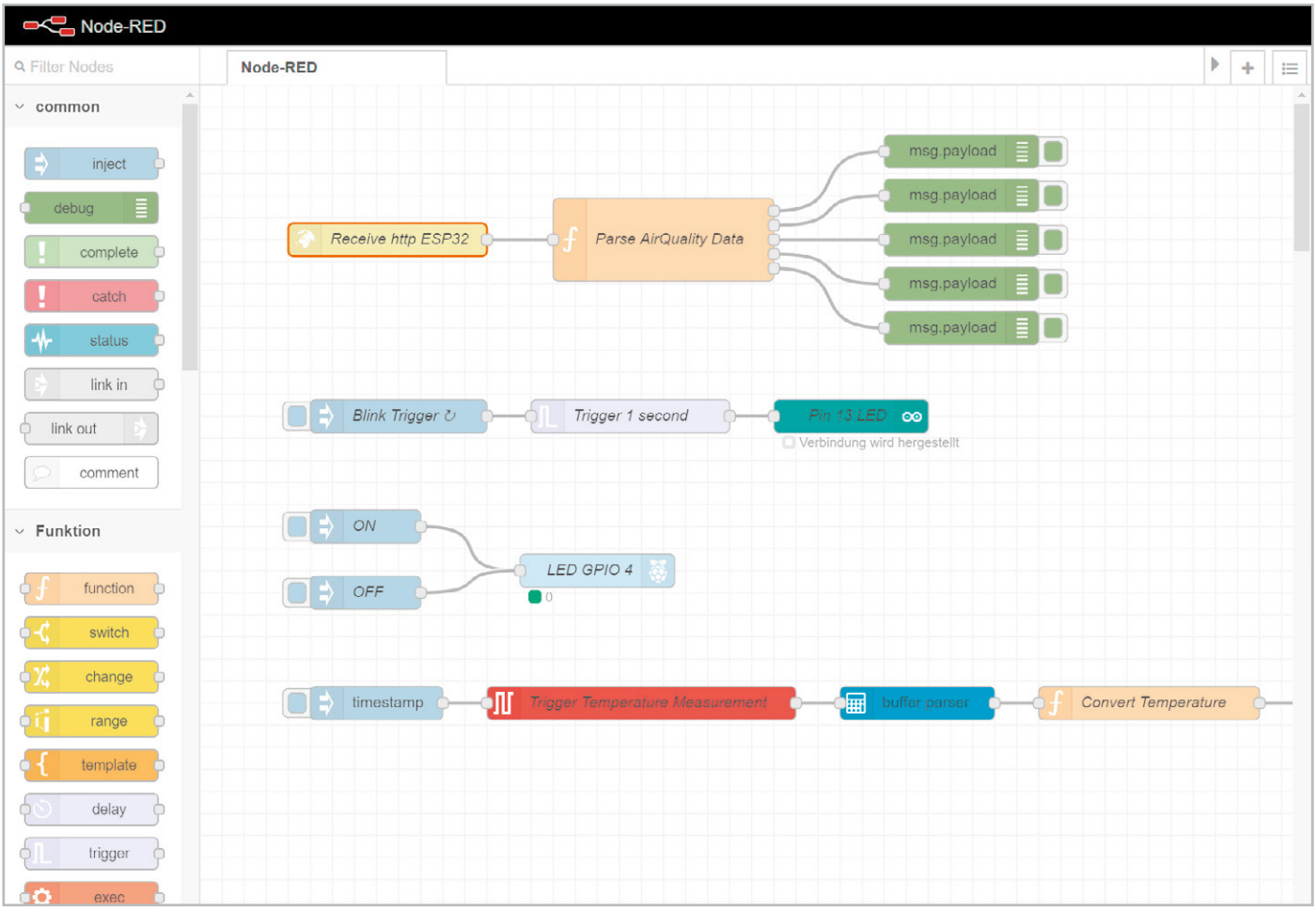
Programmieren (fast) ohne Code

Einbinden von Raspberry Pi, ESP32, Arduino und Elektronik-Bauteilen in Node-RED

Teil 2

Im ersten Teil des Beitrags zu Node-RED haben wir uns mit der Installation und den Grundlagen des praktischen Prototyping-Tools beschäftigt. Diesmal wollen wir uns die Möglichkeiten anschauen, elektronische Bauteile in Node-RED zu integrieren. Zum einen schließen wir dabei direkt Bauteile an den Raspberry Pi, auf dem unser Node-RED läuft, an. Zum anderen nutzen wir einen Arduino-Mikrocontroller, den wir seriell verbinden, und zeigen schließlich mit einem ESP32 eine über Funk verbundene Internet-of-Things-Variante.



Vorarbeiten

Unter Linux und auch bei Node-RED empfiehlt es sich, hin und wieder das System bzw. die Module zu aktualisieren. Bei Linux erledigt man das manuell, indem man zunächst die Paketquellen in einem Terminal-Fenster mit:

sudo apt update

aktualisiert und anschließend die entsprechenden Software-Aktualisierungen mit

sudo apt upgrade

herunterlädt und dann ausführt.

In Node-RED geschieht eine Aktualisierung sowohl von Node-RED an sich als auch der installierten Pakete, indem man im Hamburger-Menü (drei waagerechte Striche im rechten oberen Bereich) „Palette verwalten" auswählt und die zu aktualisierenden Pakete installiert (Bild 1).

Unter Umständen muss man nach Installation eines Pakets Node-RED mit

node-red-stop

in einem neuen Konsolen-Fenster beenden und danach mit

node-red-start

erneut starten (Bild 2).

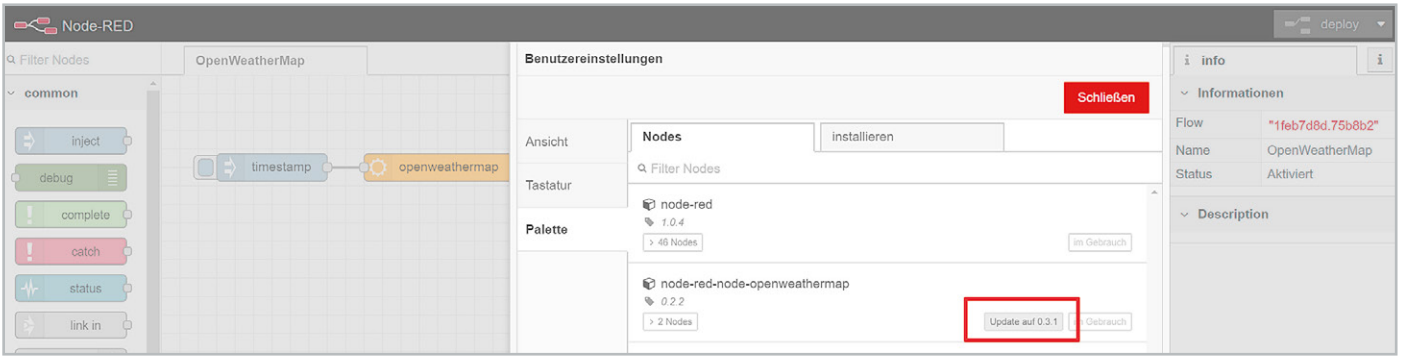


Bild 1: Aktualisierung der installierten Pakete unter Node-RED

Hardware-Basis Raspberry Pi

Da Node-RED bei den meisten Anwendern auf einem Raspberry Pi laufen dürfte, schauen wir uns als Erstes diese Hardware-Basis für die Anbindung von Elektronik-Bauteilen an. Alle Raspberry Pi-Boards (1/2/3/4) bieten mit dem zweireihigen, 40-poligen GPIO-(General Purpose Input/Output)-Anschluss im 2,54-mm-Rastermaß eine ideale Möglichkeit, um Elektronik-Bauteile oder sogar Platinen direkt anzuschließen. Dazu sehen wir uns zunächst die Anschlussleiste der Raspberry Pi-Versionen an (Bild 3).

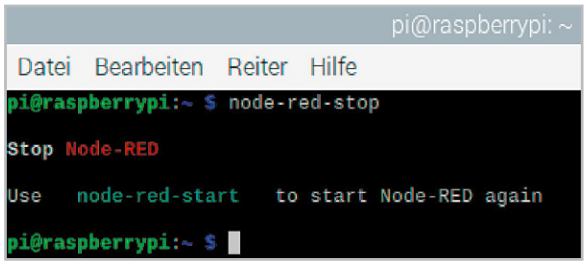


Bild 2: Node-RED wird mit node-red-stop sauber heruntergefahren.

