Wetterstation

13. Januar 2023

Vorwort

Hierbei handelt es sich um die Dokumentation zu unserem Microcontrollerprojekt im Kurs **Signalverarbeitung 2** im 3. Semester des Bachelorstudiengangs Software Engineering. Die Software befindet sich in unserem öffentlichen Git Repository.

Motivation

Die Wetterstation soll es ermöglichen Umgebungsmetriken aufzunehmen und diese dann über einen MQTT Broker zu veröffentlichen. Das System sendet nur Daten, hört also keinem Topic zu. Diese Daten können dann z.B. für Smarthome-Anwendungen verwendet werden. Folgende Metriken werden aufgezeichnet: Luftfeuchtigkeit, Luftqualität(Rauch) und Temperatur. Das System wird auf einem Espressif ESP8266 Mikrocontroller realisiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Informationsbeschaffung und Planung

Wie bereits in der Motivation erwähnt, soll unser System Daten der Umgebung anhand von Sensoren erfassen.

Wichtige Sensoren für eine Wetterstation sind z.B. **Luftfeuchtigkeit** und **Temperatur**. Es wird außerdem der Entschluss gefasst die **Luftqualität** zu analysieren.

Folgende **Sensoren** wurden ausgewählt:

- DHT-11, Sensor zum Messen von Temperatur und Luftfeuchtigkeit
- MQ-2, Sensor zur Detektierung von Gasen

Um das erfassen von Gasen auditiv zu gestalten, wurde ein **Piepser** verwendet. Dieser soll das überschreiten eines bestimmten Wertes signalisieren.

Eines der Schlüsselfähigkeiten des verwendeten **ESP8266 Boards** ist die Wifi und Bluetooth Funktion. Um diese zu nutzen, soll der µC über Wifi Daten versenden können. Ein weit verbreitetes Protokoll ist hierbei MQTT. Mithilfe dieses Protokolls können Systeme untereinander Kommunizieren. Die Idee ist, dass der Controller über einen MQTT Server (auch Broker genannt) Daten veröffentlichen kann. Messergebnisse stehen dann z.B. für Smarthome Systeme wie Homeassistant zur Verfügung.

1.1 DHT-11 (Temperatur und Luftfeuchtigkeit)

Um den DHT-11 Sensor auslesen zu können wird ein GPIO Port mit Tri-State oder Open-Drain benötigt. Bei der Kommunikation zwischen dem Host und dem Sensor handelt es sich um einen Ein-Draht-Bus.

Wenn wir uns das Datagramm des Sensors anschauen, dann sehen wir, dass der Host, in unserem Fall der µC, das **'Startsignal'** geben muss. Hier wird der Eingang aktiv auf Low geschaltet. Hierauf reagiert der Sensor und sendet seine Daten über den Bus an den Host, welcher nach dem Startsignal auf das zusenden der Daten wartet.

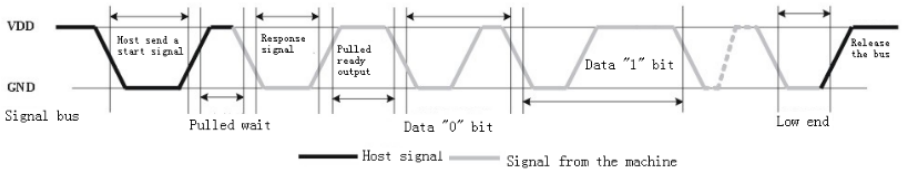

Abbildung 1: Datagramm DHT-11 Sensor[1]

Ein Datagramm hat insgesamt 40 Bit und besteht aus:

- 8 Bit Feuchtigkeitswert (Dezimal)
- 8 Bit Feuchtigkeitswert (Komma)
- 8 Bit Temperaturwert (Dezimal)
- 8 Bit Temperaturwert (Komma)
- 8 Bit Paritätsbit

Hier nun eine Beispielrechnung:

$$\frac{00010100}{Luftfeuchtigkeit}+\frac{00000000}{Nachk.Luftf.}+\frac{01000101}{Temperatur}+\frac{00000000}{Nachk.Temp}=\frac{01011001}{Parittsbit}$$

Die Empfangen Daten sind Korrekt da die Summe das **Paritätsbit** ergeben!

