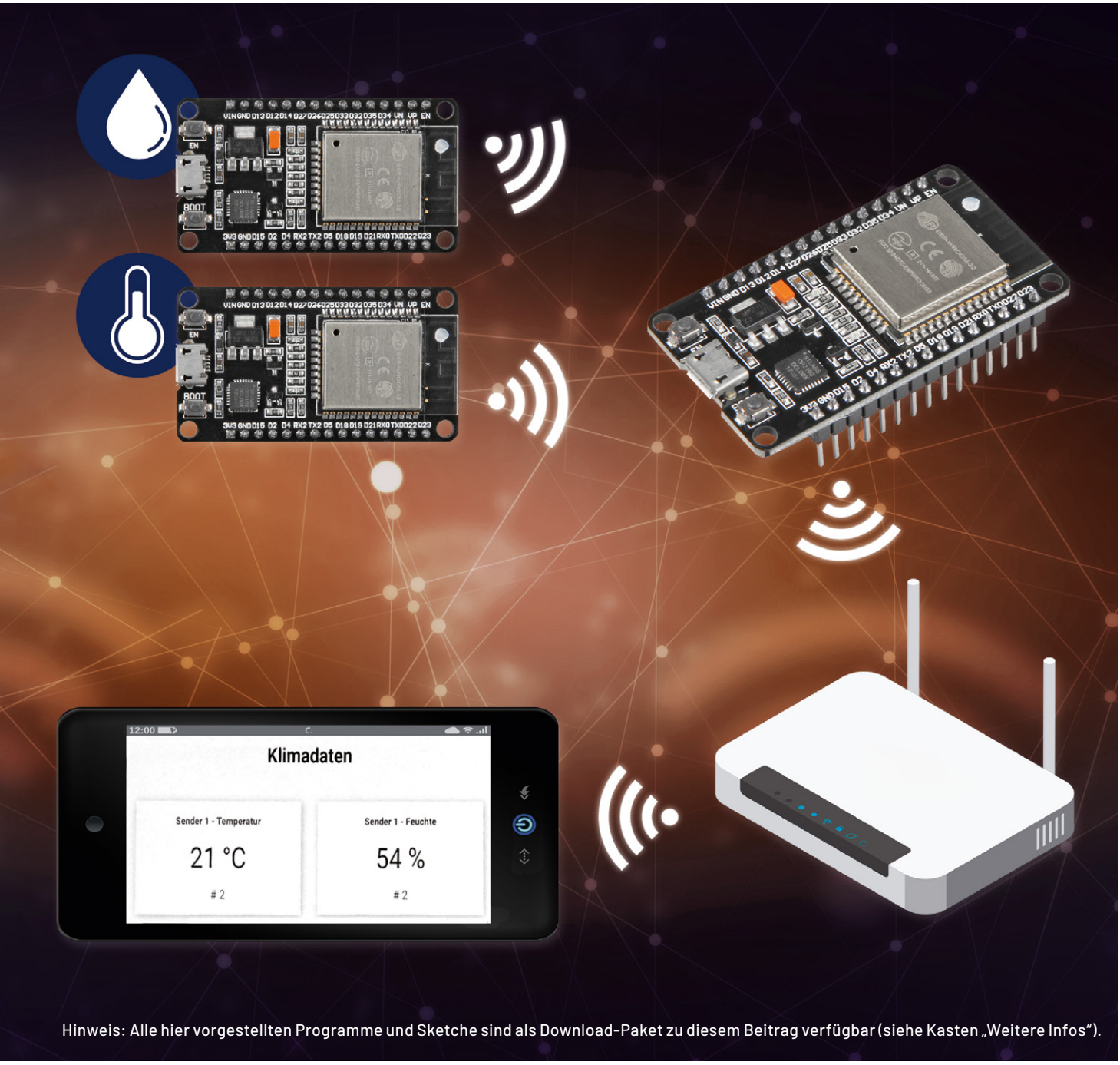
ESP-NOW

Teil 4

Datentransfer ins Internet

In den letzten Beiträgen zu dieser Artikelserie wurden die Grundlagen und Anwendungen des ESP-NOW-Systems ausführlich erläutert und dargestellt. Allerdings erfolgte der drahtlose Datenaustausch immer nur zwischen zwei oder mehreren Boards. Es ist jedoch auch möglich, auf einem ESP32-Controller einen Webserver zu betreiben und gleichzeitig das ESP-NOW-Kommunikationsprotokoll einzusetzen. So kann man etwa mehrere ESP32-Boards verwenden, um Sensormesswerte über ESP-NOW an einen zentralen ESP32-Empfänger zu übermitteln, der dann wiederum alle gesammelten Messwerte auf einem Webserver ablegt. Von dort können die Daten dann via WLAN von jedem Gerät, das über einen Browser verfügt, also von PCs, Laptops, Smartphones oder Tablets etc., abgerufen werden.



