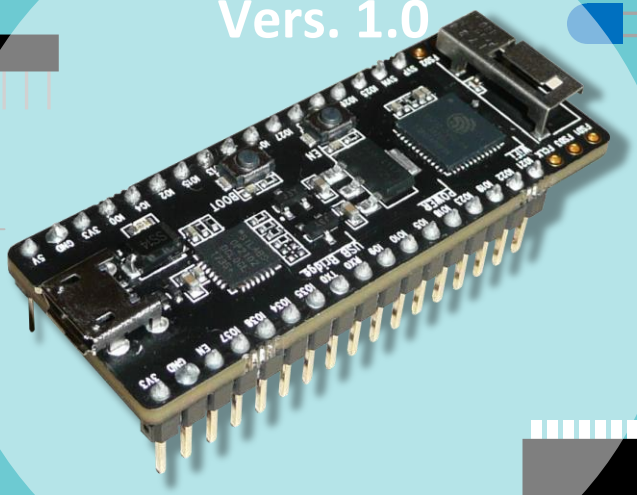
ESP32 Pico-Board

ESP32 Addon

ESP32 und Node-RED Datenversand

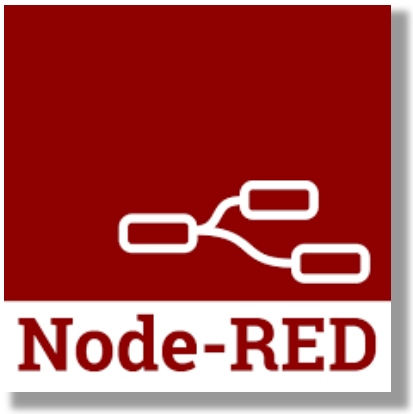
Vers. 1.0



Easy Learning

ESP32 und Node-Daten

Dieses ESP32-Addon baut auf das vorangegangene auf. Es wurden Daten von Node-RED [2] über eine TCP-Verbindung an den ESP32 versendet und dort zur Anzeige gebracht, bzw. ausgewertet. Nun wollen wir in diesem Addon den umgekehrten Weg gehen und Daten vom ESP32 zu Node-RED versenden und dort anzeigen bzw. auswerten. Das geht über eine TCP-Verbindung sehr einfach und ist schnell implementiert. Das Beispiel soll natürlich nur die Möglichkeiten aufzeigen und Lust auf mehr machen. Die Einsatzgebiete sind nahezu unbegrenzt. Detailinformationen zu Node-RED sind in meinem Buch mit dem Titel *IOT-Programmierung mit Node-RED* [3] zu finden.



Der Workflow

Wollen wir uns zu Beginn den Weg der Daten anschauen und das am besten über eine Grafik. Unser Sender ist diesmal das ESP32-Picoboard und der Empfänger Node-RED, das ja als Server auf dem Raspberry Pi installiert ist. Das Übertragungsprotokoll ist wiederum TCP.

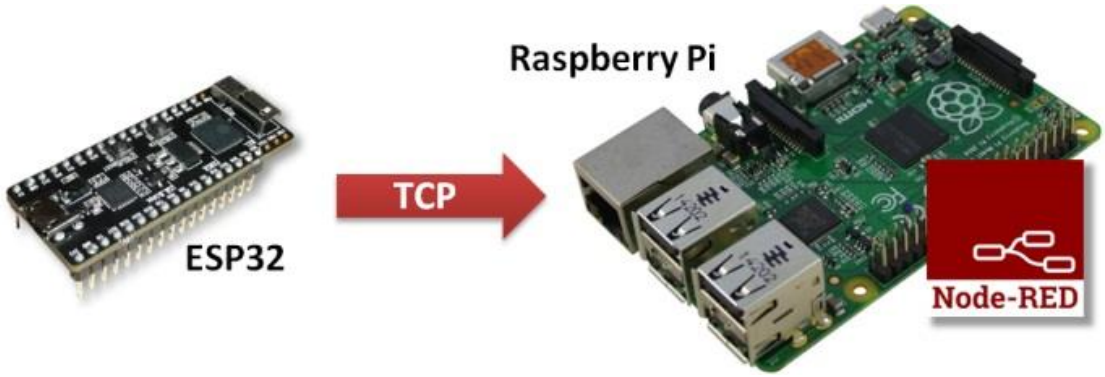


Abbildung 1 Der Weg der Daten

Was wollen wir für unser Beispiel an Daten nutzen, ohne gleich aufwendige Messwerterfassungen zu realisieren? Nun, auf dem ESP32-Pico-Discoveryboard [4] gibt es ein Potentiometer, mit dem wir analoge Spannungen simulieren können. Was würde dagegen sprechen, den erzeugten Spannungswert an einen analogen Eingang des ESP32-Picoboards zu leiten und diesen dann per TCP zu versenden. Die Verkabelung schaut dann wie folgt aus.