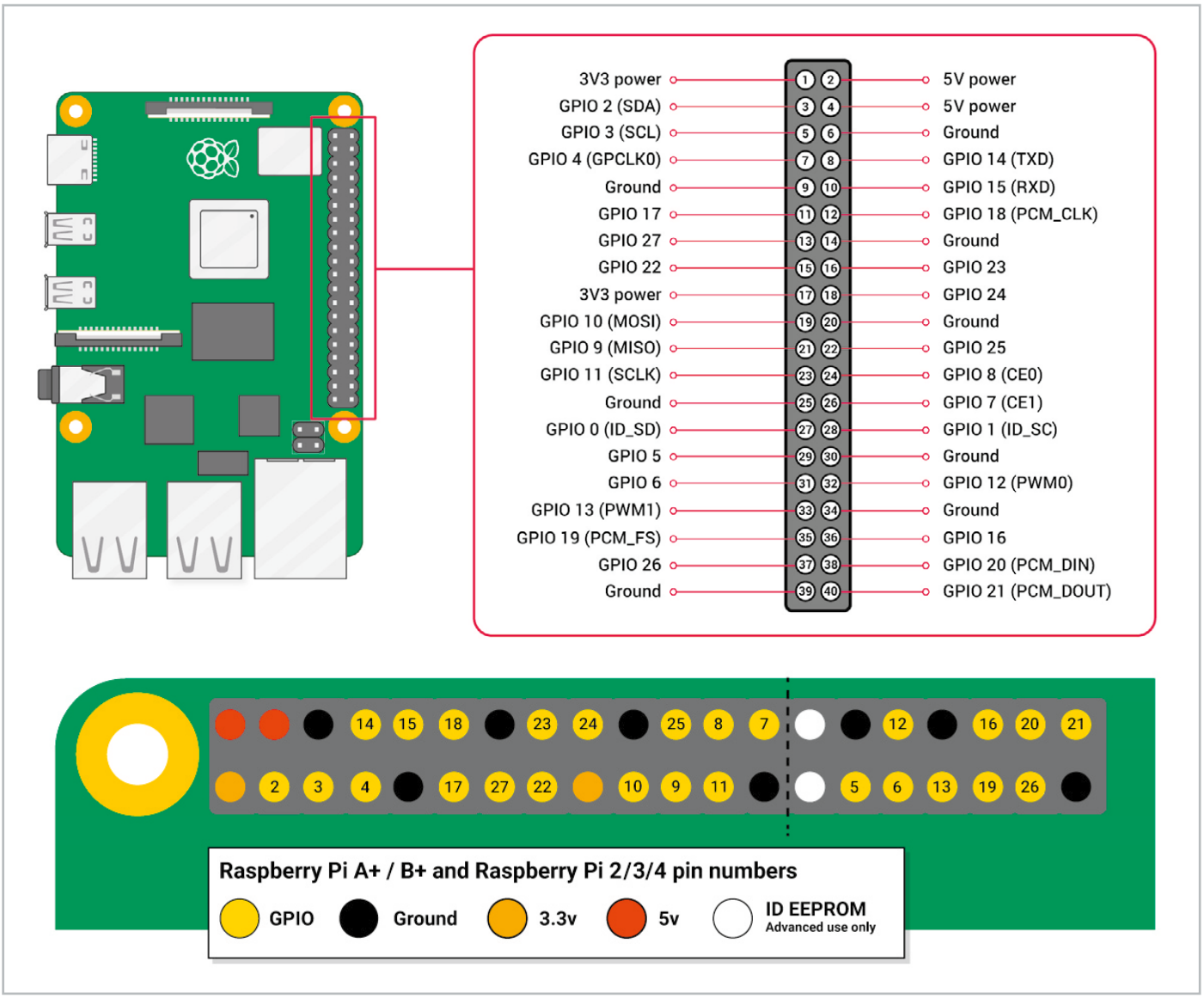


Bild 3: Pinout vom Raspberry Pi 1/2/3/4
Quelle: Raspberry Pi Foundation (https://www.raspberrypi.org/documentation/usage/gpio/README.md), https://creativecommons.org/licenses/by-sa/4.0/legalcode

Eine praktische Referenz kann auf dem Raspberry Pi durch Öffnen eines Terminalfensters und Ausführen des Befehls pinout aufgerufen werden. Dieses Tool wird von der GPIO-Zero-Python-Bibliothek zur Verfügung gestellt, die standardmäßig auf dem Desktop-Image von Raspberry Pi OS installiert ist, jedoch nicht auf Raspberry Pi OS Lite (Bild 4).

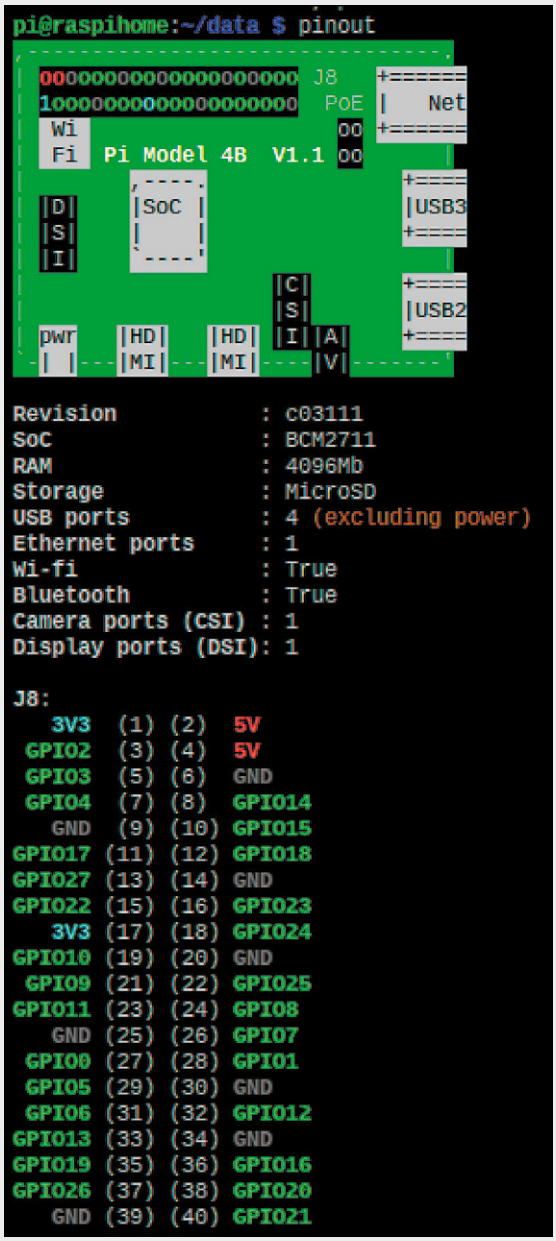


Bild 4: Mit dem Pinout-Befehl auf der Konsole kann die Pin-Belegung des verwendeten Raspberry Pi abgerufen werden.

An Spannungen werden 5 und 3,3 V an jeweils zwei Pins bereitgestellt, Masse (Ground) kann man gleich an acht Pins abgreifen. Manche GPIOs haben mehrere Funktionen, so sind z. B. GPIO 2 und GPIO 3 sowohl bidirektionale Ein- und Ausgänge als auch die Anschlusspins für I²C (SDA/SCL). Die GPIOs sind allerdings nur 3,3-V-tolerant, d. h. wenn Signale mit einem Logik-Level von 5 V an den Raspberry Pi angebunden werden sollen, muss man einen sogenannten Level-Shifter verwenden, um die Spannungshöhe von 5 V auf 3,3 V herunterzusetzen (und umgekehrt).

In unserem Prototypenadapter-Set PAD4 [1] ist beispielsweise ein bidirektionaler 8-Kanal-Pegelwandler zur Verbindung von Systemen mit unterschiedlichen Spannungspegeln vorhanden. Im weiteren Verlauf des Beitrags werden wir noch weitere Module aus diesem Set verwenden.

Während das Anschließen einfacher Komponenten an die GPIO-Pins in der Regel unkompliziert und sicher ist, sollte man beispielsweise bei LEDs Widerstände verwenden, um den durch sie fließenden Strom schon im Vorfeld zu begrenzen. Peripherie, die höhere Ströme verlangt, sollte generell über mit einer eigenen Spannungsversorgung ausgestattete Module (Beispiel: Motortreiber) erfolgen.

Die einzelnen GPIOs des Raspberry Pi können mit max. 16 mA und über alle GPIOs insgesamt mit 50 mA Strom belastet werden. Weitere interessante Werte zur Spannung bzw. den Stromverbräuchen der verschiedenen Raspberry Pi-Modelle in den unterschiedlichen Konfigurationen findet man unter [2].

Unbedingt aufpassen sollte man beim Anschließen externer Komponenten, damit man keinen Kurzschluss verursacht, der u. U. das gesamte Board unbrauchbar macht.

Hardware „Hallo Welt" mit LED in Node-RED

Analog zu unserem Software-Beispiel im ersten Teil zur Einführung in NODE-Red [3], bei dem wir einen Webserver aufgesetzt haben, der uns über eine Webseite die Meldung „Hallo Welt" zurückgegeben hat, starten wir die Anbindung von Hardware mit einer einfachen LED, die wir mit einem geeigneten Widerstand an den Raspberry Pi anschließen und danach im Node-RED-Flow an- und ausschalten können.

Tipp: Der als Schwellenspannung oder auch Vorwärtsspannung (Vf) bezeichnete Spannungsabfall einer in Durchlassrichtung betriebenen LED (LED = Light-Emitting Diode) liegt bei roten, grünen und gelben LEDs bei ungefähr 2 V, wobei je nach Bauteil sehr unterschiedliche, maximal zulässige Stromaufnahmen und Schwellenspannungen zugrunde liegen können. Daher sollte man sich für die jeweils tatsächlich verwendete LED im Datenblatt die entsprechenden elektrischen Spezifikationen genau anschauen.