Hinweis: Alle hier vorgestellten Programme und Sketche sind als Download-Paket zu diesem Beitrag verfügbar (siehe Kasten „Weitere Infos").

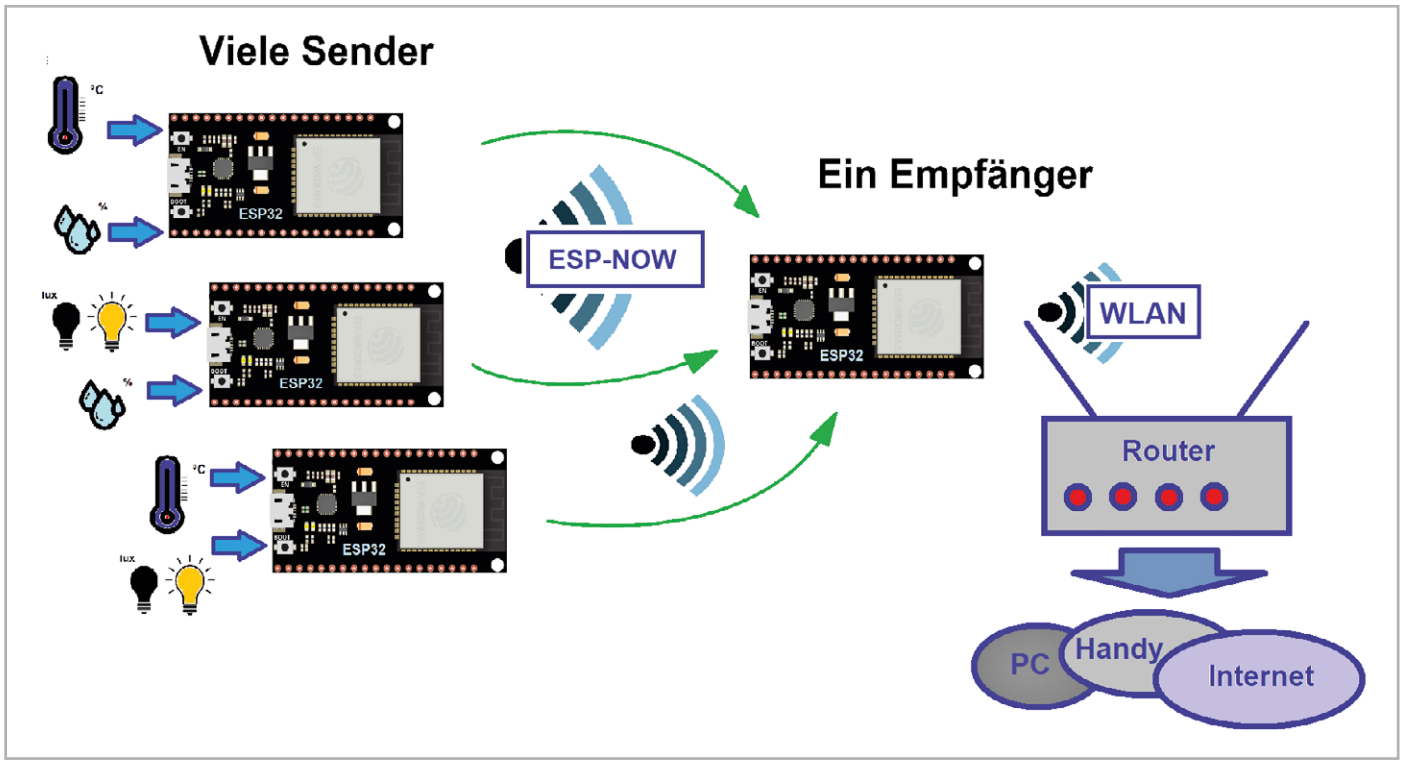


Bild 1: Übersicht zum ESP-NOW-Webserver

ESP32 als Webserver

Für diese Anwendung sind drei ESP32-Boards und zwei SHT20-Temperatur- und Luftfeuchtesensoren erforderlich (s. Materialliste am Ende des Artikels).

Die Daten der Sensoren werden über ESP-NOW an ein zentrales Empfängerboard gesendet. Dieses verarbeitet die Datenpakete und legt die Messwerte auf einem Webserver ab. Der Webserver wird dabei jedes Mal automatisch aktualisiert, wenn er mithilfe von Server-Sent Events (SSE) einen neuen Messwert erhält (Bild 1).

