned char* weather_icon [] PROGMEM = {sunny, partly_cloudy,
    sunny_cloudy_rain, cloudy, drizzle, mist, rain, sleet, snow, unknown};
int weather_icon_index = 9;

// Parameters for 1 hour forecast.
// RGB - 'R' last 1 hour values.
int R_array [] =
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
// Rain last 1 hour values.
int r_array [] =
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
// Temperature last 1 hour values.
int T_array [] =
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int forecast_array_index = -1; // array update index
int divider = 0; // divider for taking average
int R_average_value = 0;
int r_average_value = 0;
int T_average_value = 0;
long previousMillis = 0; // last time the sensor value was updated

// Notify callback for BLE connection
static void notifyCallback(
BLERemoteCharacteristic* pBLERemoteCharacteristic,
uint8_t* pData,
size_t length,
bool isNotify) {
   // do nothing, as values are read explicitly
}
```

```
class WeatherClientCallback : public BLEClientCallbacks {
   void onConnect(BLEClient* pclient) {
   }

   void onDisconnect(BLEClient* pclient) {
      connected = false;
      Serial.println("onDisconnect");
   }
};

// BLE connect to server, code adapted from BLE client example of
// ESP32 BLE Arduino BLE_client.
bool connectToServer() {
   Serial.print("Forming a connection to remote station");
   Serial.println(device->getAddress().toString().c_str());

   BLEClient* pClient = BLEDevice::createClient();
   Serial.println(" - Created client");

   pClient->setClientCallbacks(new WeatherClientCallback());

   // Connect to the remove BLE Server.
   pClient->connect(device);
   Serial.println(" - Connected to remote station");

   // Obtain a reference to the weather service in the remote BLE server.
   BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
   if (pRemoteService == nullptr) {
      Serial.print("Failed to find weather service UUID: ");
      Serial.println(serviceUUID.toString().c_str());
      pClient->disconnect();
      return false;
   }
   Serial.println(" - Found weather service");


   // Obtain a reference to the characteristic of weather service of the
       remote BLE server.
   pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
   if (pRemoteCharacteristic == nullptr) {
      Serial.print("Failed to find characteristic UUID: ");
      Serial.println(charUUID.toString().c_str());
      pClient->disconnect();
      return false;
   }
   Serial.println(" - Found characteristic");

   // Read the value of the characteristic.
   if(pRemoteCharacteristic->canRead()) {
```

```
      std::string value = pRemoteCharacteristic->readValue();
      Serial.print("The characteristic value was: ");
      Serial.println(value.c_str());
   }

   if(pRemoteCharacteristic->canNotify())
      pRemoteCharacteristic->registerForNotify(notifyCallback);

   connected = true;
   return true;
}

class WeatherAdvertisedDeviceCallbacks: public
    BLEAdvertisedDeviceCallbacks {
   void onResult(BLEAdvertisedDevice advertisedDevice) {
   Serial.print("BLE Advertised Device found: ");
   Serial.println(advertisedDevice.toString().c_str());

   if (advertisedDevice.haveServiceUUID() &&
       advertisedDevice.isAdvertisingService(serviceUUID)) {
      BLEDevice::getScan()->stop();
      device = new BLEAdvertisedDevice(advertisedDevice);
      doConnect = true;
      doScan = true;
      }
   }
};

// Helper function to convert from string to integer vector.
std::vector<int> convertToIntArray(std::string input)
{
   std::replace(input.begin(), input.end(), ',', ' ');
   std::istringstream stringReader{ input };
   std::vector<int> result;

   int number;
   while (stringReader >> number)
   {
      result.push_back(number);
   }

   return result;
}

// Forecast weather based on sensor values.
String forecast(int T, int t, int R, int r) {
   // decide based on temperature.
   // rain case:
   if(T > TEMP_MIN) {
```

```
if(t >= HOUR_SIX && t < HOUR_TWELVE) {
if(R >= RED_COLOUR_MAX){
if(r >= RAIN_AREA) {
weather_icon_index = 2;
return "SUNNY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 2;
return "SUNNY with RAIN";
} else {
weather_icon_index = 0;
return "SUNNY";
}

