WETTERSTATION

SYT & ITP Projekt

27. MAI 2025

VERFASSER: MAIK GAO & MARCEL HOFMANN
2CHIT

Inhalt

Einführung

Im Rahmen dieses Projekts wurde eine einfache Wetterstation auf Basis des ESP32 entwickelt. Ziel war es, Umweltdaten wie Temperatur, Luftfeuchtigkeit und Magnetfeldstärke zu erfassen und über ein Webinterface darzustellen. Zusätzlich wurde das Projekt um zahlreiche Funktionen erweitert, um praktische Anwendungsmöglichkeiten zu demonstrieren und die Arbeit mit Mikrocontrollern praxisnah zu vertiefen.

Projektbeschreibung

Es wurde eine Wetterstation realisiert, die über Sensoren Temperatur, Luftfeuchtigkeit und mithilfe eines Hall-TTL-Sensors magnetische Felder misst. Die Daten werden auf einem Webinterface angezeigt, ergänzt durch einen einfachen Graphen zur historischen Darstellung sowie durch ein lokales Display. Weitere Features wie Sleep Mode, Status-LED-Steuerung und ein Wi-Fi Manager wurden implementiert.

Theorie

Für die Umsetzung dieses Projekts ist grundlegendes Wissen über Sensorik, Webserver-Programmierung auf Mikrocontrollern und energieeffiziente Datenverarbeitung erforderlich. Zum Einsatz kamen folgende Komponenten:

- **DHT11**: Misst Temperatur und Luftfeuchtigkeit. Der Sensor arbeitet digital und ist einfach mit dem ESP32 integrierbar.
- **Hall-Sensor (TTL-Ausgang)**: Erkennt Magnetfelder und ermöglicht die Erfassung von Bewegungen oder Kontaktzuständen.
- **ESP32 Mikrocontroller**: Verfügt über integriertes Wi-Fi, zahlreiche GPIO-Pins, PWM-Fähigkeiten und unterstützt den Sleep Mode zur Energieeinsparung.
- **OLED-Display**: Zeigt lokal aktuelle Sensordaten und Systemstatus an.
- **Status-LED (RGB)**: Visualisiert Zustände wie WLAN-Verbindung, Messvorgang oder Grenzwertüberschreitung.

Zusätzlich wurde ein **einfacher Graph** mittels Chart.js auf dem Webinterface implementiert, um historische Durchschnittswerte visuell darzustellen. Dabei wurden sinnvolle Intervalle zur Reduktion des Datenvolumens gewählt (z. B. Mittelwert alle 5 Minuten für die letzte Stunde).

Der **Wi-Fi Manager** erlaubt eine flexible Verbindung zu WLAN-Netzen oder das Aufspannen eines Access Points über das AsyncWebServer-Modul [11].

Die theoretische Grundlage zur Visualisierung historischer Daten beruht auf der Aggregation von Mittelwerten, um Speicherbedarf und Ladezeiten gering zu halten, wie in typischen IoT-Anwendungen üblich.

Arbeitsschritte

Zu Beginn des Projekts erhielt ich die Aufgabe, eine funktionierende Wetterstation mit einem Mikrocontroller aufzubauen. Ziel war es, sowohl Temperatur- als auch Luftfeuchtigkeitswerte zu messen, zusätzlich einen zweiten Sensor zu verwenden (in meinem Fall war das ein Hallsensor), die Werte auf einem OLED-Display darzustellen, sie über einen Webserver zugänglich zu machen und dort zusätzlich einen historischen Verlauf der Sensordaten grafisch darzustellen. Außerdem sollte über das Webinterface eine RGB-LED steuerbar sein, die als Statusanzeige dient. Die benötigten Bauteile erhielt ich von der Schule. Ich bekam einen ESP32-C3 DevKitM1 Mikrocontroller, einen DHT11 Sensor zur Messung von Temperatur und Luftfeuchtigkeit, einen Hallsensor, ein 0,96-Zoll OLED-Display mit 128x64 Pixeln, ein Steckbrett, mehrere Jumperkabel sowie ein USB-Mini-Kabel zur Programmierung und Stromversorgung.