Um den Webserver aufzubauen, müssen die folgenden Bibliotheken installiert werden:

- ESPAsyncWebServer auf
  https://github.com/me-no-dev/ESPAsyncWebServer
- AsyncTCP auf
  https://github.com/me-no-dev/AsyncTCP

Diese Bibliotheken sind nicht über den Arduino Library Manager installierbar, daher müssen die Bibliotheksdateien als ZIP-Ordner vom GitHub-Server in den Ordner „Arduino Installation Libraries" kopiert werden.
Alternativ sind sie in der Arduino IDE einzubinden über

Sketch → Include Library → Add .zip Library

Die ArduinoJson-Bibliothek ist ebenfalls erforderlich. Diese steht jedoch im Library Manager zur Verfügung. Mittels

Sketch → Include Library → Manage Libraries

kann man wie üblich nach dem Bibliotheksnamen suchen und die Lib dann installieren.

Nach dem Installieren der Libs wird das Empfänger-Board vorbereitet. Dazu wird dessen MAC-Adresse in der bekannten Weise ausgelesen und inventarisiert. Details dazu finden sich im ersten Artikel [1] zu dieser Serie.

Damit das ESP32-Empfängerboard die Pakete der Sender empfangen und die Werte auf einen Server zur Verfügung stellen kann, wird der nachfolgende Code auf das Empfängerboard geladen.

Das Programm ist darauf ausgerichtet, die Werte von zwei verschiedenen ESP-Sendern zu empfangen. Da der Code bereits recht umfangreich ist, werden im Folgenden nur die wichtigsten Teile dargestellt.

Der vollständige Sketch findet sich wie üblich im Download-Paket zu diesem Artikel.

```
// ESP-NOW-web-server.ino
// ESP32 € IDE 1.8.16

#include <esp_now.h>
#include <WiFi.h>
#include "ESPAsyncWebServer.h"
#include <Arduino_JSON.h>
#include "my_secs.h"

const char* ssid = SECRET_SSID;
const char* password = SECRET_PASS;

typedef struct struct_message {  int id;  float temp;  float hum;  unsigned int readingId;} struct_message;
struct_message incomingReadings;

JSONVar board;

AsyncWebServer server(80);
AsyncEventSource events("/events");

void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len)
{ char macStr[18];
  Serial.print("Packet received from: ");
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.println(macStr);
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));

  board["id"] = incomingReadings.id;
  board["temperature"] = incomingReadings.temp;
  board["humidity"] = incomingReadings.hum;
  board["readingId"] = String(incomingReadings.readingId);
  String jsonString = JSON.stringify(board);
  events.send(jsonString.c_str(), "new_readings", millis());

  Serial.printf("Board ID %u: %u bytes\n", incomingReadings.id, len);
  Serial.printf("t value: %4.2f \n", incomingReadings.temp);
  Serial.printf("h value: %4.2f \n", incomingReadings.hum);
  Serial.printf("readingID value: %d \n", incomingReadings.readingId);
  Serial.println();
}

...


void loop() {
  static unsigned long lastEventTime = millis();
  static const unsigned long EVENT_INTERVAL_MS = 5000;
  if ((millis() - lastEventTime) > EVENT_INTERVAL_MS) {
    events.send("ping",NULL,millis());
    lastEventTime = millis();
  }
}
```

Für den Zugang zum WLAN muss eine Datei „my_secs.h“ erstellt werden, welche die Zugangsdaten (SSID und Passwort) zum lokalen WLAN enthält:

#define SECRET_SSID „WLANBox xxxx“
#define SECRET_PASS „12345678901234567890“

Diese Daten sollten immer in einer separaten Datei gespeichert werden. Damit wird das Risiko minimiert, dass sie versehentlich mit einem Programm weitergegeben werden. Diese Datei wird über

#include „my_secs.h“

in den Code eingebunden. Im Programm selbst wird dann die JSON Lib benötigt, da eine JSON-Variable mit den von jedem Board empfangenen Daten erstellt wird. Diese JSON-Variable wird verwendet, um alle erforderlichen Informationen an die Webseite zu senden, die später im Projekt aufgebaut wird.

Die empfangenen Daten werden wieder in eine structure („struct_message“) abgelegt. Diese enthält die Board-IDs der Sender, die Temperatur- und Feuchtigkeitsmesswerte des SHT20-Sensors sowie die Messwert-ID. Anschließend wird daraus eine JSON-Variable aufgebaut und ein asynchroner Webserver auf Port 80 gestartet. Um die Informationen automatisch auf dem Webserver anzuzeigen, kommt ein Server-Sent Event (SSE) zum Einsatz. Vom Server gesendete Ereignisse ermöglichen es einer Webseite (dem „Client“), Updates von einem Server zu erhalten. Dies wird hier verwendet, um automatisch neue Messwerte auf der Webserverseite anzuzeigen.

Hinweis: Vom ESP32- Server gesendete Ereignis-Updates werden vom Internet Explorer z. T. nicht unterstützt. Hier kann ein manueller „Refresh" der Seite erforderlich sein, um die Daten zu aktualisieren. Bei alternativen Browsern wie Chrome oder Firefox arbeitet die Aktualisierung dagegen meist problemlos.

