```
/*
* Author: Michael Ali
* source for encoder code: https://garrysblog.com/2021/03/20/reliably-debouncing-rotary-encoders-with-arduino-and-esp32/
*/
#include <Arduino.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Servo.h>
#include <EEPROM.h>
#include <TEA5767Radio.h>

#include <hd44780.h> // main hd44780 header
#include <hd44780ioClass/hd44780_I2Cexp.h> // i2c expander i/o class header

hd44780_I2Cexp lcd(0x27); // declare lcd object: auto locate & auto config expander chip



TEA5767Radio radio = TEA5767Radio();
float station=90.9;
float oldStation=station;

const byte adcPin = A0;
const int analogInPin = A0; // Analog input pin that the potentiometer is attached to
int sensorValue = 512; // value read from the pot
long lastRead;

uint8_t radio_on = 0;
uint8_t weather_on = 0;
uint8_t switch_transition_hilo = 0;
uint8_t switch_transition_lohi = 0;
int old_sensorValue;
int i = 0;
int m = 0;
int failCounter=0;

// Rotary Encoder Inputs
#define encoderCLK 12 //D6
#define encoderDT 13 //D7
#define encoderSW 14 //D5

// Define rotary encoder pins
#define ENC_A 12
#define ENC_B 13
```

```
volatile int counter = 909;
int stationCounter = 10;
int servoAngle = 0;
int crntCLK = 0;
int prvsCLK;
uint8_t SWstate = 0;
uint8_t toggle = 0;

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using I2C
#define OLED_RESET -1 // Reset pin
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Replace with your SSID and password details
char ssid[] = "YOUR SSID";
char pass[] = "YOUR WIFI PASSWORD";

WiFiClient client;

// Open Weather Map API server name
const char server[] = "api.openweathermap.org";

// Replace the next line to match your city and 2 letter country code
//String nameOfCity = "REPLACE_WITH_YOUR_CITY,REPLACE_WITH_YOUR_COUNTRY_CODE";
// How your nameOfCity variable would look like for Lagos on Nigeria
String nameOfCity = "Baltimore,US";
String NameOfCity[6] = { "Brooklyn,NY,US", "Denver,US", "Beltsville,US", "Irvine,US", "Miami,US","Lebanon,NH,US" };
int VECTOR_LENGTH = 6;
byte cityNameByte;

// Replace the next line with your API Key
String apiKey = "YOUR OPENWEATHER API KEY";

String text;

int jsonend = 0;
boolean startJson = false;
int status = WL_IDLE_STATUS;

int rainLed = 2; // Indicates rain
int clearLed = 3; // Indicates clear sky or sunny
int snowLed = 4; // Indicates snow
int hailLed = 5; // Indicates hail

#define JSON_BUFF_DIMENSION 500

unsigned long lastConnectionTime = 10 * 60 * 1000; // last time you connected to the server, in milliseconds
```

```
const unsigned long postInterval = 10 * 60 * 1000; // posting interval of 10 minutes (10L * 1000L; 10 seconds delay for testing)
//unsigned long lastConnectionTime = 1 * 20 * 1000; // 20 seconds for testing last time you connected to the server, in milliseconds
//const unsigned long postInterval = 1 * 20 * 1000; // 20 seconds for testing posting interval of 10 minutes (10L * 1000L; 10 seconds delay for testing)

unsigned long lastDebounceTime = 0; // the last time the output pin was toggled
unsigned long debounceDelay = 300; // for the switch
unsigned long last_interrupt_time = 0;
#define DEBOUNCE 0 //debounce time delay in ms for the encoder

byte radioStationByte1;
byte radioStationByte2;
String radioStation = "";



ICACHE_RAM_ATTR void encoderInterrupt() {
if (millis() - lastDebounceTime > debounceDelay) {
lastDebounceTime = millis();
toggle = 1;
//Serial.println("interrupt: " + String(toggle));
}
}

ICACHE_RAM_ATTR void read_encoder() {
// Encoder interrupt routine for both pins. Updates counter
// if they are valid and have rotated a full indent

static uint8_t old_AB = 3; // Lookup table index
static int8_t encval = 0; // Encoder value
static const int8_t enc_states[] = { 0, -1, 1, 0, 1, 0, 0, -1, -1, 0, 0, 1, 0, 1, -1, 0 }; // Lookup table
static unsigned long lastInterruptTime = 0;
unsigned long interruptTime = millis();

old_AB <<= 2; // Remember previous state

if (digitalRead(ENC_A)) old_AB |= 0x02; // Add current state of pin A
if (digitalRead(ENC_B)) old_AB |= 0x01; // Add current state of pin B

encval += enc_states[(old_AB & 0x0f)];