Zuerst prüfte ich die technischen Eigenschaften der Komponenten und plante die Pinbelegung des ESP32-C3. Ich schloss den DHT11 Sensor an GPIO2 an, wobei ich den Datenpin mit einem 10kΩ Pull-Up-Widerstand gegen 3.3V absicherte. Die Stromversorgung des Sensors erfolgte über die 3.3V- und GND-Pins des Boards. Danach verband ich das OLED-Display mit dem ESP32 über Software-I2C. SDA legte ich auf GPIO1 und SCL auf GPIO10, da diese Pins für Software-I2C am ESP32-C3 gut geeignet sind. Der Hallsensor wurde mit GPIO3 verbunden. Da mein Hallsensor sowohl digital als auch analog ausgelesen werden konnte, entschied ich mich nach einem ersten Funktionstest für die analoge Auswertung, um feinere Schwankungen in der Magnetfeldstärke zu erkennen. Ich schloss den Sensor entsprechend an 3.3V und GND an und las die Werte über analogRead() im Code ein.

Für die Entwicklung verwendete ich die Arduino IDE in der aktuellen Version. Ich installierte über den Boardverwalter das ESP32-Boardpaket von Espressif und wählte das Board "ESP32C3 Dev Module" aus. Im Bibliotheksverwalter fügte ich alle notwendigen Bibliotheken hinzu: „DHT sensor library“, „Adafruit Unified Sensor“, „U8g2“ für die OLED-Anzeige, „Adafruit NeoPixel“ für die RGB-LED, „ESPAsyncWebServer“ und „AsyncTCP“ für den Webserver sowie „WiFiManager“ zur einfachen WLAN-Konfiguration. Ich testete jede Komponente einzeln mit einem Minimalcode, um sicherzustellen, dass sie korrekt funktionierte. Der DHT11 wurde zunächst mit einem einfachen Sketch getestet, der Temperatur und Luftfeuchtigkeit auf

dem seriellen Monitor ausgab. Danach schrieb ich einen Code, der mit der U8g2-Bibliothek testweise Text auf dem OLED-Display ausgab, um sicherzustellen, dass die I2C-Kommunikation funktionierte. Für die RGB-LED verwendete ich einen Beispielcode, mit dem verschiedene Farben durchgeschaltet wurden. Für den Hallsensor schrieb ich ein kurzes Skript, das den analogen Wert alle 500ms auf der seriellen Konsole ausgab. Ich hielt einen Magneten an den Sensor und beobachtete, wie sich der Wert veränderte, was funktionierte.

Nachdem alle Komponenten funktionierten, begann ich mit der Entwicklung des vollständigen Codes. Ich integrierte zuerst den WiFiManager, der beim ersten Start einen Access Point öffnete und eine einfache Website bereitstellte, über die ich die WLAN-Zugangsdaten eingeben konnte. Dadurch entfiel das manuelle Eintragen im Code. Nach erfolgreicher Verbindung zum WLAN wurde über configTime() ein NTP-Zeitserver eingebunden, um aktuelle Uhrzeit und Datum zu erhalten. Diese Daten wurden später für die zeitliche Zuordnung der Messwerte verwendet. Ich programmierte die Hauptschleife so, dass alle 5 Sekunden Temperatur, Luftfeuchtigkeit und der Hallsensorwert erfasst wurden. Aus mehreren Einzelmessungen wurde jeweils der Mittelwert berechnet. Extremwerte wurden automatisch als Ausreißer erkannt und ignoriert, was die Genauigkeit verbesserte. Die aktuellen Werte wurden auf dem OLED-Display dargestellt, das alle paar Sekunden neu beschrieben wurde. Dabei wurde die Schriftart so gewählt, dass alle relevanten Informationen gleichzeitig auf dem kleinen Display lesbar waren. Ich verwendete eine feste Anordnung: oben links die Uhrzeit, darunter Temperatur und Luftfeuchtigkeit, rechts daneben der Hallwert.