Für unser Rechenbeispiel nehmen wir eine Vorwärtsspannung von 2 V und einen Stromverbrauch von 3 mA an, daraus ergibt sich unter Verwendung des ohmschen Gesetzes (R = U/I):

R (Widerstand in Ω) = (3,3 V - 2 V)/0,003 A = 500 Ω

Wir können also für unser Rechenbeispiel einen 470-Ω-Widerstand als Strombegrenzung unserer LED für den Anschluss an einem GPIO des Raspberry Pi verwenden.

In unserem Prototypenadapter Set PAD2 [4] ist übrigens – neben den zahlreichen anderen Modulen wie Operationsverstärker, Transistoren, MOSFETs und Spannungsreglern – ein Adapter mit einer roten Kingbright-LED (V_f 1,95–2,5 V) und einem entsprechenden Vorwiderstand (470 Ω) zur einfachen Verwendung auf einem Breadboard enthalten, hier beträgt der tatsächliche Stromverbrauch an 3,3 V 3,2 mA.

Vorbereiten von Node-RED

Nachdem wir die richtigen Werte für den Anschluss der LED mit dem Vorwiderstand ermittelt haben, können wir mit der Verdrahtung beginnen. Es empfiehlt sich, nicht nur für erste Tests, sondern auch für alle folgenden Beispiele in diesem Beitrag vorher die Spannungsversorgung vom Raspberry Pi nach dem vorhergehenden ordnungsgemäßen Herunterfahren per

sudo shutdown -h now

zu entfernen.

Ist dies erfolgt, schließen wir per Steckkabel an GPIO 4 des Raspberry Pi zunächst den Widerstand an und verbinden diesen dann mit der Anode (+, langes Beinchen) der LED. Die Katode (-, kurzes Beinchen) der LED wird dann wiederum mit einem Ground-Pin des Raspberry Pi verbunden. Der Einsatz unserer Prototypenadapter macht den Aufbau einfacher und platzsparender (Bild 5).

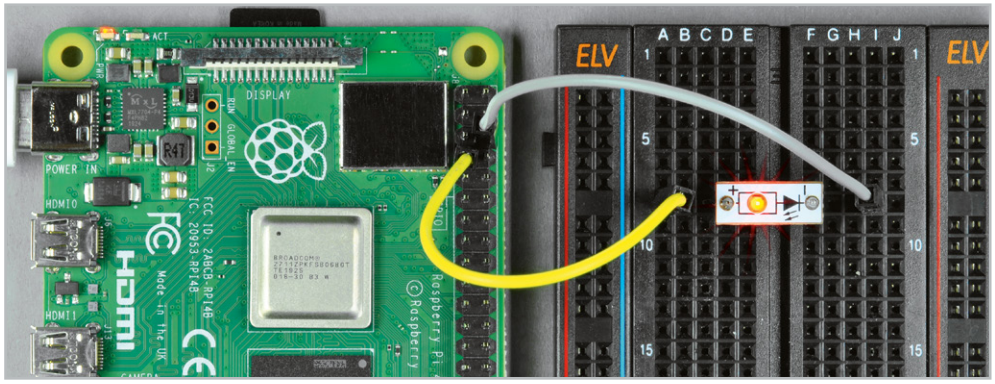


Bild 5: Anschluss der LED über GPIO 4 und Ground (mit LED-Prototypendapter aus PAD2)

Wir sollten diese – wenn auch einfache Schaltung – zunächst kontrollieren, bevor wir den Raspberry Pi wieder mit Spannung versorgen.

Nodes zum Steuern von GPIOs

Zum Ansteuern der GPIOs benötigen wir das entsprechende Paket node-red-node-pi-gpio, das wir wie schon beschrieben über die Paletten-Verwaltung installieren (Bild 6).

Einige der vorhergehenden bzw. folgenden Vorgehensweisen haben wir schon in Teil 1 [3] beschrieben und führen diese hier nur noch verkürzt auf.

Als Nächstes ziehen wir zwei inject-Nodes in das Editor-Feld und geben in dem einen als Payload eine 1 (String az) ein und benennen ihn mit ON, der andere bekommt eine Payload mit 0 und als Name OFF. Als Nächstes ziehen wir einen rpi-gpio-out-Knoten in den Editor. Klicken wir diesen Knoten zur Konfiguration an, finden wir ein Pinbelegungsschema. Dort aktivieren wir GPIO 4, geben als Type einen Digital Output an, aktivieren „Initialise pin state", setzen diesen auf low (0) und geben dem Knoten den Namen LED GPIO 4 (Bild 7).

Das Initialisieren des Pin-States ist in diesem Fall optional, gehört aber zum guten Ton des Programmierens, um sauber definierte Ausgangslevel zu haben. Hat man z. B. einen Motor o. Ä. (über eine Treiberstufe) an einem Ausgangs-Pin angeschlossen, soll dieser beim Starten ja auch nicht undefinierbar anlaufen, was u. U. eine Gefahr bedeuten kann.

Wir verbinden beide inject-Knoten mit dem rpi-gpio-out-Knoten und deployen den Flow (Bild 8). Klicken wir nun auf den ON- bzw. OFF-inject-node, geht die LED entsprechend an (Bild 5) bzw. aus.

Die Flows aus diesem Teil finden Sie zum Download unter [5].

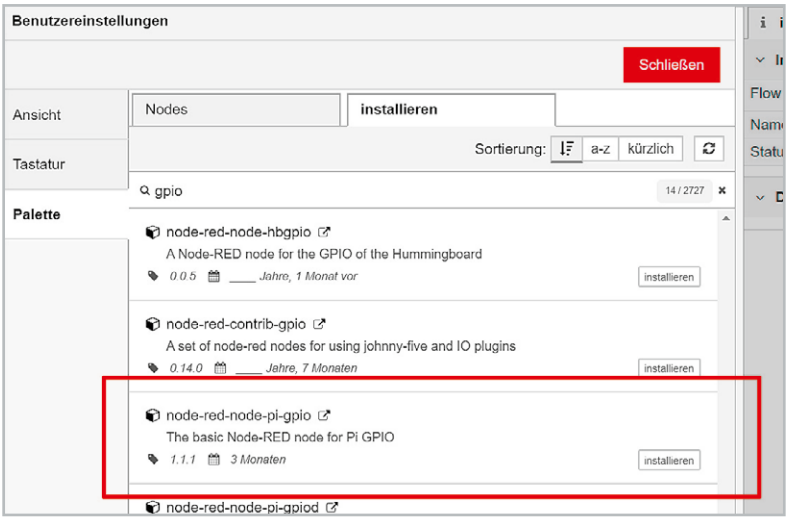


Bild 6: Installation von node-red-node-pi-gpio für die Steuerung der GPIOs

Über die GPIOs des Raspberry Pi kann man natürlich nicht nur LEDs an- und ausschalten, sondern diese mit anderen Knoten bzw. Flows verknüpfen und ereignisgesteuert ansprechen – z. B. wenn die Türklingel betätigt wurde und dies zusätzlich mit einem optischen Signal angezeigt werden soll. Die Möglichkeiten sind aber auch hier durch die Kombinationsmöglichkeiten mit anderen Nodes oder Ereignissen unheimlich vielfältig.

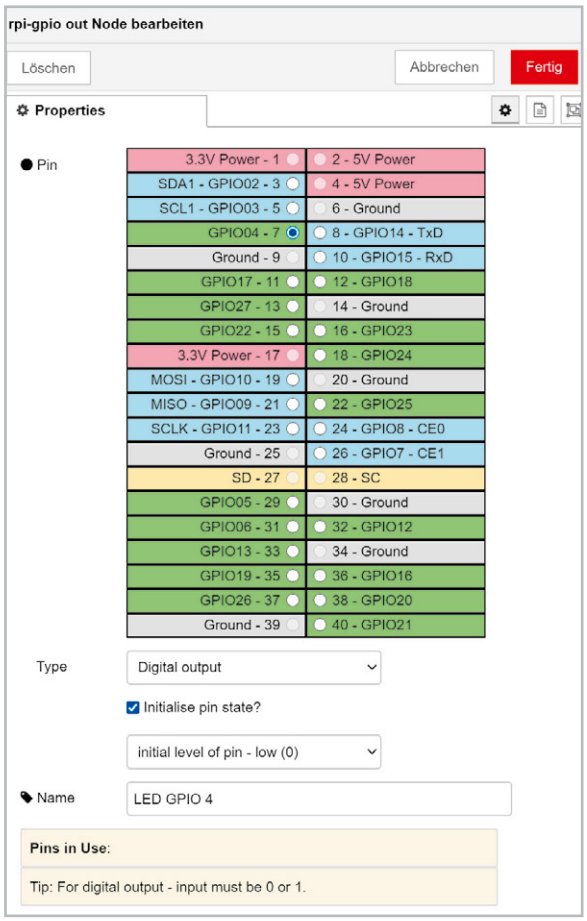


Bild 7: Konfiguration des rpi-gpio-out-Knotens

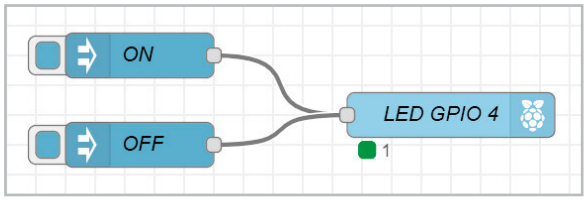


Bild 8: Flow zum Steuern einer LED über die GPIOs des Raspberry Pi

Als Type kann man neben einem digitalen Output den Ausgang auch per PWM mit einem Duty-Cycle von 0 bis 100 % ansteuern. Für zeitkritischere Ansteuerungen lohnt sich ein Blick auf node-red-node-pigpiod, da dies vom Timing genauer ist und sich besser beispielsweise für den Antrieb von Servos eignet.