} else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
if(r >= RAIN_AREA) {
weather_icon_index = 2;
return "PARTIALLY CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 2;
return "PARTIALLY CLOUDY with RAIN";
} else {
weather_icon_index = 1;
return "PARTIALLY CLOUDY";
}
} else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
if(r >= RAIN_AREA) {
weather_icon_index = 6;
return "MOSTLY CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 6;
return "MOSTLY CLOUDY with RAIN";
} else {
weather_icon_index = 3;
return "MOSTLY CLOUDY";
}
} else {
if(r >= RAIN_AREA) {
weather_icon_index = 6;
return "CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 6;
return "CLOUDY with RAIN";
} else {
weather_icon_index = 3;
return "CLOUDY";
}
}
} else if(t >= HOUR_TWELVE && t < HOUR_EIGHTEEN){
if(R >= RED_COLOUR_MAX){
```

```
if(r >= RAIN_AREA) {
weather_icon_index = 2;
return "SUNNY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 2;
return "SUNNY with RAIN";
} else {
weather_icon_index = 0;
return "SUNNY";
}
} else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
if(r >= RAIN_AREA) {
weather_icon_index = 2;
return "PARTIALLY CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 2;
return "PARTIALLY CLOUDY with RAIN";
} else {
weather_icon_index = 1;
return "PARTIALLY CLOUDY";
}
} else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
if(r >= RAIN_AREA) {
weather_icon_index = 6;
return "MOSTLY CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 6;
return "MOSTLY CLOUDY with RAIN";
} else {
weather_icon_index = 3;
return "MOSTLY CLOUDY";
}
} else {
if(r >= RAIN_AREA) {
weather_icon_index = 6;
return "CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 6;
return "CLOUDY with RAIN";
} else {
weather_icon_index = 3;
return "CLOUDY";
}
}
} else if(t >= HOUR_EIGHTEEN && t < HOUR_TWENTYFOUR){
if(R >= RED_COLOUR_MAX){
if(r >= RAIN_AREA) {
weather_icon_index = 2;
return "SUNNY with HEAVY RAIN";
```

```
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 2;
return "SUNNY with RAIN";
} else {
weather_icon_index = 0;
return "SUNNY";
}
} else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
if(r >= RAIN_AREA) {
weather_icon_index = 2;
return "PARTIALLY CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 2;
return "PARTIALLY CLOUDY with RAIN";
} else {
weather_icon_index = 1;
return "PARTIALLY CLOUDY";
}
} else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
if(r >= RAIN_AREA) {
weather_icon_index = 6;
return "MOSTLY CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 6;
return "MOSTLY CLOUDY with RAIN";
} else {
weather_icon_index = 3;
return "MOSTLY CLOUDY";
}
} else {
if(r >= RAIN_AREA) {
weather_icon_index = 6;
return "CLOUDY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 6;
return "CLOUDY with RAIN";
} else {
weather_icon_index = 3;
return "CLOUDY";
}
}
} else {
if(R >= RED_COLOUR_MAX){
if(r >= RAIN_AREA) {
weather_icon_index = 2;
return "SUNNY with HEAVY RAIN";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 2;
return "SUNNY with RAIN";
```