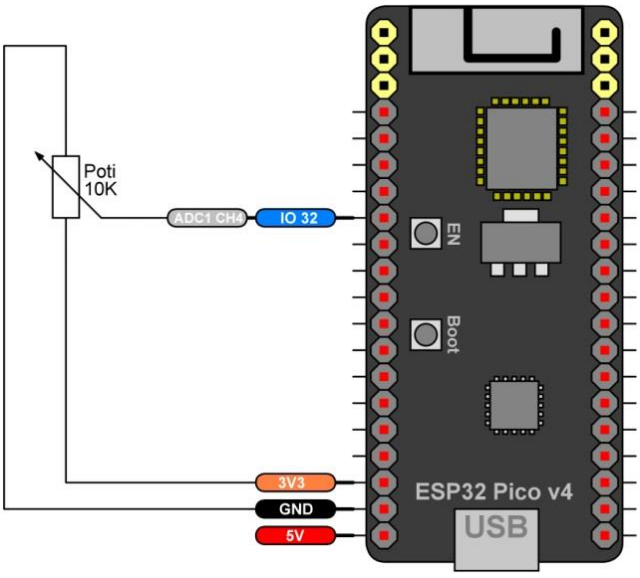


Abbildung 2 Der Anschluss des Potentiometers am ESP32-Picoboard

Zur Programmierung bzw. Abfrage des analogen Eingangs kommen wir natürlich jetzt noch zu sprechen.

Der ESP32-Code

Starten wir mit der Programmierung des Senders, also dem ESP32-Picoboard mit dem Code [5]. Der folgende Abschnitt führt die grundlegenden Deklarationen durch.

```
#include <WiFi.h>
const char* ssid     = "<SSID>";
const char* password = "<Passwort>";

const char* server = "192.168.178.27"; // IP-Adresse des Node-RED-Servers
const int port = 8088; // Port
unsigned long previousMillis = 0; // Letzter Update-Zeitstempel
const long interval = 1000;       // Sende-Intervall in ms
```

Die hier angegebene IP-Adresse muss die des Raspberry Pi sein, auf dem Node-RED läuft. Der nachfolgende Abschnitt innerhalb der *setup*-Funktion ist wieder für die Verbindungsaufnahme zu Wifi zuständig.

```
void setup() {
  Serial.begin(9600);
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected to IP address: ");
  Serial.println(WiFi.localIP());
}
```

Innerhalb der *loop*-Funktion erfolgt die Verbindungsaufnahme via TCP-zum Server, auf dem Node-RED läuft, was im angegebenen Intervall stattfindet. Ich habe keine *delay*-Funktion für das Erzeugen einer Pause genutzt, weil das den kompletten Programmablauf unterbrechen würde. Die Verbindungsaufnahme erfolgt immer nur für den Zeitpunkt des Versendens von Informationen und wird unmittelbar darauf wieder geschlossen.

```
void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    Serial.print("Connecting to ");
    Serial.println(server);
    WiFiClient client; // Für TCP-Connection

    if (!client.connect(server, port)) {
        Serial.println("Connection failed");
        return;
    }
    int analogValue = analogRead(32); // Messwerterfassung
    Serial.println(analogValue);
    client.print(analogValue); // Sende Daten zum Server
    client.stop();             // Schließe TCP-Verbindung
  }
}
```

Über die *analogRead*-Funktion wird der IO-Pin 32 abgefragt, der als ADC1/CH4 arbeitet. Das Versenden des Messwertes und die TCP-Unterbrechung erfolgt über die beiden folgenden Zeilen.

```
...
client.print(analogValue); // Sende Daten zum Server
client.stop();             // Schließe TCP-Verbindung
...
```

Über den Aufruf des Serial-Monitors können wir die Initiierung der Wifi-Verbindung und das Versenden des Potentiometerwertes via TCP-Verbindung verfolgen.