Neben digitalen Ausgängen ist die Ansteuerung von I²C-Bausteinen per Node-RED ein weiteres Beispiel, um die Funktionalitäten des Raspberry Pi mit(auf ihm nicht vorhandener) Hardware zu ergänzen. Auf diese Weise kann man beispielsweise LCDs, Real-Time-Clocks (RTC) oder ADC (Analog-to-Digital-Converter) verbinden. Im Folgenden zeigen wir den Anschluss eines I²C-Temperatur- und Feuchtesensors SHT20 aus unserem Prototypenadapter-Set PAD4 [1] mithilfe des node-red-contrib-i2c-Pakets.

Tipp: Für einige Bauteile gibt es spezielle Module – hier lohnt sich die Suche in der Node-RED-Paletten-Verwaltung oder der Übersicht der Node-RED-Website zu Paketen und Flows unter [6]. Dort kann man durch Sortierung anhand der Menge der Downloads als zusätzliche Information die Beliebtheit der Pakete erkennen.

Node-RED spricht I²C

Vor dem Anschließen des SHT20 aktivieren wir per

```
sudo raspi-config
```

in einem Terminal unter 5 – Interfacing Options → P5 I²C noch den ARM-I²C-Treiber. Dann stoppen wir Node-RED und fahren den Raspberry Pi sauber herunter.

An die Spannungsschiene des Breadboards schließen wir 3,3 V und Ground des Raspberry Pi an – so können wir eine zusätzliche, optionale Prototypenadapter-LED (s. o.) zur Signalisierung der vorhandenen Spannungsversorgung einbauen. Den GPIO 2 (SDA) und GPIO 3 (SCL) verbinden wir mit den entsprechenden Pins des SHT20. Zusätzlich bekommt der SHT20 noch die 3,3 V des Raspberry Pi als Spannungsversorgung (Bild 9) und eine Verbindung mit Ground.

Da der Raspberry Pi an den beiden I²C-Anschlüssen bereits 1,8-kΩ-Pull-up-Widerstände hat, müssen diese nicht extern beschaltet werden.

Zur vereinfachten Ansteuerung von I²C-Bauteilen gibt es das schon erwähnte Paket node-red-contrib-i2c, das wir über die Paletten-Verwaltung installieren.

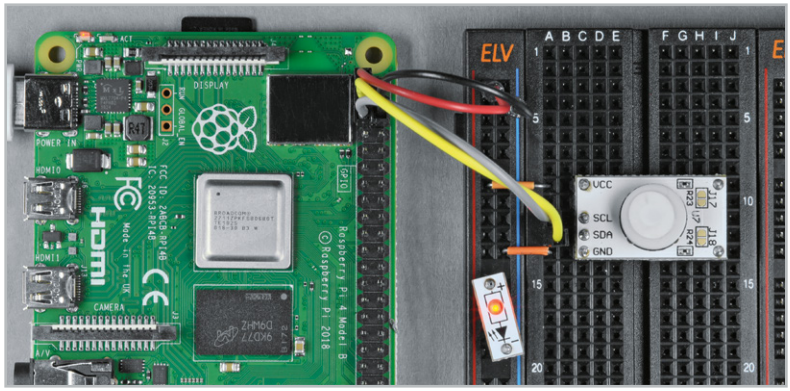


Bild 9: Anschluss des SHT20 mit optionaler LED zur Anzeige der Spannungsversorgung

Danach stehen uns drei neue Nodes zur Verfügung: i2c scan, i2c in und i2c out.

Um Messungen anzustoßen bzw. Werte aus den Registern eines I²C-Bauteils auszulesen oder zu setzen, muss man zum einen das I²C-Bauteil mit seiner Geräteadresse ansprechen und zudem einen Sendebefehl an eine bestimmte Adresse schicken, der beispielsweise die Rückgabe eines gemessenen Werts auslöst. Wir stellen diese Vorgänge auf dem I²C-Kommunikationsbus hier nur sehr verkürzt dar – wer sich tiefergehend einlesen will, dem sei der Beitrag aus dem ELVjournal unter [7] empfohlen.

Zum Senden eines sogenannten Write-Befehls nutzen wir den i2c-in-Node, den wir in den Editor ziehen. Zum manuellen Anstoßen des Prozesses nehmen wir wieder einen inject-Node; den Vorgang können wir aber später natürlich mithilfe anderer Nodes automatisieren.

Doch wie findet man nun die Adressen zum einen für das I²C-Bauteil an sich und zum anderen für das Auslesen von Werten (in unserem Beispiel der Temperatur) aus den Registern des Bauteils? Dazu lohnt sich ein Blick in das Datenblatt des SHT20, das wir unter [8] finden (Bild 10).

Die Adresse für einen Write-Befehl besteht aus der 7-bit-I²C-Geräte-Adresse (binär 0b 1000 000) und der „Richtung" – in unserem Fall eine 0. Binär 1000 000 ergibt hexadezimal 0x40 oder dezimal 64. Da in Node-RED die Adressen dezimal angegeben werden, merken wir uns diesen Wert. Nun brauchen wir noch die Adresse (Befehl) für das Auslesen des Temperaturwerts, die wir im Datenblatt in Tabelle 6 finden (Bild 11). Binär 0b 1110 0011 bedeutet 227 dezimal. Auch diesen Wert notieren wir uns.

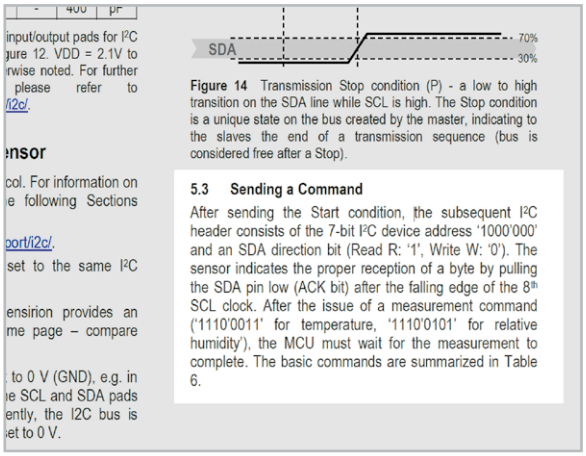


Bild 10: Datenblatt des SHT20 mit der I²C-Adresse (device address)

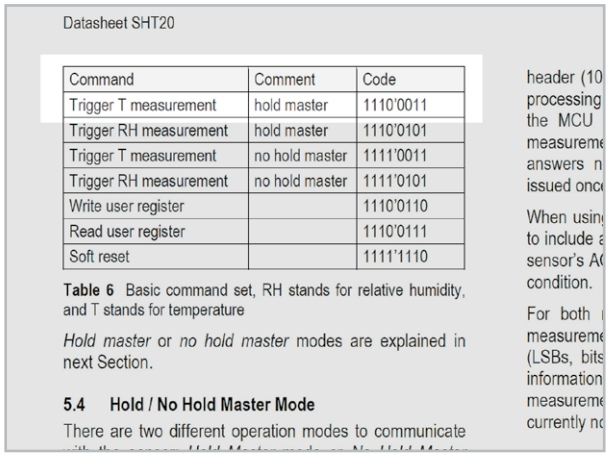


Bild 11: Befehlssatz des SHT20

Hat man eine Temperaturmessung angefordert, schickt in unserem Fall der SHT20 zwei Bytes in der Folge MSB, LSB (Most Significant Byte, Least Significant Byte) zurück. In Bild 12 sieht man das Beispiel aus dem Datenblatt für eine Messung der Luftfeuchte. Unser Beispiel mit der Temperatur kann man entsprechend anwenden. Für unser Beispiel ist zunächst nur die Anzahl der Bytes – 2 – wichtig.

Kommunikation per I²C

Wir öffnen nun unseren i2c-in-Node, setzen die entsprechenden Werte (Bild 13) ein und geben dem Node einen Namen. An den i2c-in-Node hängen wir vorerst einen debug-Node, verbinden alle Knoten und deployen den Flow. Stoßen wir nun per inject-Node eine Messung an, bekommen wir als Ergebnis eine Ausgabe mit zwei Werten – unseren MSB und LSB der Temperaturmessung (Bild 14). Im speziellen Fall die Werte [106, 176].