Die JSON-String-Variable enthält die von den Sender-Boards übermittelten Informationen:

```
board["id"] = incomingReadings.id;
board["temperature"] = incomingReadings.temp;
board["humidity"] = incomingReadings.hum;
board["readingId"] = String(incomingReadings.readingId);
String jsonString = JSON.stringify(board);
```

Nachdem alle empfangenen Daten in der JSON-String-Variable gesammelt wurden, werden diese Informationen als Ereignis („new_readings") an den Browser übermittelt. Zusätzlich werden alle empfangenen Informationen auf dem Serial Monitor der Arduino IDE ausgegeben. Der ServerCode kann nun auf den ersten ESP32-Controller geladen werden (ESP-NOW-web-server.ino).

Hinweis: Die Zugangsdaten zum WLAN sind vor dem Laden in der „Datei my_secs.h" einzutragen.

Aufbau der Webseite

Die von den Empfängern erhaltenen Daten sollen auf einer Webseite dargestellt werden. Der dazu erforderliche HTML-Code wurde in eine eigene Header-Datei (webpage.h) ausgelagert. Die Variable „index_html" enthält den gesamten Code zum Erstellen der Webseite. Diese Variable wird über die Anweisung

```
#include „webpage.h"
```

in das Programm eingebunden. Auf diese Weise kann z. B. auch die Entwicklungsarbeit in einem Team aufgeteilt werden. Während einige Programmierer den Hauptcode erstellen, können andere die Browserseite unabhängig in der webpage.h-Datei aufbauen.

An dieser Stelle können nicht alle Details zum Aufbau von Webseiten vorgestellt werden. Hierzu sei auf die umfangreiche Literatur zu diesem Thema verwiesen. Der Fokus soll vielmehr auf den vom Server gesendeten Ereignissen liegen.

Für die Webseite wird ein EventSource-Objekt erstellt. Dazu wird die URL der Seite angegeben, die die Aktualisierungen sendet:

```
if (!!window.EventSource) { var source = new EventSource(, /events'); ...
```

Nachdem die Ereignisquelle gestartet wurde, wird mit „addEventListener()" auf Nachrichten vom Server gewartet.

Hier wird die JSON-Bibliothek benötigt, da diese die von jedem Board empfangene Daten enthält. Die Daten beinhalten alle erforderlichen Informationen, um die erzeugte Webseite aufbauen zu können. Wie aus den letzten Artikeln bekannt, werden alle Daten im ESP-NOW-System über eine „Structure" versendet. Sie enthält in diesem Fall die Board-ID, die gemessenen Temperatur- und Feuchtigkeitswerte sowie die Messwert-ID. Die Funktion OnDataRecv() wird jedes Mal ausgeführt, wenn ein neues ESP-NOW-Paket empfangen wird.

Wenn der ESP32 ein neues Paket erhält, sendet er einen JSON-String mit den Messwerten als Ereignis („new_readings") an den Client. Die folgenden Zeilen verarbeiten dieses Ereignis:

```
console.log(„new_readings", e.data);
var obj = JSON.parse(e.data);
document.getElementById(„t"+obj.id).innerHTML = obj.temperature.toFixed(2);
document.getElementById(„h"+obj.id).innerHTML = obj.humidity.toFixed(2);
```

Nach dem Starten des Programms werden die IP-Adresse und die WiFi-Kanalnummer des Empfängers ausgegeben (Bild 2).

Nun kann ein beliebiger Browser gestartet werden. Die eben ausgelesene IP-Andresse wird in das Adressfenster des Browsers eingegeben (Bild 3).

Da noch kein Sender verfügbar ist, werden auf der Seite auch noch keine Messdaten angezeigt. Lediglich die Maske für die Klimadaten ist bereits sichtbar.