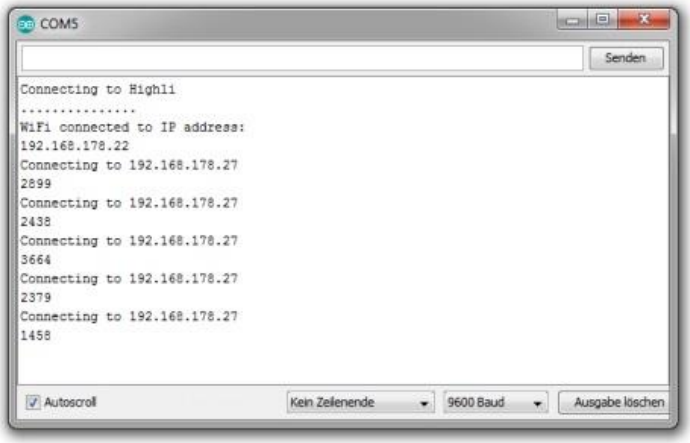


Abbildung 3 Die Statusinformationen im Serial-Monitor

Das einzige, was wir also in Richtung Node-RED versenden, ist der nackte Zahlenwert des Analog/Digital-Wandlers, der an Pin 32 arbeitet. Sehen wir jetzt, was auf der Seite von Node-RED erforderlich ist, um diesen Wert zu empfangen und zu verarbeiten. Wir wollen nämlich den Messwert auch grafisch über ein geeignetes Anzeigeinstrument - eine sogenannte *Gauge* - visualisieren. Dazu ist die Node-RED-Erweiterung *Dashboard* erforderlich. Die notwendige Installation ist im vorangegangenen Addon beschrieben.

Was tut sich in Node-RED

Der von mir erstellte Node-RED-Flow schaut dabei wie folgt aus.

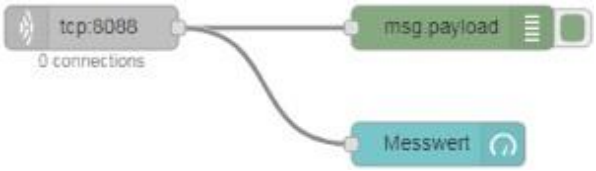


Abbildung 4 Der Node-RED-Flow für das Empfangen und Anzeigen der Daten

Auf der linken Seite ist die TCP-Node zu sehen, die für den Empfang des Messwertes zuständig ist. Diese Node leitet ihre Informationen an zwei weitere Nodes weiter. Da ist zum einen die obere *Debug*-Node mit der Aufgabe, den empfangenen Zahlenwert in einem speziellen Fenster von Node-RED anzuzeigen, zum anderen haben wir die untere *Gauge*-Node, die es uns ermöglicht, den Zahlenwert über ein Zeigerinstrument zu visualisieren. Sehen wir uns das Ganze im Detail an. Die TCP-Node für den Empfang des Zahlenwertes ist in der *Input*-Palette zu finden.

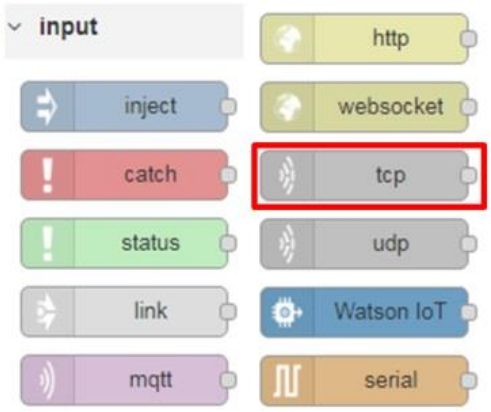


Abbildung 5 Die Input-Palette von Node-RED

Über einen Doppelkick auf die TCP-Node nach dem Herüberziehen auf die Arbeitsfläche erscheinen die Einstellungen.

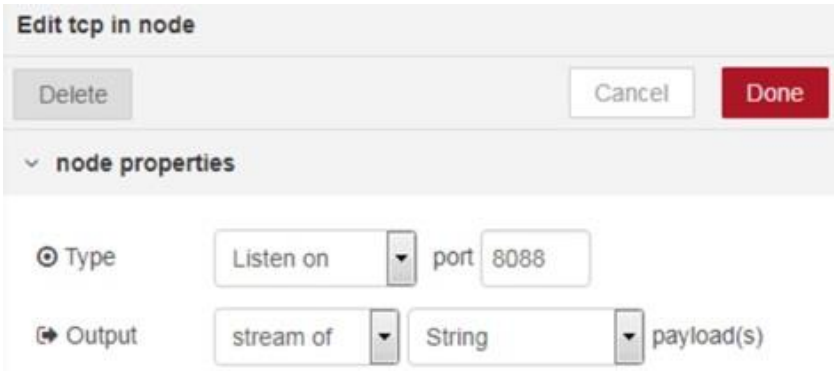


Abbildung 6 Die TCP-Einstellungen

Es ist auffällig, dass hier keine IP-Adresse und nur der erforderliche Port (8088) anzugeben ist. Die IP-Adresse ist durch die des Node-RED-Servers schon gegeben und muss nicht noch einmal explizit angegeben werden. Die *Debug*-Node befindet sich in der *Output*-Palette

Abbildung 7 Die Debug-Node in der Output-Palette

Sie kann Statusinformationen in einem Fenster von Node-RED zur Anzeige bringen und ist u.a. sehr gut für die Fehlersuche zu verwenden. Während des Datenempfangs kann der TCP-Verbindungsstatus sehr gut beobachtet werden. Unterhalb der TCP-Node ist dieser Status gut zu erkennen und wechselt zwischen den beiden hier gezeigten hin und her.