Hierbei sind die Übertragenen Werte...
Luftfeuchtigkeit: 0001 0100, 0000 0000 = 14,0H = 20,0%
Temperatur: 0100 0101, 0000 0000 = 45,0H = 69°C

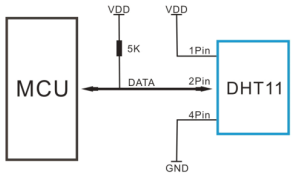

Abbildung 2: DHT-11 Anschlussplan[2]

1.2 MQ-2 (Gasdetektierung)

Mit dem MQ-2 Sensor lässt sich die Luftqualität beziehungsweise die Menge an bestimmten Gasen (Einheit: **PPM** - Parts Per Million) in der Luft. Diese bestimmten Gase sind: Wasserstoff, Kolbenstoffmonoxid, **Rauch** und noch weitere...

Zu demonstrationszwecken wurde **Rauch** ausgewählt. Das Prinzip zum Messen einer Gaskonzentration ist aber bei allen gleich. Da der Sensor über den Analogeingang mit einer Auflösung von 10 Bit betrieben wird und dadurch 'nur' Werte von 0 bis 1023 eingelesen werden, bedarf es einer Berechnung um auf die PPM Werte von Rauch in der Luft zu kommen. Hinzuzufügen ist noch, das der Sensor, um möglichst genau messen zu können, für 24h bei sauberer Luft betrieben werden muss. Im inneren des Sensors befindet sich eine Heizspule welchen den Sensor auf die spezifizierte Temperatur bringen soll.

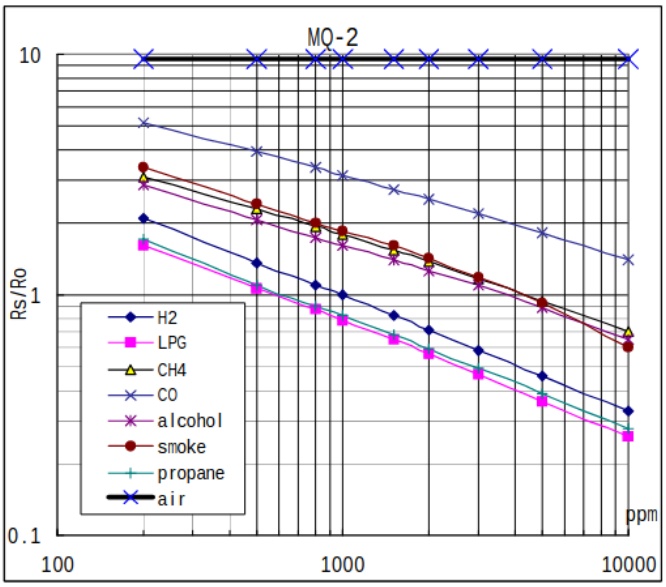

Abbildung 3: Logarithmische Skala des MQ-2[3]

Auf der Y-Achse wird der Messwiderstand Rs eines bestimmten Gases dargestellt. Während Ro der Messwiderstand bei sauberer Luft ist. Auf der X-Achse Sieht man die PPM.

Um die Steigung der Kurve bei einer logarithmischen Skala näherungsweise zu berechnen, betrachten wir die braune Kurve und fügen die Werte aus dem Schaubild in diese Formel ein:

$$slope = \frac{Y_2 - Y_1}{X_2 - X_1}$$

$$slope = \frac{\log(0,5) - \log(3,5)}{\log(10000) - \log(200)}$$

```
smoke_curve = [X1, Y1, slope]
curve_curve = [2.301, 0.554, -0.497]
```

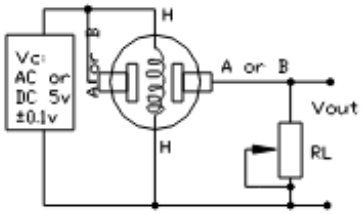

Abbildung 4: MQ-2 Schaltung[4]

Um bei der Messung des Sensors möglichst genau zu sein, ist es notwendig den Sensor vor beginn der Datenerfassung zu kalibrieren, bzw. seinen Widerstand Ro bei sauberer Luft zu berechnen.