// Update counter if encoder has rotated a full indent, that is at least 4 steps
if (encval > 3) { // Four steps forward
if (interruptTime - lastInterruptTime > 40) { // Greater than 40 milliseconds
counter++; // Increase by 1
} else if (interruptTime - lastInterruptTime > 20) { // Greater than 20 milliseconds
counter += 3; // Increase by 3
} else { // Faster than 20 milliseconds
counter += 10; // Increase by 10
```

```
}
encval = 0;
lastInterruptTime = millis(); // Remember time
} else if (encval < -3) { // Four steps backwards
if (interruptTime - lastInterruptTime > 40) { // Greater than 40 milliseconds
counter--; // Increase by 1
} else if (interruptTime - lastInterruptTime > 20) { // Greater than 20 milliseconds
counter -= 3; // Increase by 3
} else { // Faster than 20 milliseconds
counter -= 10; // Increase by 10
}
encval = 0;
lastInterruptTime = millis(); // Remember time
}
}

void initializeEncoder() {
// Set encoder switch
pinMode(encoderSW, INPUT);
// Set encoder pins
pinMode(ENC_A, INPUT_PULLUP);
pinMode(ENC_B, INPUT_PULLUP);
// Read the initial state of encoderCLK and assign it to prvsCLK variable
prvsCLK = digitalRead(encoderCLK);
attachInterrupt(digitalPinToInterrupt(encoderSW), encoderInterrupt, CHANGE);
attachInterrupt(digitalPinToInterrupt(ENC_A), read_encoder, CHANGE);
attachInterrupt(digitalPinToInterrupt(ENC_B), read_encoder, CHANGE);
}

void initializeDisplay() {
// initialize the OLED object
if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
Serial.println(F("SSD1306 allocation failed"));
for (;;)
; // Don't proceed, loop forever
}
// Clear the buffer.
display.clearDisplay();
// Display Text
display.setTextSize(2);
display.setTextColor(WHITE);
display.setCursor(0, 28);
display.println("Ready!");
display.display();
delay(2000);
display.clearDisplay();
/* */
}

void initializeLCD() {
lcd.init();
lcd.backlight();
lcd.setCursor(3, 0);
```

```
lcd.print("Setting up");
lcd.setCursor(1, 1);
lcd.print("Display Ready");
}

void WiFi_setup() {
lcd.setCursor(0,0);
lcd.print("Connecting to WF");
WiFi.disconnect();
WiFi.softAPdisconnect(true);
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, pass);
// Try forever
while (WiFi.status() != WL_CONNECTED) {
lcd.setCursor(0,0);
lcd.print("...Connecting ");
delay(1000);
}
lcd.setCursor(0,0);
lcd.print("Connected ");
printWiFiStatus();
}



void setup() {
Serial.begin(115200);
delay(1000);
initializeDisplay();
initializeLCD();
old_sensorValue = analogRead(A0);
sensorValue = old_sensorValue;
initializeEncoder();
lastConnectionTime=millis();
text.reserve(JSON_BUFF_DIMENSION);
EEPROM.begin(12);
// setup radio station
stationCounter = (EEPROM.read(2)<<8) + EEPROM.read(3);
counter = stationCounter;
station=float(stationCounter)/10;
radio.setFrequency(station);
//setup weather city
cityNameByte = EEPROM.read(4);
nameOfCity=NameOfCity[cityNameByte];
WiFi_setup();
if (sensorValue>512) {
radio_on=0;
weather_on=1;
}
else {
radio_on=1;
weather_on=0;
}
// counter=909;
```

```
// stationCounter=counter;
printToLCD();
makehttpRequest();
}

void updateCity() {
int prevCounter;
int j;
lcd.init();
lcd.backlight();
lcd.setCursor(3, 0);
lcd.print("Update City!");
lcd.setCursor(2, 1);
lcd.print(" ");
lcd.setCursor(2, 1);
lcd.print(nameOfCity);
makehttpRequest(); // in case the display is messed up due to the switch
for (j=0;j<VECTOR_LENGTH;j++) {
if (NameOfCity[j]==nameOfCity) counter=j;
}
// only call makehttpRequest when counter changes inside the loop
prevCounter=counter;
while (sensorValue>512) {
sensorValue=analogRead(A0);
delay(100);
//SWstate = digitalRead(encoderSW);
//i=SWstate;
if (counter > VECTOR_LENGTH - 1) counter = 0;
if (counter < 0) counter = VECTOR_LENGTH - 1;
//Serial.println("counter: "+String(counter));
if (counter != prevCounter) {
nameOfCity = NameOfCity[counter];
lcd.setCursor(2, 1);
lcd.print(" ");
lcd.setCursor(2, 1);
lcd.print(nameOfCity);
//Serial.println(nameOfCity);
makehttpRequest();
prevCounter=counter;
}
}
cityNameByte = counter;
lcd.setCursor(3, 0);
lcd.print(" ");
lcd.print(nameOfCity);
makehttpRequest();