```
    } else {
    weather_icon_index = 0;
    return "SUNNY";
    }
    } else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
    if(r >= RAIN_AREA) {
    weather_icon_index = 2;
    return "PARTIALLY CLOUDY with HEAVY RAIN";
    } else if(r < RAIN_AREA && r > 20) {
    weather_icon_index = 2;
    return "PARTIALLY CLOUDY with RAIN";
    } else {
    weather_icon_index = 1;
    return "PARTIALLY CLOUDY";
    }
    } else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
    if(r >= RAIN_AREA) {
    weather_icon_index = 6;
    return "MOSTLY CLOUDY with HEAVY RAIN";
    } else if(r < RAIN_AREA && r > 20) {
    weather_icon_index = 6;
    return "MOSTLY CLOUDY with RAIN";
    } else {
    weather_icon_index = 3;
    return "MOSTLY CLOUDY";
    }
    } else {
    if(r >= RAIN_AREA) {
    weather_icon_index = 6;
    return "CLOUDY with HEAVY RAIN";
    } else if(r < RAIN_AREA && r > 20) {
    weather_icon_index = 6;
    return "CLOUDY with RAIN";
    } else {
    weather_icon_index = 3;
    return "CLOUDY";
    }
    }
    }
// snow case:
} else {
    if(t >= HOUR_SIX && t < HOUR_TWELVE) {
    if(R >= RED_COLOUR_MAX){
    if(r >= RAIN_AREA) {
    weather_icon_index = 5;
    return "SUNNY with HEAVY SNOW";
    } else if(r < RAIN_AREA && r > 20) {
    weather_icon_index = 5;
    return "SUNNY with SNOW";
```

```
} else {
weather_icon_index = 0;
return "SUNNY";
}

} else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
if(r >= RAIN_AREA) {
weather_icon_index = 5;
return "PARTIALLY CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 5;
return "PARTIALLY CLOUDY with SNOW";
} else {
weather_icon_index = 1;
return "PARTIALLY CLOUDY";
}
} else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
if(r >= RAIN_AREA) {
weather_icon_index = 8;
return "MOSTLY CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 8;
return "MOSTLY CLOUDY with SNOW";
} else {
weather_icon_index = 3;
return "MOSTLY CLOUDY";
}
} else {
if(r >= RAIN_AREA) {
weather_icon_index = 8;
return "CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 8;
return "CLOUDY with SNOW";
} else {
weather_icon_index = 3;
return "CLOUDY";
}
}
} else if(t >= HOUR_TWELVE && t < HOUR_EIGHTEEN){
if(R >= RED_COLOUR_MAX){
if(r >= RAIN_AREA) {
weather_icon_index = 5;
return "SUNNY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 5;
return "SUNNY with SNOW";
} else {
weather_icon_index = 0;
```

```
return "SUNNY";
}
} else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
if(r >= RAIN_AREA) {
weather_icon_index = 5;
return "PARTIALLY CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 5;
return "PARTIALLY CLOUDY with SNOW";
} else {
weather_icon_index = 1;
return "PARTIALLY CLOUDY";
}
} else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
if(r >= RAIN_AREA) {
weather_icon_index = 8;
return "MOSTLY CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 8;
return "MOSTLY CLOUDY with SNOW";
} else {
weather_icon_index = 3;
return "MOSTLY CLOUDY";
}
} else {
if(r >= RAIN_AREA) {
weather_icon_index = 8;
return "CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 8;
return "CLOUDY with SNOW";
} else {
weather_icon_index = 3;
return "CLOUDY";
}
}
} else if(t >= HOUR_EIGHTEEN && t < HOUR_TWENTYFOUR){
if(R >= RED_COLOUR_MAX){
if(r >= RAIN_AREA) {
weather_icon_index = 5;
return "SUNNY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 5;
return "SUNNY with SNOW";
} else {
weather_icon_index = 0;
return "SUNNY";
}
} else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
```

```
if(r >= RAIN_AREA) {
weather_icon_index = 5;
return "PARTIALLY CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 5;
return "PARTIALLY CLOUDY with SNOW";
} else {
weather_icon_index = 1;
return "PARTIALLY CLOUDY";
}
} else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
if(r >= RAIN_AREA) {
weather_icon_index = 8;
return "MOSTLY CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 8;
return "MOSTLY CLOUDY with SNOW";
} else {
weather_icon_index = 3;
return "MOSTLY CLOUDY";
}
} else {
if(r >= RAIN_AREA) {
weather_icon_index = 8;
return "CLOUDY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 8;
return "CLOUDY with SNOW";
} else {
weather_icon_index = 3;
return "CLOUDY";
}
}
} else {
if(R >= RED_COLOUR_MAX){
if(r >= RAIN_AREA) {
weather_icon_index = 5;
return "SUNNY with HEAVY SNOW";
} else if(r < RAIN_AREA && r > 20) {
weather_icon_index = 5;
return "SUNNY with SNOW";
} else {
weather_icon_index = 0;
return "SUNNY";
}
} else if(R > RED_COLOUR_MED && R < RED_COLOUR_MAX) {
if(r >= RAIN_AREA) {
weather_icon_index = 5;
return "PARTIALLY CLOUDY with HEAVY SNOW";
```