Als Nächstes programmierte ich den Webserver mit ESPAsyncWebServer. Ich erstellte eine HTML-Seite mit eingebettetem CSS und JavaScript. Die Seite zeigte aktuelle Messwerte an, die über einen JSON-Endpunkt alle 5 Sekunden per AJAX abgefragt wurden. Für den historischen Verlauf nutzte ich Chart.js, eine JavaScript-Bibliothek für interaktive Diagramme. Die Daten wurden auf dem ESP32 in Arrays mit Zeitstempeln gespeichert. Im Browser wurden die letzten 10 Minuten grafisch als Linie dargestellt – sowohl Temperatur als auch Luftfeuchtigkeit. Zusätzlich baute ich eine Schaltfläche in das Webinterface ein, mit der man die RGB-LED ein- und ausschalten konnte. Die Kommunikation erfolgte über eine einfache GET-Anfrage, die vom ESP32 ausgewertet wurde und die LED je nach Parameterstatus aktivierte oder deaktivierte. Der aktuelle LED-Zustand wurde auch im Browser visualisiert.

Zum Abschluss kombinierte ich alle Komponenten in einem einzigen Sketch. Ich strukturierte den Code sauber in Abschnitte: WLAN-Setup, Sensorauslesung, Datenverarbeitung, Displayausgabe, Webserverrouten und LED-Steuerung. Der HTML- und JavaScript-Code wurde direkt in einem index_html-String gespeichert, um keine externen Dateien laden zu müssen. Ich testete die Wetterstation im laufenden Betrieb über mehrere Stunden. Ich überprüfte, ob alle Sensorwerte korrekt aktualisiert wurden, ob die Anzeige stabil blieb und ob der Verlauf im Diagramm korrekt dargestellt wurde.

Die LED ließ sich zuverlässig per Webinterface steuern, und der Zugriff auf die Weboberfläche funktionierte sowohl über PC als auch Smartphone. Alle Anforderungen der Aufgabenstellung wurden erfüllt. Die Wetterstation arbeitete stabil, die Messwerte waren zuverlässig, das Interface war benutzerfreundlich und übersichtlich. Dieses Projekt zeigte mir, wie alle Komponenten – Elektronik, Sensorik, Webentwicklung und Microcontrollerprogrammierung – zusammenarbeiten müssen, um ein funktionierendes System zu ergeben. Durch die detaillierte Dokumentation ist es für jeden mit Grundkenntnissen in Arduino möglich, die Wetterstation mit identischen Bauteilen exakt nachzubauen und weiterzuentwickeln.