}

void weather_loop() {
if (millis() - lastConnectionTime > postInterval) {
lastConnectionTime = millis();
makehttpRequest();
}
```

```
}

void determine_switch_transition() {
int delta;
switch_transition_lohi = 0;
switch_transition_hilo = 0;
sensorValue = analogRead(A0);
delta=sensorValue-old_sensorValue;
if (delta > 10) switch_transition_lohi = 1;
if (delta < -10) switch_transition_hilo = 1;
old_sensorValue = sensorValue;
//Serial.println(sensorValue);
}



void execute_toggle() {
detachInterrupt(digitalPinToInterrupt(encoderSW));
lcd.init();
lcd.setCursor(0, 0);

if (radio_on) {
lcd.print("storing station");
EEPROM.write(2, stationCounter>>8);
EEPROM.write(3, stationCounter & 0xFF);
EEPROM.commit();
}

if (weather_on) {
lcd.print("storing city");
cityNameByte = counter;
EEPROM.write(4, cityNameByte);
EEPROM.commit();
}
delay(1000); //give user time to see the display
attachInterrupt(digitalPinToInterrupt(encoderSW), encoderInterrupt, CHANGE);
}

void printToLCD() {
lcd.init();
lcd.backlight();
lcd.setCursor(0,1);
lcd.print("FM Station");
lcd.setCursor(11, 1);
lcd.print(station);
lcd.setCursor(0, 0);
lcd.print(NameOfCity[cityNameByte]);
}

void loop()
{
if (toggle) {
```

```
execute_toggle();
printToLCD();
makehttpRequest();
toggle=0;
}
determine_switch_transition();
if (sensorValue> 512) {
weather_on=1;
radio_on=0;
stationCounter=int(station*10); //preserve value
updateCity();
}

//handle returning from updateCity via switch
if ((sensorValue<512) && switch_transition_hilo) {
radio_on=1;
weather_on=0;
counter=stationCounter;
station=float(stationCounter)/10;
radio.setFrequency(station);
printToLCD();
}

//handle turning of encoder knob to change stations
station=float(counter)/10;
if (station!=oldStation && radio_on) {
oldStation=station;
printToLCD();
radio.setFrequency(station);
stationCounter=int(station*10);
}
// Serial.println(counter);
// lcd.init();
// lcd.setCursor(10, 1);
// lcd.print(sensorValue);
weather_loop();
// display.setCursor(0,55);
// display.println(String((millis()-lastConnectionTime)) + String(m));
// display.display();
}

void scrollText(String text2) {
display.clearDisplay();
display.setTextSize(1); // Draw 2X-scale text
display.setTextColor(SSD1306_WHITE);
display.setCursor(10, 0);
display.println(text2);
display.display(); // Show initial text
delay(100);
}

// print Wifi status
void printWiFiStatus() {
```

```
// print the SSID of the network you're attached to:
Serial.print("SSID: ");
Serial.println(WiFi.SSID());

// print your WiFi shield's IP address:
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);

// print the received signal strength:
long rssi = WiFi.RSSI();
Serial.print("signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
}

// to request data from OWM
void makehttpRequest() {
// close any connection before send a new request to allow client make connection to server
client.stop();

// if there's a successful connection:
if (client.connect(server, 80)) {
//Serial.println("connecting to OpenWeatherMap...");
// send the HTTP PUT request:
client.println("GET /data/2.5/forecast?q=" + nameOfCity + "&APPID=" + apiKey + "&mode=json&units=metric&cnt=2
HTTP/1.1");
// api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
client.println("Host: api.openweathermap.org");
client.println("User-Agent: ArduinoWiFi/1.1");
client.println("Connection: close");
client.println();

unsigned long timeout = millis();
while (client.available() == 0) {
if (millis() - timeout > 5000) {
Serial.println(">>> Client Timeout !");
client.stop();
return;
}
}

char c = 0;
while (client.available() || startJson) {
if (client.available() == 0) {
continue;
}
c = client.read();
// since json contains equal number of open and close curly brackets, this means we can determine when a json is
completely received by counting
// the open and close occurences,
//Serial.println(c);
```

```
//delay(7000);
if (c == '{') {
startJson = true; // set startJson true to indicate json message has started
jsonend++;
}
if (c == '}') {
jsonend--;
}
if (startJson == true) {
text += c;
}