```
      } else if(r < RAIN_AREA && r > 20) {
      weather_icon_index = 5;
      return "PARTIALLY CLOUDY with SNOW";
      } else {
      weather_icon_index = 1;
      return "PARTIALLY CLOUDY";
      }
      } else if(R > RED_COLOUR_MIN && R < RED_COLOUR_MED) {
      if(r >= RAIN_AREA) {
      weather_icon_index = 8;
      return "MOSTLY CLOUDY with HEAVY SNOW";
      } else if(r < RAIN_AREA && r > 20) {
      weather_icon_index = 8;
      return "MOSTLY CLOUDY with SNOW";
      } else {
      weather_icon_index = 3;
      return "MOSTLY CLOUDY";
      }
      } else {
      if(r >= RAIN_AREA) {
      weather_icon_index = 8;
      return "CLOUDY with HEAVY SNOW";
      } else if(r < RAIN_AREA && r > 20) {
      weather_icon_index = 8;
      return "CLOUDY with SNOW";
      } else {
      weather_icon_index = 3;
      return "CLOUDY";
      }
      }
      }
   }
}

// Update forecast arrays and parameters.
void update_forecast_values(int T, int R, int r) {
   if(divider < 60) {
   divider = divider + 1;
      }

   if(forecast_array_index == 59) {
      forecast_array_index = 0;
   } else {
         forecast_array_index = forecast_array_index + 1;
   }

   if(R > RED_COLOUR_MAX){
      R = RED_COLOUR_MAX + 10; // Normalize for very high R, add tolerance
   }
```

```
   R_array[forecast_array_index] = R;

   if(r > RAIN_AREA){
      r = RAIN_AREA + 10; // Normalize for very high r, add tolerance
   }
   r_array[forecast_array_index] = r;

   if(T > 30){
      T = 30; // Normalize for very high T
   }
   T_array[forecast_array_index] = T;
}

// Find R value average.
int R_average() {
   int sum = 0;
   int average = 0;
   for(int i = 0; i < divider; i++) {
      sum = sum + R_array[i];
   }

   average = (int) (sum / divider);
   return average;
}

// Find r value average.
int r_average() {
   int sum = 0;
   int average = 0;
   for(int i = 0; i < divider; i++) {
   s  um = sum + r_array[i];
   }

   average = (int) (sum / divider);
   return average;
}

// Find T value average.
int T_average() {
   int sum = 0;
   int average = 0;
   for(int i = 0; i < divider; i++) {
      sum = sum + T_array[i];
   }

   average = (int) (sum / divider);
   return average;
}
```

```
// The sensorData vector with index = 0 contains Read color value.
int get_colour(){
   return sensorData[0];
}

// The sensorData vector with index = 1 contains Temperature value.
int get_temperature(){
   return sensorData[1];
}

// The sensorData vector with index = 2 contains Humidity value.
int get_humidity(){
   return sensorData[2];
}


// The sensorData vector with index = 3 contains Pressure value.
int get_pressure(){
   return sensorData[3];
}

// The sensorData vector with index = 4 contains Rain value.
int get_rain(){
   return sensorData[4];
}

// The sensorData vector with index = 5 contains wind speed value.
int get_wind(){
   return sensorData[5];
}

// The sensorData vector with index = 6 contains wind direction value.
int get_wind_direction(){
   return sensorData[6];
}

tm getLocalTime()
{
   struct tm timeinfo;
   if(!getLocalTime(&timeinfo)){
      Serial.println("Failed to obtain time");
      return tm();
   }
   Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");

   return timeinfo;
}
```

```
// Create an OLED display object connected to I2C.
Adafruit_SSD1306 oled(OLED_SCREEN_WIDTH, OLED_SCREEN_HEIGHT, &Wire, -1);