Tipp

1. Hat man Schwierigkeiten, I²C-Bauteile einzubinden bzw. zu nutzen, empfiehlt sich die Installation von i2c-Tools unter Linux (auf der Kommandozeile mit sudo apt install i2c-tools). Damit kann man sich beispielsweise alle (richtig) verbundenen Bauteile bzw. deren Adressen anzeigen lassen. Fehlt hier die entsprechende Adresse – in unserem Fall für den SHT20 die hexadezimale Adresse 0x40 –, so liegen u. U. Verbindungsprobleme vor.
2. Will das I²C-Bauteil partout nicht antworten, empfiehlt es sich, zuerst die Versorgungsspannung am Bauteil (SHT20: 3,3 V) nachzumessen. Da der Raspberry Pi intern Pull-up-Widerstände hat, sollten diese nicht extern beschaltet sein. Außerdem lohnt sich immer ein Blick auf die richtige Verkabelung zwischen Raspberry Pi und I²C-Bauteil hinsichtlich der SDA- und SCL-Anschlüsse.
3. Man kann es nicht oft genug betonen – viele Raspberry Pis haben aufgrund von Kurzschlüssen oder Verkabelungsfehlern an der 40-poligen Verbindungsleiste vorzeitig das Zeitliche gesegnet. Herunterfahren – verkabeln – kontrollieren – starten, das sollte der normale Ablauf bei Anschluss von Hardware an der doppelten Pin-Anschlussleiste sein.

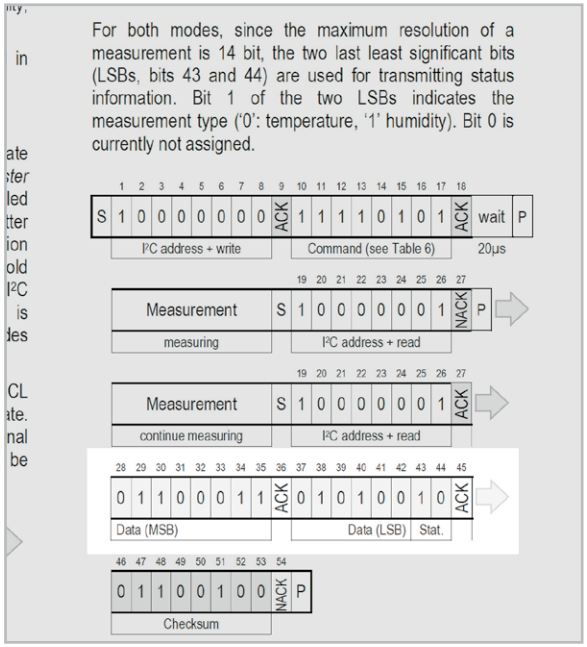


Bild 12: Sequenz einer Messung

Was jetzt noch fehlt, ist die Berechnung der Temperatur anhand der Ausgabe (S_T, s. u.) des i2c-in-Node. Zunächst müssen wir die zwei Bytes zusammenrechnen. Das erste Byte hat den Wert 106 und muss als MSB mit 256 multipliziert werden, da der gesamte Wert 16 Bit hat und die Bits des MSB daher um 8 Bit nach links verschoben werden müssen. Wir erhalten als Ergebnis:

104 * 256 = 26.624

Dazu addieren wir das LSB. Wir vernachlässigen der Einfachheit halber das Bitshifting der beiden Status-Bits wie im Datenblatt beschrieben (Bild 12 oben):

S_T = 26.624 + 176 = 26.800

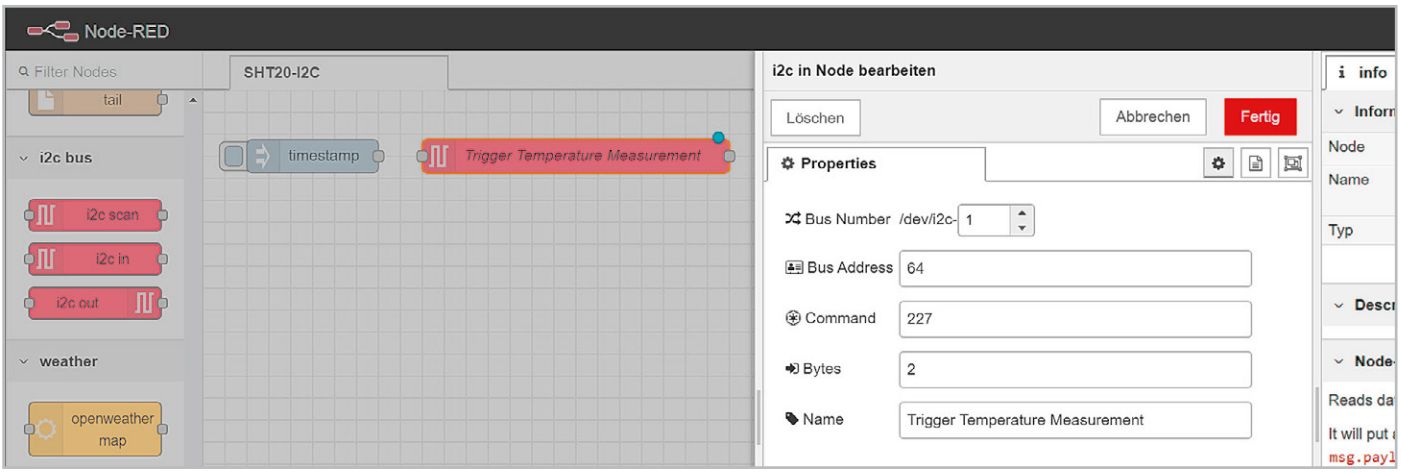


Bild 13: Konfiguration des i2c-in-Node

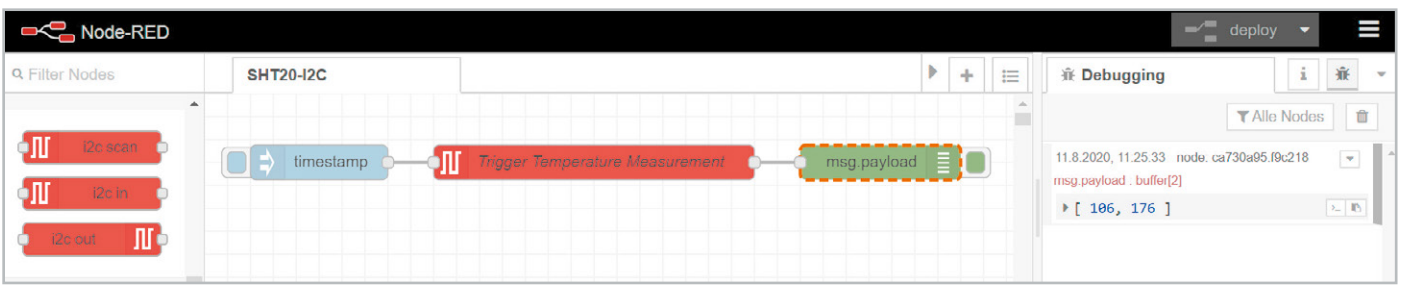


Bild 14: Ausgabe der Temperaturmessung

Im Datenblatt findet man die Formel für die Berechnung der Temperatur (Bild 15). Wir setzen den Wert ein:

T = -46.85 + 175.72 * ST/ 216
T = -46.85 + 175.72 * 26.800/ 65536
T = 25.00

Der Temperaturwert entspricht dem warmen Augusttag im Homeoffice, und ein Blick auf andere Thermometer bestätigt den Wert. Unsere Messung ist damit abgeschlossen.

Zuletzt wollen wir diese Berechnung noch in Node-RED einbauen. Auch hier sind wir programmierfaul und nutzen das Paket node-red-contrib-buffer-parser, das beispielsweise sehr einfach gesendete Werte bearbeiten und zerlegen kann.

Wir ziehen den neu vorhanden buffer-parser-Node in den Editor und verändern die Einstellung bei Output auf key/value und den Namen des Keys von item1 auf temperature. Dann hängen wir einen debug-Node an, verbinden alles, deployen den Flow und lösen wieder eine Messung aus (Bild 16). Wir haben damit nicht nur automatisch den 2-Byte-Wert berechnet, sondern gleichzeitig auch als im JSON-Format vorliegendes Key-/Value-Pair zur einfachen Weiterverarbeitung erzeugt.

Als letzter Schrift fehlt noch die Umrechnung der Temperatur mithilfe des ausgegebenen Wertes. Dazu müssen wir einen function-Node nutzen und tatsächlich – wenn auch nur wenige Zeilen – programmieren:

```
var temperature = msg.payload.temperature;
var tempConversion = -46.85+(175.72*(temperature/65536));

msg.payload = {"temperature": tempConversion};

return msg;
```

Die Programmierung des function-Node und die Ausgabe der berechneten Temperatur sieht dann aus wie in Bild 17.

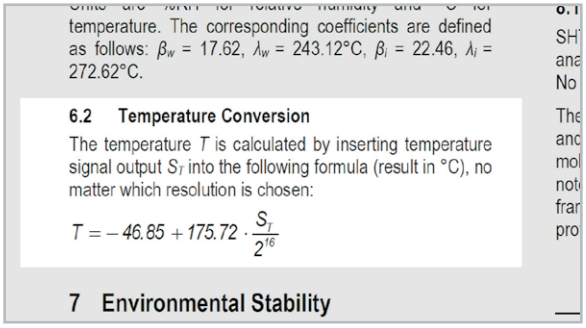
Es gibt zahlreiche Bauteile, die man an den I²C-Bus des Raspberry Pi anschließen kann, daher dient unser Beispiel nur als Anregung. Den Flow kann man unter [5] herunterladen.