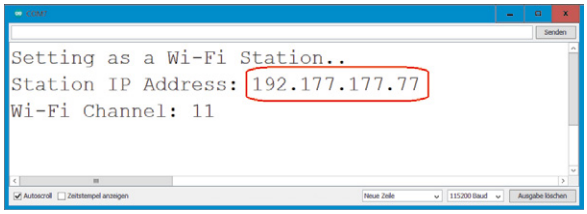


Bild 2: Ausgabe der IP-Adresse im seriellen Monitor

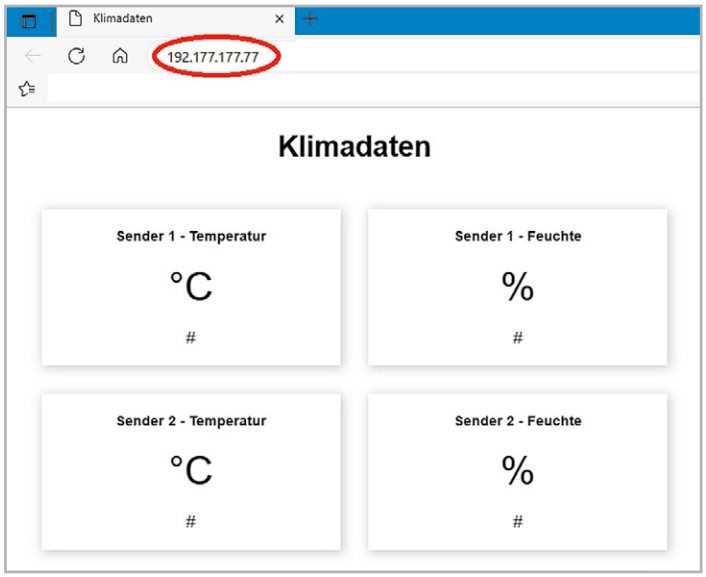


Bild 3: Anzeige der im Empfänger gespeicherten Web-Page

Die Datensender

Um die Internetseite mit Werten zu füllen, sind entsprechende Datensender erforderlich. Diese Sender sind mit jeweils einem SHT20-Temperatur- und Feuchtigkeitssensor ausgestattet. Die Ansteuerung erfolgt also wieder über die I2C-Schnittstelle. Damit können die bereits aus dem zweiten Beitrag (ESP-NOW Teil 2: „Drahtlose Datenübertragung ohne WLAN") bekannten Sensoren weiter verwendet werden. Erfahrene Anwender können natürlich auch andere Messwandler einsetzen.

Bild 4 zeigt das Schaltbild und Bild 5 den Aufbau eines Senders mit SHT20-Sensor.