Code

```
#include <WiFi.h>
#include <WiFiManager.h>
#include <ESPAsyncWebServer.h>
#include <AsyncTCP.h>
#include <U8g2lib.h>
#include "DHT.h"
#include "time.h"
#include <Adafruit_NeoPixel.h>
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
#define HALLPIN 3
#define NEOPIXEL_PIN 8
#define NUMPIXELS 1
Adafruit_NeoPixel pixel(NUMPIXELS, NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800);
bool rgbEnabled = true;
U8G2_SSD1306_128X64_NONAME_F_SW_I2C u8g2(U8G2_R0, 10, 1, U8X8_PIN_NONE);
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 3600;
const int daylightOffset_sec = 3600;
AsyncWebServer server(80);
float lastTemp = NAN;
float lastHum = NAN;
int hallValue = 0;
unsigned long lastDisplayUpdate = 0;
const unsigned long displayUpdateInterval = 1000;
unsigned long lastMeasurement = 0;
const unsigned long measurementInterval = 10000;
#define MAX_DATA_POINTS 50
String labels[MAX_DATA_POINTS];
float tempData[MAX_DATA_POINTS];
float humData[MAX_DATA_POINTS];
int dataIndex = 0;
float averageTemperature() {
```

```
  float sum = 0;
  int count = 0;
  for (int i = 0; i < 5; i++) {
    float t = dht.readTemperature();
    if (!isnan(t) && t > -40 && t < 80) {
      sum += t;
      count++;
    }
    delay(100);
  }
  return count > 0 ? sum / count : NAN;
}
float averageHumidity() {
  float sum = 0;
  int count = 0;
  for (int i = 0; i < 5; i++) {
    float h = dht.readHumidity();
    if (!isnan(h) && h >= 0 && h <= 100) {
      sum += h;
      count++;
    }
    delay(100);
  }
  return count > 0 ? sum / count : NAN;
}
void setPixelColor(uint8_t r, uint8_t g, uint8_t b) {
  if (!rgbEnabled) {
    pixel.clear();
    pixel.show();
    return;
  }
  pixel.setPixelColor(0, pixel.Color(r, g, b));
  pixel.show();
}
void updateDisplay(struct tm timeinfo) {
  u8g2.clearBuffer();
  u8g2.setFont(u8g2_font_ncenB08_tr);
  char timeStr[20];
  strftime(timeStr, sizeof(timeStr), "%H:%M:%S", &timeinfo);
  u8g2.drawStr(0, 12, timeStr);
  String tempStr = "Temp: " + String(lastTemp, 1) + " C";
  String humStr = "Feuchte: " + String(lastHum, 1) + " %";
  String hallStr = "Hall: " + String(hallValue);
  u8g2.drawStr(0, 30, tempStr.c_str());
  u8g2.drawStr(0, 45, humStr.c_str());
  u8g2.drawStr(0, 60, hallStr.c_str());
  u8g2.sendBuffer();
}
```

```
String createHTML() {
  String html = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>ESP32 Wetterstation</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      background: #f0f4f8;
      color: #333;
      text-align: center;
      padding: 20px;
    }
    h1 {
      color: #2c3e50;
    }
    #values {
      background: #fff;
      padding: 15px;
      border-radius: 10px;
      box-shadow: 0 2px 5px rgba(0,0,0,0.1);
      max-width: 400px;
      margin: 20px auto;
    }
    #values p {
      font-size: 1.1em;
      margin: 10px 0;
    }
    label {
      display: inline-block;
      margin-bottom: 15px;
      font-weight: bold;
    }
    input[type="checkbox"] {
      transform: scale(1.3);
      margin-left: 10px;
    }
    canvas {
      max-width: 100%;
      background: #fff;
      border-radius: 10px;
      box-shadow: 0 2px 5px rgba(0,0,0,0.1);
      padding: 10px;
      margin-top: 20px;
    }
```

```
 </style>
</head>
<body>
 <h1>ESP32 Wetterstation</h1>
 <label>Status-LED:
  <input type="checkbox" id="ledToggle" checked
onchange="toggleLED(this.checked)">
 </label>
 <div id="values">
  <p><b>Aktuelle Temperatur:</b> <span id="temp">--</span> °C</p>
  <p><b>Aktuelle Luftfeuchtigkeit:</b> <span id="hum">--</span> %</p>
  <p><b>Aktueller Hall-Wert:</b> <span id="hall">--</span></p>
 </div>
 <canvas id="tempChart" width="400" height="200"></canvas>
 <script>
 let tempChart;
 async function fetchData() {
  try {
   const res = await fetch('/data');
   const json = await res.json();
   document.getElementById("temp").innerText = json.temp.toFixed(1);
   document.getElementById("hum").innerText = json.hum.toFixed(1);
   document.getElementById("hall").innerText = json.hall;
   if (!tempChart) {
    const ctx = document.getElementById('tempChart').getContext('2d');
    tempChart = new Chart(ctx, {
     type: 'line',
     data: {
      labels: json.labels,
      datasets: [
       {
        label: 'Temperatur (°C)',
        data: json.temps,