1.3 MQTT

Wie bereits erwähnt wird die MQTT Architektur zum veröffentlichen der Nachrichten im Netzwerk genutzt. Es handelt sich hierbei um eine **publish-subscribe** Architektur. Der Client kommuniziert hier nicht direkt mit einem Server, sondern mit einer middleware, dem **Broker**, welcher die Nachrichten nur an den Client ausliefert wenn dieser ein bestimmtes Toppic **'abonniert'** hat.

In diesem Fall ist der Publisher der ESP8266. Dieser veröffentlicht Daten auf den Topics: '`/wetterstation/temperature`', '`/wetterstation/humidity`' und '`/wetterstation/smoke`'.

Die Payload der Nachrichten ist bei MQTT ein oder mehrere Strings beziehungsweise Strings die mit JSON encoded sind.

1.4 Programmablauf

Mit den Informationen zur Durchführung im Hinterkopf ergibt folgende Vorgehensweise:

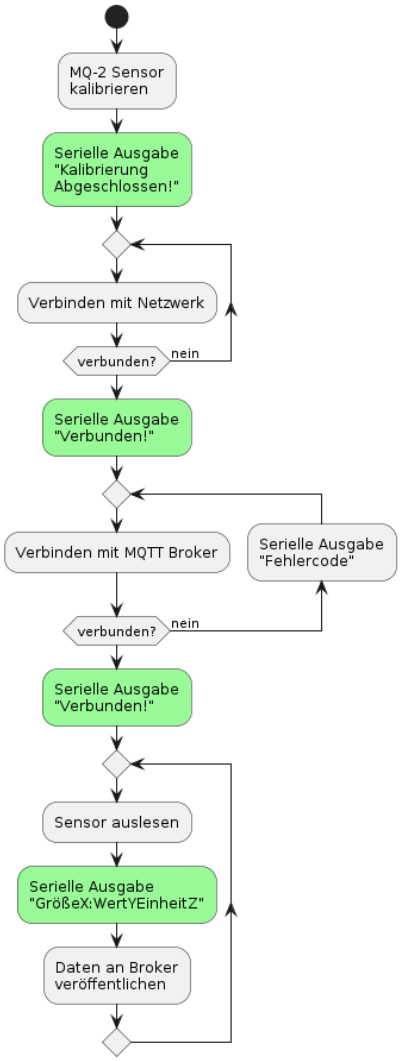

Abbildung 5: Aktivitätsdiagramm Wetterstation

2 Durchführung

2.1 DHT-11 Codeblock

Zum Auslesen der Werte für **Temperatur** und **Luftfeuchtigkeit** wurden die Programmbibliotheken `<DHT.h>`, `<DHT_U.h>` & `<Adafruit_Sensor.h>` verwendet.

Programmatisch wurde das Auslesen der Sensorwerte wie folgt durchgeführt...

Temperatur:

```
...
  if (isnan(event.temperature)) {
    Serial.println(F("Error reading temperature!"));
  }
  else {
    Serial.print(F("Temperature: "));
    Serial.print(event.temperature);
...
```

Fast gleich, hier aber nicht aufgeführt ist der Codeblock für die **Luftfeuchtigkeit**.

2.2 MQ-2 Codeblock

Um plausible Daten für die Rauchmessung zu erhalten, muss der MQ-2 Sensor an sauberer Luft kalibriert werden. Es wird das Arithmetische Mittel von Ro berechnet:

```
...
  for(int i = 0; i < 20; i++) {
    val += mqResistanceCalc(analogRead(MQPIN));
    Serial.print(F("."));
    delay(500);
  }
  val = val / 20;
  val = val / CLEAN_AIR_FACTOR;
...
```

Berechnung von Ro:

```
...
float mqResistanceCalc(int adc){
  return(((float)RL_VALUE*(1023/adc)/adc));
}
...
```

Wert von Rs bestimmen:

```
1  void mqRead(){
2  ...
3    float rs = 0;
4    for(int i = 0; i < 5; i++){
5      rs += mqResistanceCalc(analogRead(MQPIN));
6      delay(50);
7    }
8    rs = rs / 5;
9  ...
```