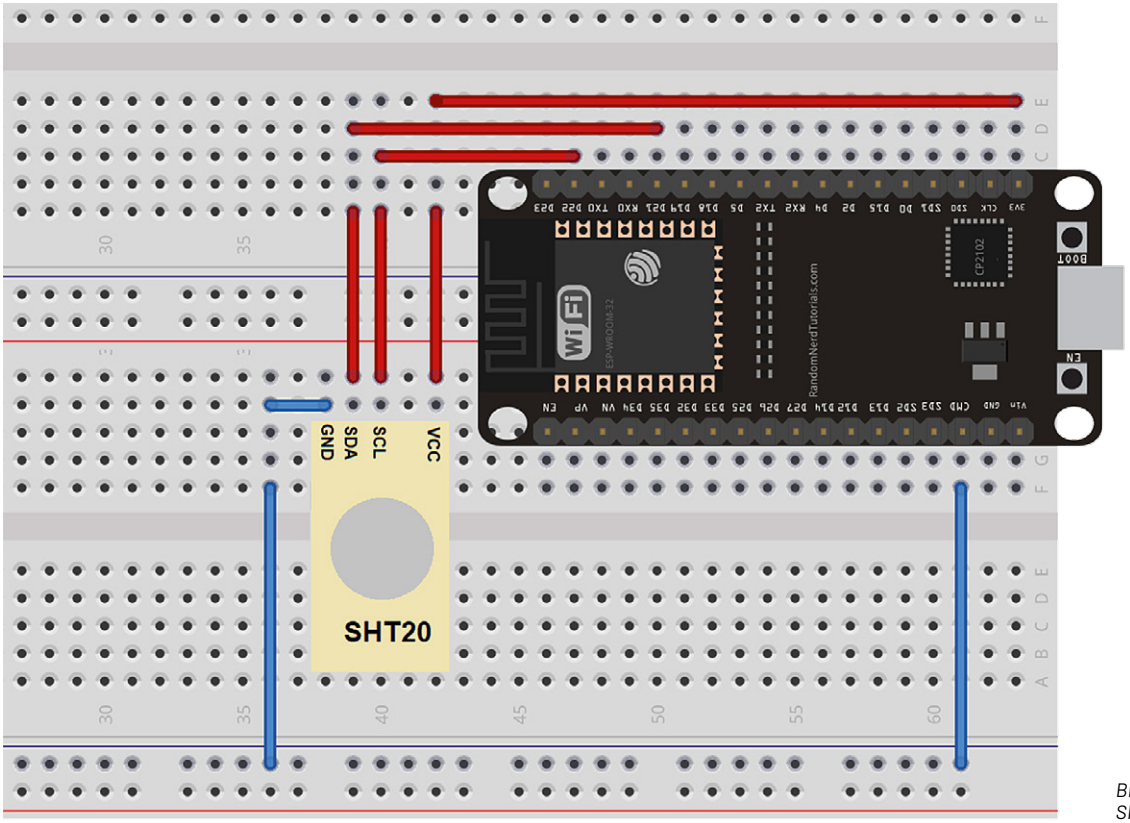


Bild 4: Schaltplan zum Sender mit SHT20-Modul

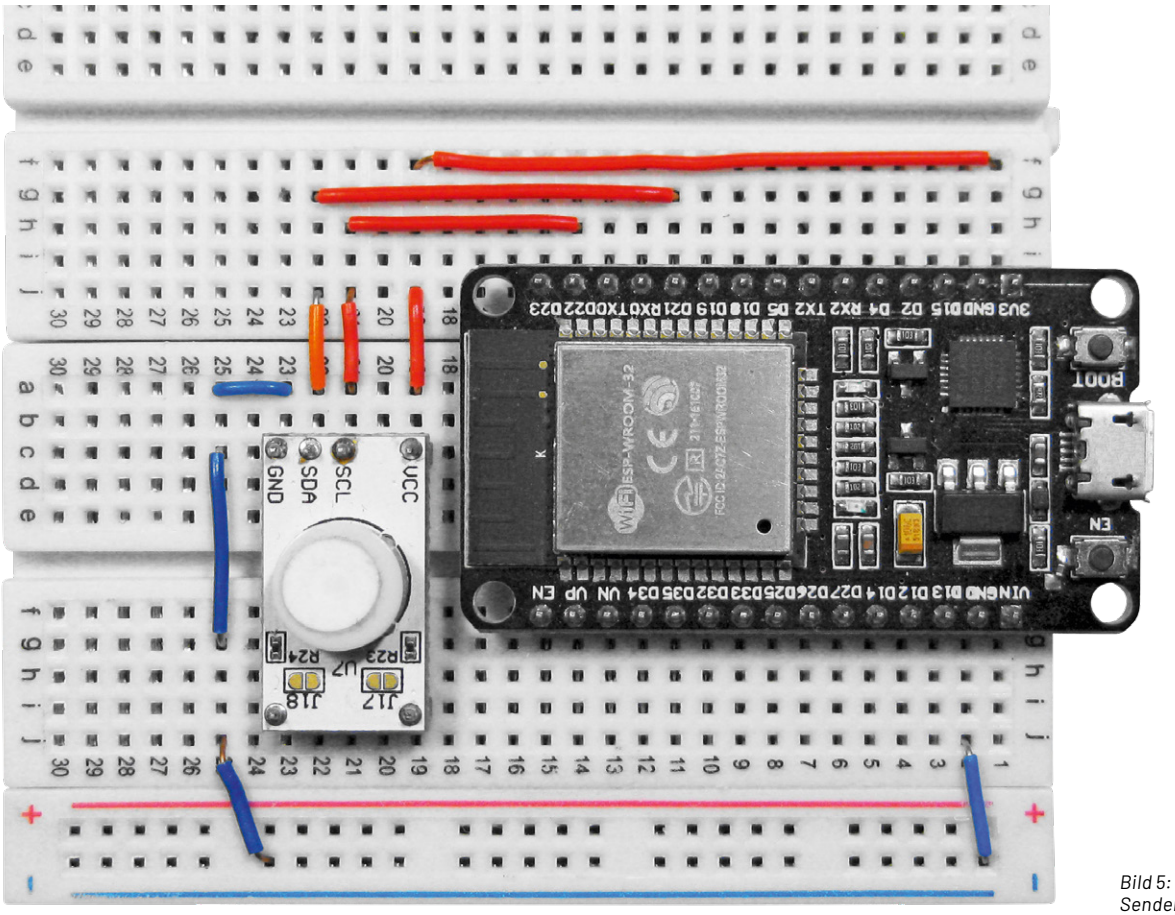


Bild 5: Aufbauvorschlag zum Sender mit SHT20-Modul

Um den Aufbau testen zu können, werden die Messdaten zunächst auf die Konsole der Arduino IDE ausgegeben. Hierzu ist die Bibliothek „uFire_SHT20" erforderlich, die über den Board-Manager geladen werden kann. Die Ausgabe der Werte erfolgt mit folgendem Sketch (SHT20_ESP32.ino):

```
// SHT20_uFire.ino
// ESP32@IDE1.8.9

#include „uFire_SHT20.h"
uFire_SHT20 sht20;

void setup()
{ Serial.begin(115200);
  delay(3000);
  Serial.print(„Temp");Serial.print(„\t\t");
  Serial.println(„Hum");
  Serial.println(„=========================");
  Wire.begin();sht20.begin();
}

void loop()
{ sht20.measure_all();
  Serial.print((String)sht20.tempC + „ °C"); Serial.print(„\t");
  Serial.println((String)sht20.RH + „ %RH");
  delay(1000);
}
```

Die Funktion des Sensors kann durch Anhauchen überprüft werden. Die Werte für Temperatur und Luftfeuchte sollten dabei ansteigen. Bild 6 zeigt eine entsprechende Ausgabe auf dem Seriellen Monitor der Arduino IDE.