        borderColor: 'crimson',
        backgroundColor: 'rgba(220, 20, 60, 0.1)',
        fill: true,
        tension: 0.3
       },
       {
        label: 'Feuchtigkeit (%)',
        data: json.hums,
        borderColor: 'dodgerblue',
        backgroundColor: 'rgba(30, 144, 255, 0.1)',
        fill: true,
        tension: 0.3
       }
      ]
     },
```

```
        options: {
          responsive: true,
          animation: false,
          scales: {
            y: {
              beginAtZero: true
            }
          },
          plugins: {
            legend: {
              labels: {
                color: '#333'
              }
            }
          }
        }
      });
    } else {
      tempChart.data.labels = json.labels;
      tempChart.data.datasets[0].data = json.temps;
      tempChart.data.datasets[1].data = json.hums;
      tempChart.update();
    }
  } catch (error) {
    console.error('Fehler beim Laden der Daten:', error);
  }
 }
 function toggleLED(state) {
  fetch('/led?state=' + (state ? 'on' : 'off'));
 }
 fetchData();
 setInterval(fetchData, 5000);
</script>
</body>
</html>
)rawliteral";
 return html;
}
void setupWebServer() {
 server.on("/", HTTP_GET, [](AsyncWebServerRequest* request) {
  request->send(200, "text/html", createHTML());
 });
 server.on("/data", HTTP_GET, [](AsyncWebServerRequest* request) {
 String json = "{\"labels\":[";
 for (int i = 0; i < dataIndex; i++) {
  json += "\"" + labels[i] + "\"";
  if (i < dataIndex - 1) json += ",";
 }
```

```
  json += "],\"temps\":[";
  for (int i = 0; i < dataIndex; i++) {
    json += String(tempData[i], 1);
    if (i < dataIndex - 1) json += ",";
  }
  json += "],\"hums\":[";
  for (int i = 0; i < dataIndex; i++) {
    json += String(humData[i], 1);
    if (i < dataIndex - 1) json += ",";
  }
  json += "],";
  json += "\"temp\":" + String(lastTemp, 1) + ",";
  json += "\"hum\":" + String(lastHum, 1) + ",";
  json += "\"hall\":" + String(hallValue);
  json += "}";
  request->send(200, "application/json", json);
});
  server.on("/led", HTTP_GET, [](AsyncWebServerRequest* request) {
    if (request->hasParam("state")) {
      String state = request->getParam("state")->value();
      rgbEnabled = (state == "on");
    }
    request->send(200, "text/plain", rgbEnabled ? "on" : "off");
  });
  server.begin();
}
void setup() {
  Serial.begin(115200);
  dht.begin();
  pinMode(HALLPIN, INPUT);
  pixel.begin();
  pixel.setBrightness(50);
  pixel.show();
  u8g2.begin();
  WiFiManager wm;
  if (!wm.autoConnect("ESP_Config")) {
    setPixelColor(255, 0, 0); // rot = kein WLAN
    delay(1000);
    ESP.restart();
  }
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  struct tm timeinfo;
  while (!getLocalTime(&timeinfo)) {
    delay(500);
  }
  setupWebServer();
}
```

```
void loop() {
  hallValue = analogRead(HALLPIN);
  setPixelColor(0, 0, 255); // Messung im Gange
  float temperature = averageTemperature();
  float humidity = averageHumidity();
  if (!WiFi.isConnected()) setPixelColor(255, 0, 0); // rot
  else if (temperature > 28.0) setPixelColor(255, 165, 0); // orange
  else setPixelColor(0, 255, 0); // grün
  unsigned long nowMillis = millis();
  if (nowMillis - lastDisplayUpdate > displayUpdateInterval) {
    lastDisplayUpdate = nowMillis;
    struct tm timeinfo;
    if (getLocalTime(&timeinfo)) updateDisplay(timeinfo);
  }
  if (nowMillis - lastMeasurement > measurementInterval) {
    lastMeasurement = nowMillis;
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) return;
    if (isnan(temperature) || isnan(humidity)) return;
  lastTemp = temperature;    // <--- hier setzen
  lastHum = humidity;
    char label[10];
    strftime(label, sizeof(label), "%H:%M", &timeinfo);
    if (dataIndex < MAX_DATA_POINTS) {
      labels[dataIndex] = String(label);
      tempData[dataIndex] = temperature;
      humData[dataIndex] = humidity;
      dataIndex++;
    } else {
      for (int i = 1; i < MAX_DATA_POINTS; i++) {
        labels[i - 1] = labels[i];
        tempData[i - 1] = tempData[i];
        humData[i - 1] = humData[i];
      }
      labels[MAX_DATA_POINTS - 1] = String(label);
      tempData[MAX_DATA_POINTS - 1] = temperature;
      humData[MAX_DATA_POINTS - 1] = humidity;
    }
  }
}
```