Verhältniss von Rs zu Ro berechnen und die in Abschnitt 1.2 berechnete 'Rauchkurve' als Parameter an Annäherungsfunktion übergeben:

```
1  ...
2  void mqRead(){
3  ...
4    float rs_ro_ratio = rs / ro;
5    ppm = smokeLogScale(rs_ro_ratio, smoke_curve);
6  ...
```

Berechnung des 'Rauchwerts' in PPM:

```
1  int smokeLogScale(float ratio, float* curve){
2    return pow(10, (((log(ratio) - curve[1]) / curve[2])
3            + curve[0]));
4  }
```

2.3 String Formatting für MQTT Message Publishing

Es wurde bereits im Kapitel MQTT darauf eingegangen, dass Nachrichten als Strings veröffentlicht werden. Da die gemessenen Werte aber andere Datentypen sind, müssen diese erst in einem String repräsentiert werden. Die Vorgehensweise ist bei bei allen drei Sensoren identisch. Daher nur ein Beispiel:

```
1  ...
2    ppm = smokeLogScale(rs_ro_ratio, smoke_curve);
3    ...
4    sprintf(ppm_str, "%d", ppm);  // String Formatting
5    const char* ppm_cstr = ppm_str;
6    client.publish("/wetterstation/smoke", ppm_cstr);
7  ...
```

Die letzte Zeile des vorherigen Codeblocks zeigt wie die Payload veröffentlicht wird (Ebenfalls identisch zum Rest).

2.4 MQTT Konfiguration

Der Broker muss um Nachrichten zu Empfangen, gestartet und Konfiguriert werden. Das ist über verschiedene Wege möglich. Es gibt MQTT Service Anbieter im Internet. Diese kann man für Tests verwenden, man sollte aber vorsichtig sein, da die Daten öffentlich für jeden Subscriber sind, der auf dem Topic zuhört.

Alternativ dazu können MQTT Broker auch als Deamon unter einem Betriebssystem oder in einem Docker Container gestartet werden. Smarthome Systeme wie Homeassistant erlauben es auch MQTT Broker zu integrieren und diese als Addon laufen zu lassen. In diesem Fall wird Mosquitto als Homeassistant Addon verwendet.

2.4.1 Publisher

Wetterstation benötigt um auf dem Broker zu veröffentlichen Benutzer und Zugriffsrechte. Angenommen die Daten sind vorhanden, so werden diese in die unter `‘wetterstation/src/mqtt_config.h‘` als Werte zu den Entsprechenden Variablen eingetragen:

```
1  const char* mqtt_server = "<Broker␣IP␣Adresse>";
2  const int mqtt_port = 1883; // Standart Port
3  const char* mqtt_user = "<Benutzername␣MQTT-User>";
4  const char* mqtt_password = "<Passwort␣MQTT-Broker>";
```

2.4.2 Subscriber

Der Subscriber kann ein beliebiger client sein. Als Beispiel wird hier ein Server verwendet. In der Konfiguration des Servers müssen ebenfalls Benutzer und Zugriffsdaten hinterlegt werden. Natürlich müssen diese auch Vorhanden sein. In Homeassistant unter `‘configuration.yaml‘` können diese eintragen werden:

```
1  ...
2  mqtt:
3    broker: <Adresse des Brokers>
4    username: "<username>"
5    password: "<password>"
6    sensor:
7      - name: "<freien␣Namen␣vergeben>"
8        state_topic: "/wetterstation/humidity"
9        unit_of_measurement: "%"
10     ...
```

3 Resultat

Das Projekt konnte erfolgreich beendet werden. Es ist gelungen Messgrößen von (digital & analog) einzulesen und diese in einem Netzwerk zu weiteren verarbeitung zu veröffentlichen. Durch die Grundlagen die durch dieses Projekt geschaffen wurden, können nun jegliche Anwendungsfälle bezogen auf MQTT und Sensoren in angriff genommen werden. Folgend noch ein paar Bilder zur erfolgreichen Integration in die Smarthome Anwendung.