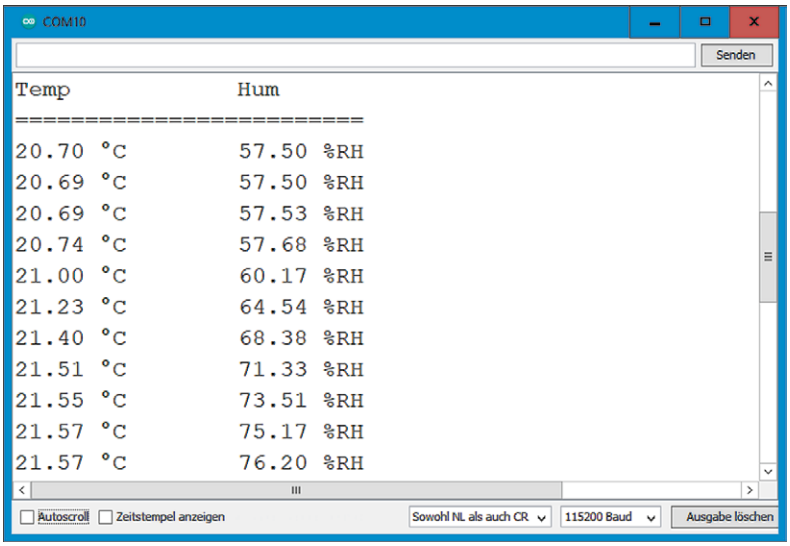


Bild 6: Messwerte des SHT20-Moduls

Jeder Sender überträgt über ESP-NOW eine Struktur, welche die Board-ID, die Temperatur, die Luftfeuchtigkeit und die Messwert-ID enthält. Mit der Messwert-ID steht die Information zur Verfügung, wie viele Nachrichten bereits gesendet wurden.

Beim Laden des Sender-Codes auf die Boards muss die Board-ID-Nummer

#define BOARD_ID 1 # Board 1
#define BOARD_ID 2 # Board 2
usw.

für jeden Sender um jeweils den Wert 1 erhöht werden. Zudem muss jeweils die Empfänger-MAC-Adresse (broadcastAddress[]) angegeben werden. Damit sieht der Sender-Code für den ersten Sender so aus:

```
// ESP-NOW-wi-fi-web-server_data_station_SHT20.ino

// ESP32 @ IDE 1.8.16

#define BOARD_ID 1

#include <esp_now.h>
#include <esp_wifi.h>
#include <WiFi.h>
#include <Wire.h>
#include "uFire_SHT20.h"

uFire_SHT20 sht20;

uint8_t broadcastAddress[] = {0x00,0x00,0x00,0x00,0x00,0x01};  // Rx: Board 01

typedef struct struct_message {int id; float temp; float hum; int readingId;} struct_message;
struct_message myData;
```

```
unsigned long previousMillis = 0;   // Stores last time temperature was published
const long interval = 10000;        // Interval at which to publish sensor readings
unsigned int readingId = 0;
constexpr char WIFI_SSID[] = "YOUR_SSID"; // Insert your SSID

int32_t getWiFiChannel(const char *ssid)
{ if (int32_t n = WiFi.scanNetworks())
  { for (uint8_t i=0; i<n; i++)
    { if (!strcmp(ssid, WiFi.SSID(i).c_str()))  return WiFi.channel(i); }
    return 0;
  }
}

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup()
{ Serial.begin(115200);
  Wire.begin(); sht20.begin();
  WiFi.mode(WIFI_STA);
  int32_t channel = getWiFiChannel(WIFI_SSID);
  WiFi.printDiag(Serial); // Uncomment to verify channel number before
  esp_wifi_set_promiscuous(true);
  esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
  esp_wifi_set_promiscuous(false);
  WiFi.printDiag(Serial); // Uncomment to verify channel change after
  if (esp_now_init() != ESP_OK) { Serial.println("Error initializing ESP-NOW"); return; }
  esp_now_register_send_cb(OnDataSent);
  esp_now_peer_info_t peerInfo;   //Register peer
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  { Serial.println("Failed to add peer"); return;
  }
}

void loop()
{ unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval)
  { previousMillis = currentMillis;

    myData.id = BOARD_ID;
    sht20.measure_all();
    myData.temp = sht20.tempC;    Serial.println(sht20.tempC);
    myData.hum = sht20.RH;        Serial.println(sht20.RH);
    myData.readingId = readingId++;

    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
    if (result == ESP_OK) Serial.println("Data Sent"); else Serial.println("Send Fail");
  }
}
```

Natürlich muss auch hier wieder die korrekte MAC-Adresse des Empfänger-Boards angegeben werden:

```
uint8_t broadcastAddress[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x01};  // Rx: Board 01
```

Es folgt die Festlegung der Datenstruktur für die zu übertragenden Daten. Die Routine

```
getWiFiChannel(const char *ssid)
```

sorgt für die automatische Zuordnung eines WiFi-Kanals.
Hierzu muss über

```
constexpr char WIFI_SSID[] = „YOUR_SSID“; // Insert your SSID
```

auch die SSID des verwendeten WLANs angegeben werden.
Im Setup werden wie üblich alle erforderlichen Dienste gestartet.