Arduino an Raspberry Pi an Node-RED

Nicht immer will oder kann man Hardware direkt an den Raspberry Pi anschließen. Beispielsweise wenn das Ansteuern zu kompliziert bzw. bei einigen zeitkritischen Anwendungen überhaupt nicht möglich ist.

Hier bietet sich u. a. ein Arduino-Mikrocontroller-Board an [9], das günstig ist und durch eine große Community viele Anwendungsbeispiele bereits fertig ausgearbeitet vorliegen hat.

Zunächst installieren wir das Paket node-red-node-arduino über die Paletten-Verwaltung auf dem Raspberry Pi, das mit einem Arduino auf dem Firmata – eine Firmware zur Kommunikation z. B. mit Node-RED als Host – installiert ist, über die serielle Schnittstelle kommunizieren kann. Da diese Verbindung über die USB-Schnittstelle realisiert wird, müssen wir diesmal nicht den Raspberry Pi herunterfahren.


temperature. The corresponding coefficients are defined as follows: β_w = 17.62, λ_w = 243.12°C, β_i = 22.46, λ_i = 272.62°C.

6.2 Temperature Conversion

The temperature T is calculated by inserting temperature signal output S_T into the following formula (result in °C), no matter which resolution is chosen:

$$T = -46.85 + 175.72 \cdot \frac{S_T}{2^{16}}$$

7 Environmental Stability

Bild 15: Formel zur Berechnung der Temperatur

Danach flashen wir die Firmata-Firmware auf den Arduino – wir benutzen dazu die Arduino IDE. Firmata finden wir unter Datei → Beispiele → Firmata → Standard Firmata. Nach dem Hochladen schließen wir den Arduino an unseren Raspberry Pi, auf dem das Node-RED läuft, über die USB-Schnittstelle an.

Als ersten Flow bauen wir das bekannte Blink-Beispiel auf dem Arduino in Node-RED nach. Dazu benötigen wir einen inject- einen trigger- und einen arduino-out-Node, die wir alle in den Editor-Bereich ziehen. Den inject-Node konfigurieren wir mit einem Intervall von 1 s, den trigger-Node mit

Senden 09 1
warten auf
1 Sekunden
dann senden 09 0

In der Kombination mit dem inject-Node haben wir so das typische Blink-Beispiel nachgebaut. Als Letztes müssen wir noch den an der USB-Schnittstelle angeschlossenen Arduino dem Node-RED bekannt machen. Dazu öffnen wir mit einem Doppelklick auf den arduino-out-Node das Konfigurationsmenü und können im Drop-down-Menü bzw. mit Klick auf das Stift-Symbol die Schnittstelle auswählen, beim Type Digital (0/1) einstellen, den Pin 13 bestimmen und einen sprechenden Namen wie Pin 13 LED eingeben (Bild 18).

Danach verbinden wir die Knoten, deployen den Flow, und der Arduino müsste die Onboard-LED an Pin 13 im Sekundentakt zwischen an und aus wechseln. Auch dieser Flow findet sich im Downloadbereich [5].

Mit den arduino-in- und arduino-out-Nodes lassen sich

Digital – 0 oder 1
Analog – 0 bis 255
Servo – 0 bis 180

Pins setzen und auslesen.

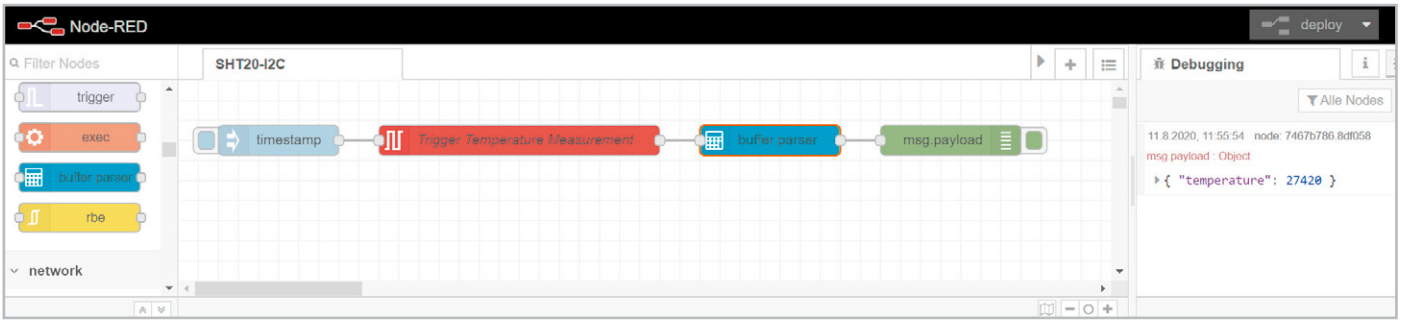


Bild 16: Der Ausgabewert wird mit der buffer-parser-Node automatisch umgerechnet.

Eine weitere Möglichkeit, mit einem Arduino bzw. mit an ihm angeschlossener Hardware zu kommunizieren, besteht über die serielle Schnittstelle mithilfe der serial-in- und serial-out-Nodes (Paket node-red-node-serialport). Wie das funktioniert, haben wir ausführlich in dem Beitrag „Dashboard für Feinstaubmessungen – Anzeige von Umweltdaten mit dem Raspberry Pi und Node-RED" beschrieben, der kostenlos unter [10] herunterzuladen ist. Damit kann man beispielsweise den oben verwendeten Temperatur- und Feuchtesensor SHT20 mit den entsprechenden Arduino-Bibliotheken auslesen und die Daten über die serielle Schnittstelle an Node-RED senden.

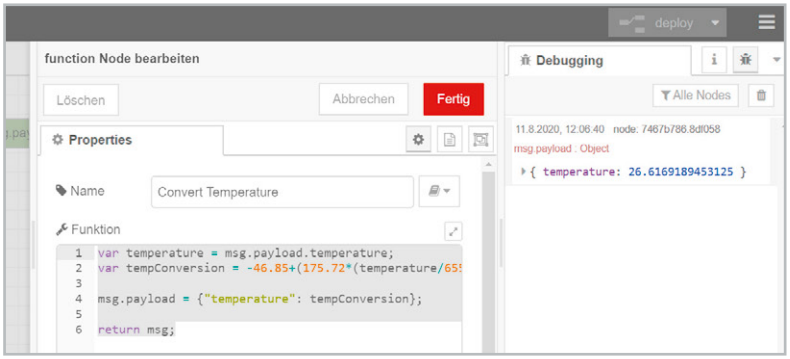


Bild 17: Programmierung des function-Node und Ausgabe des Temperaturwertes

Node-RED + ESP32 = IoT

War bis jetzt die Hardware direkt verbunden, werden wir sie nun drahtlos mit Node-RED verknüpfen. Dazu können beispielsweise die beliebten Espressif-ESP8266- bzw. ESP32-Module [11] verwendet werden. Für unser Beispiel nutzen wir die JOY-iT-Entwicklungsplatine NodeMCU mit ESP32 und den JOY-iT-Umgebungssensor BME680 für Raspberry Pi und Arduino [12] (Bild 19).

Der BME680 erfasst:

- Temperaturen (-40 bis +85 °C)
- Luftfeuchte (0–100 %)
- Luftdruck (300–1000 hPa)
- Gase (IAQ-Index 0–500)

und wird uns als kleine, per Funk angebundene Umweltstation dienen, die ihre Werte an Node-RED sendet. Die Werte können von dort an weitere Node-RED angebundene Geräte oder Dienste übermittelt werden.

Auf dem ESP32 nutzen wir neben der Stromversorgung mit 3,3 V und Ground die Pins D22 (SCL) und D21 (SDA) (Bild 20) und verbinden diese mit den entsprechenden Anschlüssen am BME680.

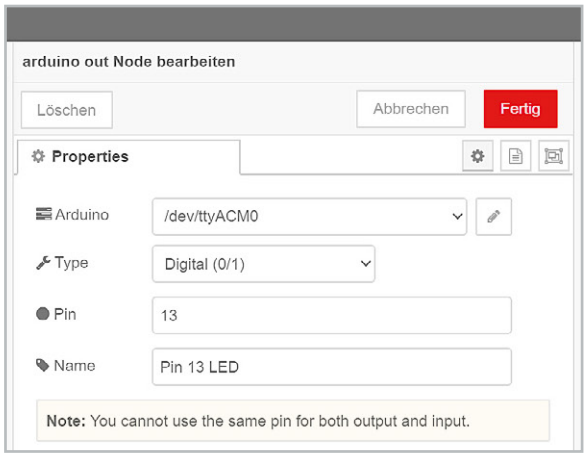


Bild 18: Konfiguration des arduino-out-Node

Firmware für den ESP32

In der Arduino IDE müssen wir als Vorbereitung für die Verwendung der Firmware mit dem BME680 über die Bibliotheksverwaltung die beiden Bibliotheken

Adafruit BME680 Library und

Adafruit Unified Sensor Library

in der neuesten Version einbinden und danach die Arduino IDE neu starten.