// Initialize base station capabilities.
void setup() {
   Serial.begin(9600);

   BLEDevice::init("");

   // Obtain a scanner and, do active scan and run for 5 seconds.
   BLEScan* pBLEScan = BLEDevice::getScan();
   pBLEScan->setAdvertisedDeviceCallbacks(new
       WeatherAdvertisedDeviceCallbacks());
   pBLEScan->setWindow(449);
   pBLEScan->setInterval(1349);
   pBLEScan->setActiveScan(true);
   pBLEScan->start(5, false);

   // Connect to WiFi.
   Serial.printf("Connecting to %s ", ssid);
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
   }
   Serial.println("WIFI CONNECTED");

   // Init and get the time
   configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
   getLocalTime();

   // Initialize OLED display with I2C address 0x3C.
   if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
      Serial.println(F("failed to start SSD1306 OLED"));
      while (1);
   }

   delay(2000);              // wait two seconds for initializing
   oled.clearDisplay();      // clear display
   oled.setTextSize(1);      // set text size
   oled.setTextColor(WHITE); // set text color
   oled.setCursor(0, 10);    // set position to display
   oled.display();           // display on OLED

   // Disconnect WiFi as it's no longer needed
   WiFi.disconnect(true);
   WiFi.mode(WIFI_OFF);
}
```

```
void loop() {
  // Do the BLE connection and sensor data receive first.
  if (doConnect == true) {
    if (connectToServer()) {
      Serial.println("We are now connected to the remote BLE Server.");
    } else {
      Serial.println("We have failed to connect to the remote server.");
    }
    doConnect = false;
  }

  // Once connected to the remote BLE Server, read sensor data.
  if (connected) {
    // Read the value of the characteristic, the sensor data.
    if(pRemoteCharacteristic->canRead()) {
      std::string value = pRemoteCharacteristic->readValue();
      sensorData = convertToIntArray(value);
      Serial.print("The sensor value was: ");
      Serial.println(value.c_str());
    }
  } else if (doScan){
    BLEDevice::getScan()->start(0); // scan after disconnect
  }

  oled.clearDisplay();
  oled.setCursor(0, 0);

  struct tm lTime = getLocalTime();
  String date_time = String(lTime.tm_mday)+"/"+String(lTime.tm_mon)+" "+
  String(lTime.tm_hour)+":"+String(lTime.tm_min)+":"+
  String(lTime.tm_sec);

  // Display date and time in OLED
  oled.println(date_time);

  // Get individual sensor data.
  int R = get_colour();
  int T = get_temperature();
  int H = get_humidity();
  int P = get_pressure();
  int r = get_rain();
  int w = get_wind();
  int d = get_wind_direction();

  int t = lTime.tm_hour;
  if(t >= HOUR_SIX && t < HOUR_EIGHTEEN){
  oled.setCursor(85, 0);
  oled.println("Daytime");
  }else {
```

```
oled.setCursor(85, 0);
oled.println("Night");
}

oled.setCursor(0, 10);
String R_str = "RGB - R  : "+String(R);
oled.println(R_str);
Serial.println(R_str);
String T_str = "Temperature: "+String(T);
oled.println(T_str + " " + (char)247 + "C");
Serial.println(T_str + "°C");
String H_str = "Humidity : "+String(H);
oled.println(H_str + " %");
Serial.println(H_str + "%");
String P_str = "Pressure : "+String(P);
oled.println(P_str + " kpl");
Serial.println(P_str + "kpl");
String r_str = "Rain     : "+String(r);
oled.println(r_str +" %");
Serial.println(r_str +"%");
String w_str = "Wind     : "+String(w);
oled.println(w_str +" kmh");
Serial.println(w_str +" kmh");

oled.display();
delay(5000);
oled.clearDisplay();
oled.setCursor(0, 0);