Die Hauptschleife sorgt schließlich dafür, dass die Daten alle 10 Sekunden aktualisiert werden.
Das Aktualisierungsintervall kann dabei über die Variable

```
const long interval = 10000;
```

in Millisekunden (10000 ms = 10 s) gewählt werden.

Datenempfang und Übertragung ins WLAN

Nachdem beide Sender aktiviert wurden, erscheinen die Daten schließlich auch auf der Webseite (Bild 7). Somit ist es also möglich, Daten aus verschiedenen ESP-Messstationen zu sammeln und zentral anzuzeigen. Das System kann ohne Probleme auch auf mehr als zwei Messstationen erweitert werden. Auf diese Weise können die verschiedensten Daten aus Haus und Hof oder Büro gesammelt und auf einer ein-zigen Webseite angezeigt werden.

Da der ESP32 eine Standard-Webseite erzeugt, kann diese nicht nur auf einem PC oder Laptop dargestellt werden. Vielmehr können alle internetfähigen Geräte die Daten anzeigen. So können auch Mobiltelefone oder Tablets als Anzeigeeinheiten dienen. Bild 8 zeigt die Webseite des ESP32 auf einem Smartphone und einem Tablet.

Damit bietet das ESP-NOW System vielfältige Möglichkeiten, Messdaten aller Art sowohl stationär als auch mobil darzustellen. Hieraus ergibt sich eine nahezu unbegrenzte Vielfalt von Anwendungen. Neben der hier exemplarisch vorgestellten Klimadatenerfassung können beispielsweise auch

- Lichtwerte zur Messung von Raumhelligkeiten
- Alarmsensoren
- barometrische Wetterdaten
- Hallsensoren zur Überwachung von Fenstern und Türen etc.

erfasst und flexibel angezeigt werden. Den Anwendungsmöglichkeiten sind hier kaum Grenzen gesetzt.

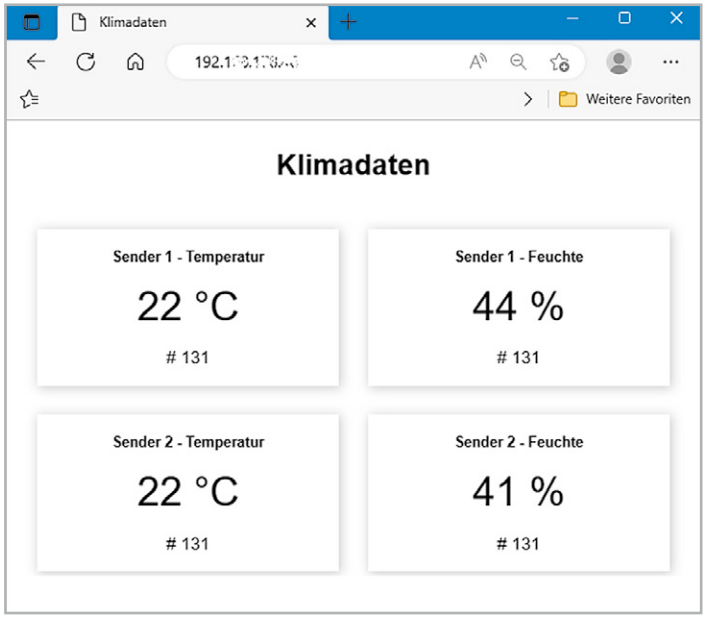


Bild 7: Klimadaten im Browser

Bild 8: Darstellung der Daten auf verschiedenen Mobilgeräten

Zusammenfassung und Ausblick

Mit diesem Beitrag findet die Serie zur Anwendung des ESP-NOW-Systems ihr Ende. Ausgehend von der einfachen Datenübertragung über bidirektionale Übertragungswege bis hin zu komplexen Netzwerken mit flexiblem Informationsfluss zwischen allen Stationen wurden die vielfältigen Möglichkeiten des Systems vorgestellt. Die Anwendungsmöglichkeiten reichten dabei von der Übertragung von Sensordaten über die Fernsteuerung von Modellen bis hin zum Aufbau kompletter Webseiten mit umfangreicher Datendarstellung.

Natürlich kann eine vierteilige Artikelserie nicht alle erdenklichen Applikationen abdecken. In jedem Fall sollte der Anwender aber nach dem Durcharbeiten der Serie in der Lage sein, eigene Ideen im Bereich Heimautomatisierung, Sicherheitstechnik, Datenerfassung oder auch Modellbau erfolgreich umzusetzen. ELV

Material	Artikel-Nr.
3x ESP32-Board	145164
2x SHT-20-Sensoren Diese sind beispielsweise enthalten im Prototypen Adapterset PAD-4 „Digitale Komponenten und Mikroprozessor-Peripherie“	155107

Weitere Infos

Download-Paket:
Artikel-Nr. 253837