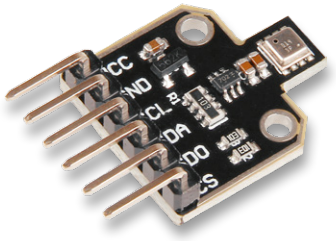


Bild 19: Umweltsensor Bosch BME680 auf einem Breakout-Board

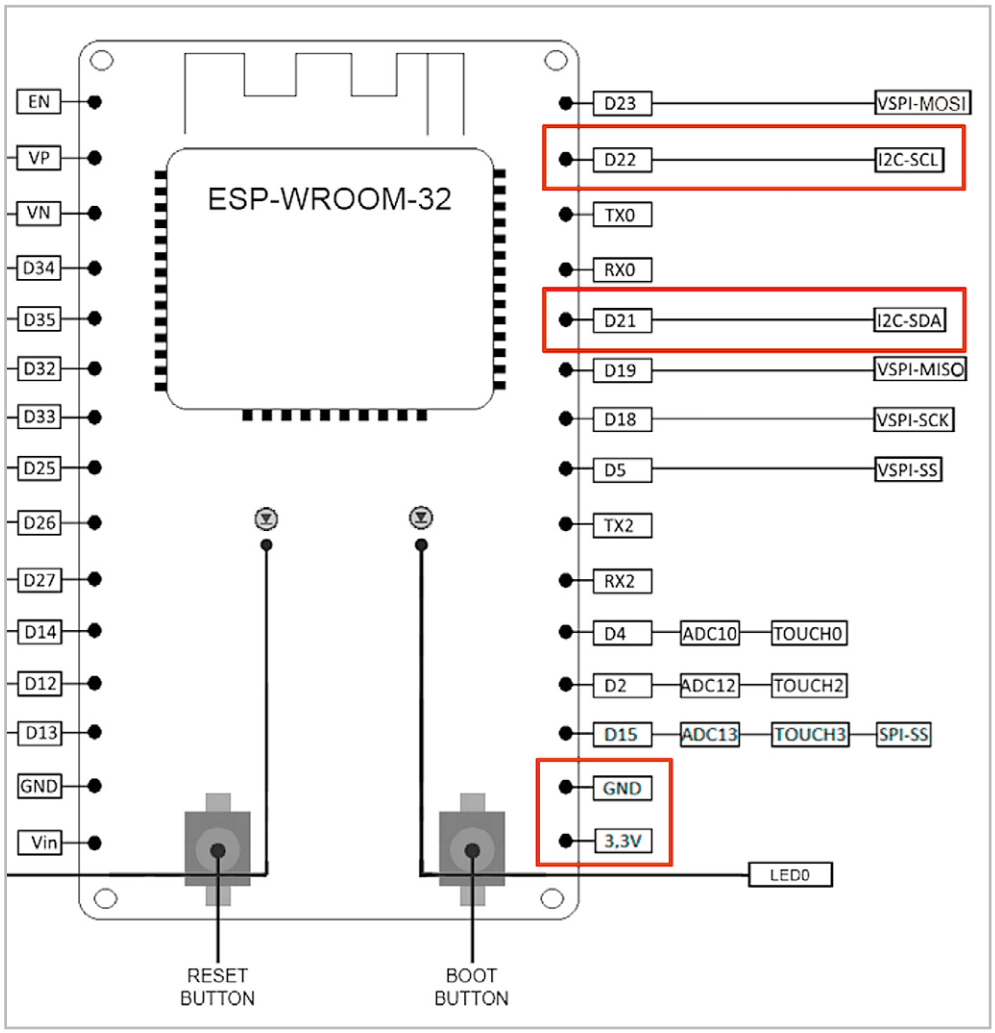


Bild 20: Anschlusspins der JOY-iT-Entwicklungsplatine NodeMCU mit ESP32

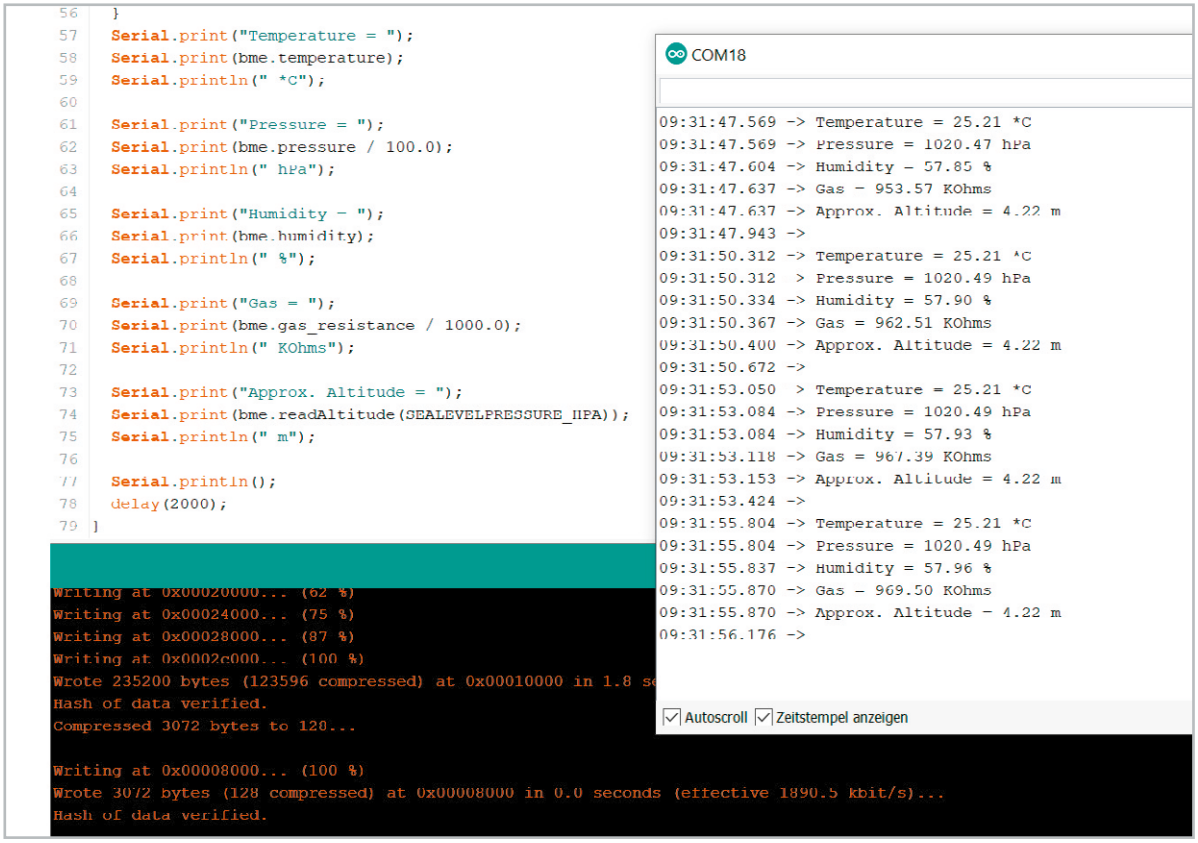


Bild 21: Ausgabe der Werte des BME680 im seriellen Monitor

Zum Testen des BME680 flashen wir den Sketch bme680test, den wir unter Datei → Beispiele → Adafruit BME680 Library finden, auf unseren ESP32. Im seriellen Monitor sollten wir nun Ausgabe der Werte für Temperatur, Luftfeuchtigkeit, Luftdruck und Gase finden (Bild 21). Um die Daten nun an den Raspberry Pi zu senden, auf dem unser Node-RED läuft, müssen wir grundlegend zwei Dinge tun:

1. Aufsetzen und Einbinden des ESP32 in das WLAN
2. Senden der Daten an einen in Node-RED aufgesetzten (Web-)Server

Wir nehmen den bereits oben verwendeten bme680test-Sketch und modifizieren ihn zunächst so, dass sich der ESP32 später in das WLAN verbinden kann:

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
#include <WiFi.h>          ❶

#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10

#define SEALEVELPRESSURE_HPA (1013.25)

const char* ssid     = "SSID"; // Eintragen der SSID
const char* password = "Passwort"; // Passwort
const char* host = "IP_RaspberryPi_Node_RED"; // Hier wird die IP vom Raspberry Pi ❷
eingetragen, auf dem das Node-RED läuft

Adafruit_BME680 bme; // I2C
//Adafruit_BME680 bme(BME_CS); // hardware SPI
//Adafruit_BME680 bme(BME_CS, BME_MOSI, BME_MISO,  BME_SCK);

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println(F(„BME680 test"));
```

```
  if (!bme.begin()) {
    Serial.println("Could not find a valid BME680 sensor, check wiring!");
    while (1);
  }

  // Set up oversampling and filter initialization
  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms

  // WiFi Start
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }

}
```

❸

Die im Code grau umrandeten Bereiche (mit ❶, ❷ und ❸ gekennzeichnet) müssen hinzugefügt und die entsprechenden WLAN-Einstellungen (SSID, Passwort) sowie die IP des Raspberry Pi, auf dem das Node-RED läuft, eingetragen werden.

In der Loop-Schleife