// Forecast the current weather.
String curr_forecast = forecast(T, t, R, r);
oled.println(curr_forecast);
Serial.print("current weather : ");
Serial.println(curr_forecast);

oled.setCursor(4, 14);
bool iswind = false;
if(w >= WIND_MAX) {
   iswind = true;
   oled.println("Heavy Wind ->");
   Serial.println("Heavy Wind ->");
}else if(w > WIND_MIN && w < WIND_MAX ){
   iswind = true;
   oled.println("Windy ->");
   Serial.println("Windy ->");
} else {
   iswind = false;
   oled.println("No Wind ->");
   Serial.println("No Wind ->");
```

```
}

oled.setCursor(4, 24);
if(iswind) {
   if(d >= 900 && d < 100) {
      oled.println("South");
      Serial.println("South");
   } else if(d >= 800 && d < 900) {
      oled.println("SouthEast");
      Serial.println("SouthEast");
   } else if(d >= 650 && d < 800) {
      oled.println("East");
      Serial.println("East");
   } else if(d >= 550 && d < 650) {
      oled.println("NorthEast");
      Serial.println("NorthEast");
   } else if(d >= 450 && d < 550) {
      oled.println("North");
      Serial.println("North");
   } else if(d >= 350 && d < 450) {
      oled.println("NorthWest");
      Serial.println("NorthWest");
   }else if(d >= 200 && d < 350){
      oled.println("West");
      Serial.println("West");
   } else {
      oled.println("SouthWest");
      Serial.println("SouthWest");
   }
}

// Draw weather icon.
int icon_width = 57;
int icon_height = 54;
oled.drawBitmap(70, 7, weather_icon[weather_icon_index], icon_width,
    icon_height, BLACK, WHITE);

// Forecast next 1 hour weather.
if(previousMillis == 0) {
   previousMillis = 1;
   update_forecast_values(T, R, r);
   R_average_value = R_average();
   r_average_value = r_average();
   T_average_value = T_average();
}
long currentMillis = millis();
// if 60000ms/1Min have passed, check the sensor level:
if (currentMillis - previousMillis >= 60000) {
   previousMillis = currentMillis;
```

```
      update_forecast_values(T, R, r);
      R_average_value = R_average();
      r_average_value = r_average();
      T_average_value = T_average();
   }
   String one_hour_forecast = forecast(t+1, T_average_value,
       R_average_value, r_average_value);
   oled.setCursor(0, 40);
   oled.println("In 1 hour:");
   oled.setCursor(0, 52);
   oled.println(one_hour_forecast);
   Serial.print("In 1 hour: ");
   Serial.println(one_hour_forecast);

   oled.display();
   delay(5000);
}
```

7.3 Datasheet

- Arduino Nano 33 BLE Sense: https://docs.arduino.cc/static/2fbd3b03514cbd7a6ac6f4dc8234472e/ABX00031-datasheet.pdf
- ESP32: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Bibliography

[1] V. Masson-Delmotte, "Global warming of 1.5Â°c," *Intergovernmental Panel on Climate Change*, 2019.

[2] NASA, "Interactive global composite weather satellite images," *Wayback Machine*, 2008.

[3] NOAA, "Goes eastern us sector infrared image," *NOAA*, 2008.

[4] cactus.io. Arduino weather station project. [Online]. Available: http://cactus.io/projects/weather/arduino-weather-station

[5] www.instructables.com. Arduino wireless weather station. [Online]. Available: https://www.instructables.com/Arduino-Wireless-Weather-Station/

[6] G. Tanner. Arduino nano 33 ble sense overview. [Online]. Available: https://gilberttanner.com/blog/arduino-nano-33-ble-sense-overview/

[7] os.mbed.com. Mbed. [Online]. Available: https://os.mbed.com/

[8] en.wikipedia.org. Tensilica. [Online]. Available: https://en.wikipedia.org/wiki/Tensilica

[9] circuits4you.com. Esp32 devkit esp32-wroom gpio pinout. [Online]. Available: https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/

[10] www.aeq web.com. Arduino anemometer circuit. [Online]. Available: https://www.aeq-web.com/arduino-anemometer-wind-sensor/