Zusammenfassung

Das Projekt bestand darin, eine funktionsfähige Wetterstation mit einem ESP32-C3 Mikrocontroller aufzubauen, die Temperatur-, Luftfeuchtigkeits- und Hallsensordaten erfasst, lokal auf einem OLED-Display darstellt, über eine Weboberfläche zugänglich macht und dort ein historisches Graph der Messwerte darstellt. Zusätzlich sollte eine RGB-LED per Webinterface steuerbar sein. Während der Umsetzung traten einige Schwierigkeiten auf, insbesondere bei der initialen Einrichtung der I2C-Verbindungen und bei der Auswahl stabiler Bibliotheken für den ESP32-C3, da nicht alle Beispielcodes kompatibel waren. Auch die Integration von WiFiManager und ESPAsyncWebServer erforderte einige Debugging-Zyklen, da es bei falscher Bibliotheksversion zu Kompilierungsfehlern kam. Kleinere Probleme zeigten sich bei der Datenübertragung zwischen JavaScript und Webserver sowie bei der korrekten Zeitstempelung über NTP. Diese wurden jedoch durch sorgfältige Recherche und Testläufe gelöst. Letztlich konnte das Projekt erfolgreich umgesetzt und alle vorgesehenen Funktionen realisiert werden.

Literaturverzeichnis

[1] Arduino - DHT11 | Arduino Tutorial “, Arduino Getting Started. Zugegriffen: 27. Mai 2025. [Online]. Verfügbar unter: https://arduinogetstarted.com/tutorials/arduino-dht11

[2] Chart.js “. Zugegriffen: 27. Mai 2025. [Online]. Verfügbar unter: https://www.w3schools.com/js/js_graphics_chartjs.asp

[3] „ESP32 - OLED“, ESP32 Tutorial. Zugegriffen: 27. Mai 2025. [Online]. Verfügbar unter: https://esp32io.com/tutorials/esp32-oled

[4] S. Santos, „ESP32 with DHT11/DHT22 Temperature and Humidity Sensor using Arduino IDE | Random Nerd Tutorials“. Zugegriffen: 27. Mai 2025. [Online]. Verfügbar unter: https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/

[5] Tiago, „ESP32-C3 Super Mini Real-Time Clock with OLED Display “. Zugegriffen: 27. Mai 2025. [Online]. Verfügbar unter: https://www.edgemicrotech.com/esp32-c3-super-mini-real-time-clock-with-oled-display/

[6] ESP32Async/ESPAsyncWebServer. (1. März 2025). C++. ESP32 Asynchronous Networking. Zugegriffen: 27. Mai 2025. [Online]. Verfügbar unter: https://github.com/ESP32Async/ESPAsyncWebServer

[7] tzapu, tzapu/WiFiManager. (1. März 2025). C++. Zugegriffen: 27. März 2025. [Online]. Verfügbar unter: https://github.com/tzapu/WiFiManager

[8] title, „Random Nerd Tutorials“Wifimanager. Zugegriffen: 27.März 2025 [Online]. Verfügbar unter: https://randomnerdtutorials.com/esp32-wi-fi-manager-asyncwebserver/