Die Anzeige der empfangenen Daten erfolgt im *Debug*-Reiter von Node-RED, der sich auf der rechten Seite des Node-RED-Fensters befindet.

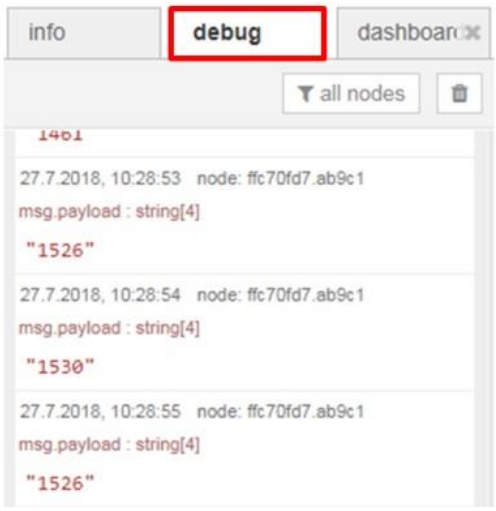


Abbildung 8 Der Debug-Reiter von Node-RED

Hier sind Messwert und Zeitstempel zu erkennen. Die *Gauge*-Node ist natürlich in der *Dashboard*-Palette zu finden.

Abbildung 9 Die Gauge-Node in der Dashboard-Palette

Die Konfiguration der *Gauge*-Node schaut wie folgt aus.

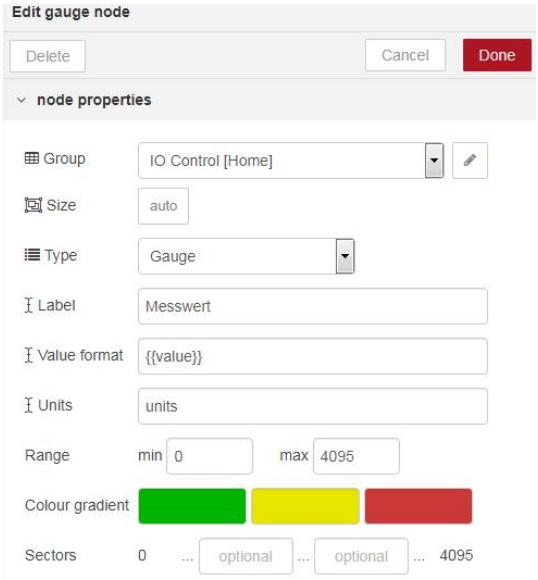


Abbildung 10 Die Konfiguration der Gauge-Node

Damit die *Gauge*-Node „weiß“, in welchen Wertebereich sich die Messwerte befinden, müssen die Grenzen unter *Range* angegeben werden. Der Analog/Digital-Wandler des ESP32-Picoboards liefert Werte von 0 bis 4095.

Um die Gauge im Browser sichtbar zu machen, muss die folgende URL als Adresse eingegeben werden, wobei natürlich die von mir verwendete IP-Adresse des Raspberry PI bzw. des Node-RED-Servers ersetzt werden muss.

```
192.168.178.27:1880/ui
```

Bei mir schaut das dann wie folgt aus.

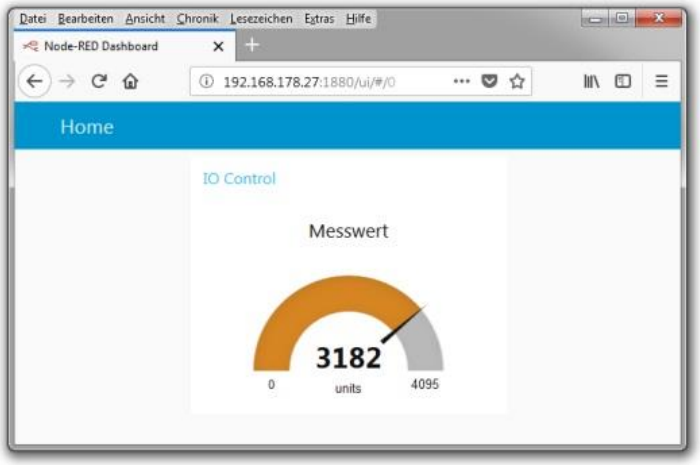


Abbildung 11 Die Gauge mit dem ermittelten Messwert

Wenn jetzt am Potentiometer gedreht wird, ändert sich die Anzeige in der Gauge im Sekundentakt.

Frohes Frickeln

Das ESP32-Praxisbuch

[1] https://www.elektor.de/das-esp32-praxisbuch

[2] https://nodered.org/

[3] https://www.elektor.de/iot-programmierung-mit-node-red

[4] https://www.erik-bartmann.de/?Projekte ESP32 Discovery-Board

[5] https://github.com/erikbartmann/ElektorESP32/tree/master/Addons/NodeREDDatenversand

https://www.erik-bartmann.de/