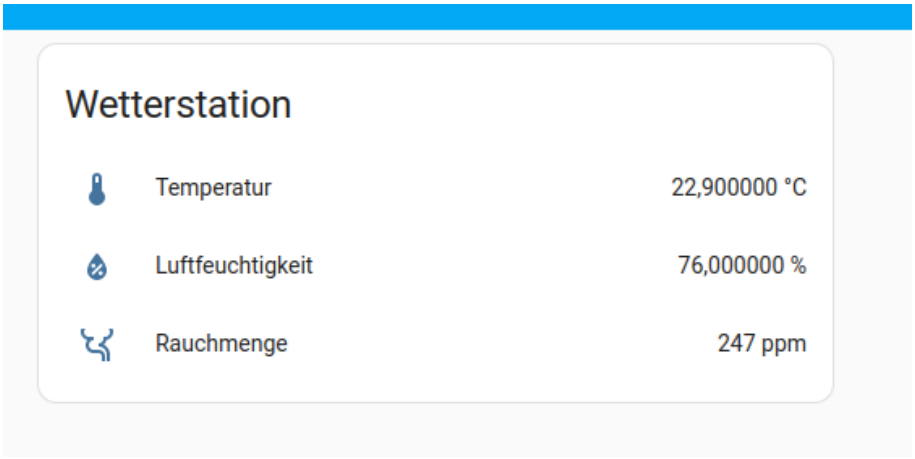

Abbildung 6: Karte mit allen Senorwerten (Rauchtest)

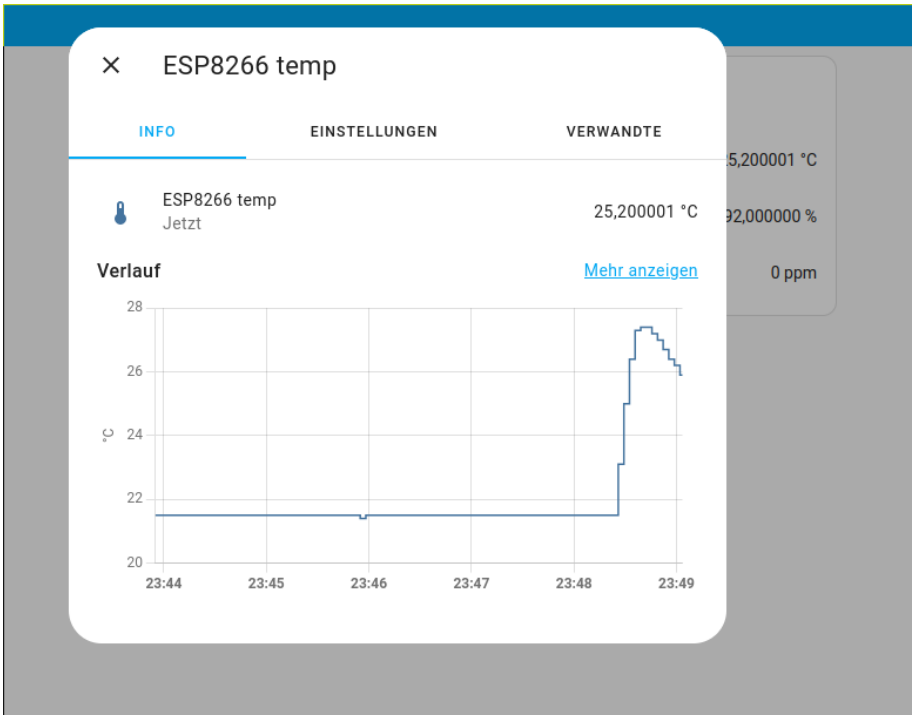

Abbildung 7: Verlauf Temperatur

4 Quellen

Prof. Dr. Frankhauser, Thomas.MQTT. Abgerufen am: 13.01.2023. [3]Unbekannt. Datasheet MQ-2. Abgerufen am: 13.01.2023. Das, Debashis. How Does MQ-2 Flammable Gas and Smoke Sensor Work with Arduino?. Abgerufen am: 13.01.2023. Unbekannt. ESP8266 MQTT Client: Publish and Subscribe Node-RED Dashboard. Abgerufen am: 13.01.2023. [1]Unbekannt. DHT11 Product Manual. Abgerufen am: 13.01.2023. [2]Unbekannt. Connection Diagramm. Abgerufen am: 13.01.2023. [4]Gupta, Sourav, Gas Detection and PPM Measurement using PIC Microcontroller and MQ Gas Sensors. Abgerufen am: 13.01.2023.