```
void loop() {
  if (! bme.performReading()) {
    Serial.println("Failed to perform reading :(");
    return;
  }

  // Aufbau der TCP-Verbindung mit unserem Raspberry Pi
  WiFiClient client;
  const int httpPort = 1880;
  if (!client.connect(host, httpPort)) {

    return;
  }
```

❹

```
  float valTemp = bme.temperature; //Abfrage Wert Temperatur
  float valPressure = bme.pressure / 100.0; //Abfrage Wert Druck
  float valHumidity = bme.humidity; //Abfrage Wert Luftfeuchtigkeit
  float valVOC = bme.gas_resistance / 1000.0; //Abfrage Wert VOC
```

❺

```
  int num = 0; // Variable für die Länge des Content-Length wird zurückgesetzt
  String var = "{\"airQuality\": {\"temperature\": \"" + String(valTemp) +
"\", \"pressure\" : \"" + String(valPressure) + "\", \"humidity\": \"" +
String(valHumidity) + "\", \"voc\": \"" + String(valVOC) +  "\"}}";
```

❻

```
  num = var.length(); // Variable für Content.length

  // Senden des http POST zum RaspberryPi
  client.println("POST /BME680 HTTP/1.1");
  client.println("User-Agent: ESP32_BME680");
  client.println("Host: „ + String(host));
  client.println("Content-Type: application/json");
  client.println("Content-Length: " + String(num));
  client.println();
  client.print(var);
  client.println();
  delay(10);

  delay(60000);
}
```

❼

entfernen wir alle Codes aus dem bme680test-Sketch bis auf das Error-Handling mit einer Information auf der seriellen Schnittstelle für einen nicht funktionierenden Sensor.

In ❹ definieren wir zunächst die Verbindung mit unserem Raspberry Pi. Wir benutzen dazu den Port 1880 von Node-RED.

Die Werte für Temperatur, Druck, Luftfeuchtigkeit und Gase werden in ❺ ausgelesen und Variablen zugeordnet. Diese werden zur einfacheren Verarbeitung im JSON-Format verschickt ❻. Dazu setzen wir einen String zusammen, dessen Länge für die spätere Versendung per http bestimmt werden muss. Daher wird die entsprechende Variable für die Länge zurückgesetzt, der String gebildet und am Ende die Länge des Strings errechnet.

In ❼ schicken wir diesen String per http schließlich an den (Web)Server in Node-RED, der die Daten entgegennimmt und weiterverarbeiten kann. Das Intervall zur Sendung von Daten legen wir der Einfachheit halber mit der delay()-Funktion und einer Verzögerung von 60 s fest.

Tipp: Prinzipiell gibt es verschiedene Arten, Daten zu senden. Möglich wäre auch die Nutzung von MQTT, die wir in der nächsten Folge noch ansprechen werden.

Wir wechseln nun zu Node-RED und ziehen einen http-in-Node als HTTP-Endpunkt für den Empfang der Daten vom ESP32 in den Editor. Da wir als Ziel-URL im Arduino-Sketch BME680 definiert haben, tragen wir das im Konfigurationsmenü ein. Als Methode wählen wir POST und geben dem Knoten wie immer einen sprechenden Namen. Mit einem debug-Node können wir überprüfen, ob der Empfang der Daten funktioniert (Bild 22). Da wir das Intervall auf 60 s festgelegt haben, kann das natürlich etwas dauern – für die ersten Tests kann man das Intervall auch heruntersetzen.

Da wir die Daten im JSON-Format verschickt haben, können wir in Node-RED sehr einfach auf die einzelnen Werte zugreifen und diese weiterverarbeiten. Folgender Code in einem function-Node liest die empfangenen Daten aus und separiert sie für die weitere Bearbeitung. Zusätzlich geben wir den Daten noch einen Zeitstempel:

```
msg1 =
{
    "payload" : msg.payload.airQuality.temperature
};

msg2 =
{
    "payload" : msg.payload.airQuality.pressure
};

msg3 =
{
    "payload" : msg.payload.airQuality.humidity
};

msg4 =
{
    "payload" : msg.payload.airQuality.voc
};

var now     = new Date();
    var year    = now.getFullYear();
    var month   = now.getMonth()+1;
    var day     = now.getDate();
    var hour    = now.getHours();
    var minute  = now.getMinutes();
    var second  = now.getSeconds();

    msg5 = {};
    msg5.payload = day + ,.' + month + ,.' + year + , - , + hour + ,:'
+ minute + ":" + second;

return [msg1, msg2, msg3, msg4, msg5];
```

Die Daten können nun weiterverarbeitet oder beispielsweise direkt in einem Dashboard angezeigt werden. Von hier aus lassen wir Ihrer Kreativität und den vielen Möglichkeiten, die Node-RED bietet, freien Lauf.

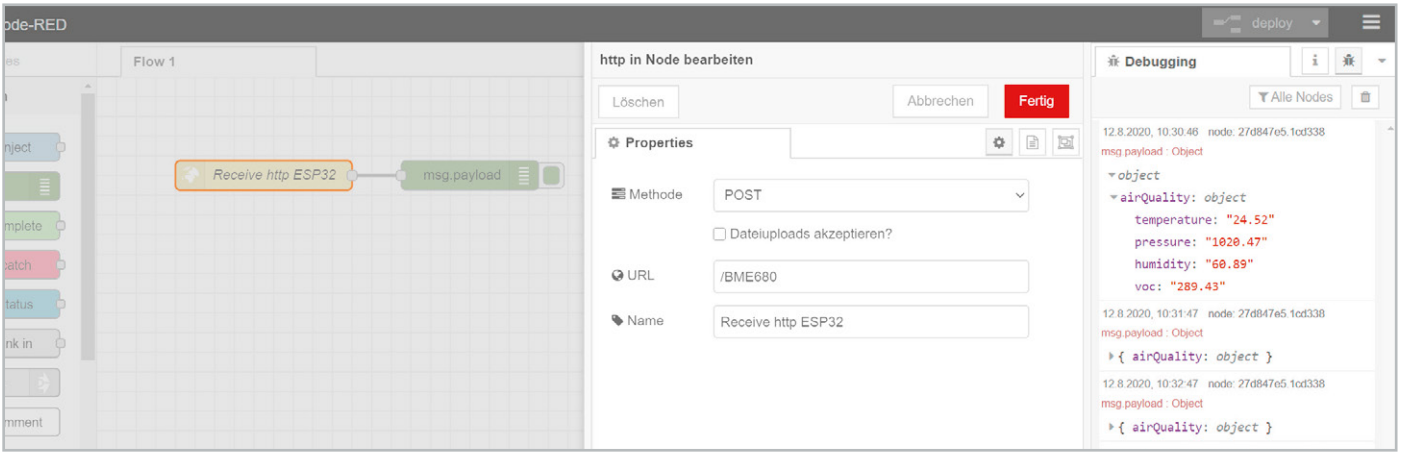


Bild 22: Empfang der vom ESP32 gesendeten Daten in Node-RED

Ausblick

Während wir uns im ersten Teil mit der Installation und den Grundlagen der Node-RED-Programmieroberfläche und in diesem Beitrag mit der Möglichkeit, elektronische Bauteile auf verschiedene Arten anzuschließen, beschäftigt haben, werden wir im dritten und abschließenden Teil einen Blick unter die Haube dieses universellen Prototyping-Tools werfen. Außerdem schauen wir uns weitere, interessante Anwendungsmöglichkeiten an, die beispielsweise eine FRITZ!Box in Node-RED einbinden oder den Datentransport per MQTT. ELV

Weitere Infos:

[1] PAD4: Bestell-Nr. 155107
[2] Raspberry Pi: Stromverbrauch, Spannungen: https://www.raspberrypi.org/documentation/faqs/#pi-power
[3] ELVjournal 4/2020, Programmieren (fast) ohne Code – NODE-Red als universelles Prototyping-Tool, Teil 1: Bestell-Nr. 251410
[4] PAD2: Bestell-Nr. 154712
[5] Download Node-RED Beispiel-Flows: de.elv.com: Webcode #10322
[6] Übersicht der Node-RED-Website zu den Modulen und Flows: https://flows.nodered.org/
[7] ELVjournal 2/2017, Digitale Hardwareschnittstellen, Teil 3: Bestell-Nr. 206581
[8] Datenblatt SHT20: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/2_Humidity_Sensors/Datasheets/Sensirion_Humidity_Sensors_SHT20_Datasheet.pdf
[9] Arduino UNO: Bestell-Nr. 122123 oder Arduino Nano: Bestell-Nr. 250005
[10] ELVjournal 4/2020, Dashboard für Feinstaubmessungen – Anzeige von Umweltdaten mit dem Raspberry Pi und Node-RED: kostenloser Download unter Bestell-Nr. 251411
[11] JOY-iT-Entwicklungsplatine NodeMCU V2 mit ESP8266: Bestell-Nr. 145163
JOY-iT-Entwicklungsplatine NodeMCU mit ESP32: Bestell-Nr. 145164
[12] JOY-iT-Umgebungssensor BME680 für Raspberry Pi und Arduino: Bestell-Nr. 250865

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links