// if jsonend = 0 then we have have received equal number of curly braces
if (jsonend == 0 && startJson == true) {
parseJson(text.c_str()); // parse c string text in parseJson function
text = ""; // clear text string for the next time
startJson = false; // set startJson to false to indicate that a new message has not yet started
}
}
} else {
// if no connection was made, let the user know for 1 second, but then just put the display back
// and try again
failCounter++;
Serial.println("connection failed " + String(failCounter) + " times");
// lcd.init();
// lcd.print("connection failed");
// delay(250);
//printToLCD();
lastConnectionTime=millis()-postInterval; //force weather_loop to call makehttpRequest with no delay
return;
}
}



//to parse json data recieved from OWM
void parseJson(const char *jsonString) {
//StaticJsonBuffer<4000> jsonBuffer;
const size_t bufferSize = 2 * JSON_ARRAY_SIZE(1) + JSON_ARRAY_SIZE(2) + 4 * JSON_OBJECT_SIZE(1) + 3 *
JSON_OBJECT_SIZE(2) + 3 * JSON_OBJECT_SIZE(4) + JSON_OBJECT_SIZE(5) + 2 * JSON_OBJECT_SIZE(7) + 2 *
JSON_OBJECT_SIZE(8) + 720;
DynamicJsonBuffer jsonBuffer(bufferSize);

// FIND FIELDS IN JSON TREE
JsonObject &root = jsonBuffer.parseObject(jsonString);
if (!root.success()) {
Serial.println("parseObject() failed");
return;
}

JsonArray &list = root["list"];
JsonObject &nowT = list[0];
JsonObject &later = list[1];
```

```
// including temperature and humidity for those who may wish to hack it in
String city = root["city"]["name"];

float tempNow = nowT["main"]["temp"];
float humidityNow = nowT["main"]["humidity"];
String weatherNow = nowT["weather"][0]["description"];

float tempLater = later["main"]["temp"];
float humidityLater = later["main"]["humidity"];
String weatherLater = later["weather"][0]["description"];

// checking for four main weather possibilities
diffDataAction(weatherNow, weatherLater, "clear");
diffDataAction(weatherNow, weatherLater, "rain");
diffDataAction(weatherNow, weatherLater, "snow");
diffDataAction(weatherNow, weatherLater, "hail");
/*
Serial.println(nameOfCity);
Serial.print("Temperature: ");
Serial.println(tempNow * 9 / 5 + 32);
Serial.print("Humidity: ");
Serial.println(humidityNow);
*/
m++;
// Clear the buffer.
display.clearDisplay();
// Display Text
display.setTextSize(2);
display.setTextColor(WHITE);
display.setCursor(0, 0);
//display.println(nameOfCity);
display.setCursor(0, 12);
//display.println("Outside Temperature:");
display.println(String(tempNow * 9 / 5 + 32) + " F");
display.setCursor(0, 32);
//display.println("Outside Humidity:");
display.setTextSize(1);
display.println(String(humidityNow) + " %");
display.setTextSize(1);
display.setCursor(0, 40);
//display.print("Weather: ");
display.println(weatherLater);
display.setCursor(0,55);
display.println("# API calls: "+ String(m));
//display.println(String(m));
//display.println(millis());
display.display();
//delay(2000);
display.clearDisplay();
}

//representing the data
void diffDataAction(String nowT, String later, String weatherType) {
```

```
int indexNow = nowT.indexOf(weatherType);
int indexLater = later.indexOf(weatherType);
// if weather type = rain, if the current weather does not contain the weather type and the later message does, send notification
if (weatherType == "rain") {
if (indexNow == -1 && indexLater != -1) {
Serial.println("Oh no! It is going to " + weatherType + " later! Predicted " + later);
}
}
// for snow
else if (weatherType == "snow") {
if (indexNow == -1 && indexLater != -1) {
Serial.println("Oh no! It is going to " + weatherType + " later! Predicted " + later);
}
}
// can't remember last time I saw hail anywhere but just in case
else if (weatherType == "hail") {
if (indexNow == -1 && indexLater != -1) {
Serial.println("Oh no! It is going to " + weatherType + " later! Predicted " + later);
}
}
// for clear sky, if the current weather does not contain the word clear and the later message does, send notification that it will be sunny later
else {
if (indexNow == -1 && indexLater != -1) {
Serial.println("It is going to be sunny later! Predicted " + later);
}
}
}
```