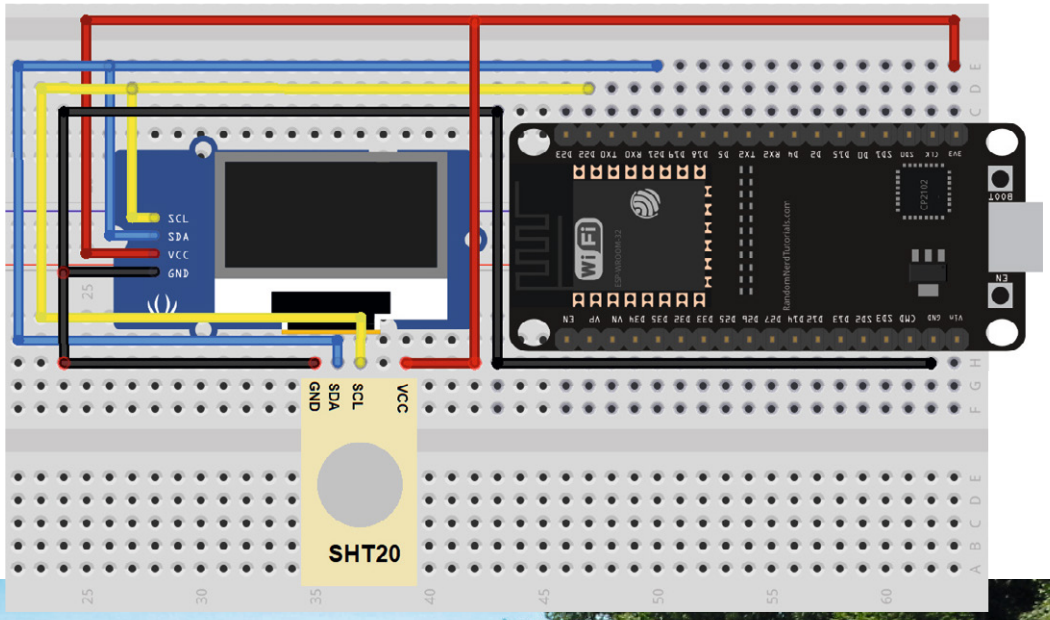
ESP-NOW

Teil 2

Drahtlose Datenübertragung ohne WLAN

Im ersten Teil dieser Artikelserie haben wir die Grundlagen des drahtlosen ESP-NOW-Kommunikationsprotokolls vorgestellt. Dabei wurden neben den verschiedenen Sende- und Empfangsfunktionen in einem ersten Beispiel auch einfache Datenpakete von einem ESP32-Board zu einem anderen gesendet. Im zweiten Teil stehen nun praktische Anwendungen im Vordergrund. Nutzdaten wie etwa Sensormesswerte können ähnlich einfach übertragen werden wie die Datenpakete, die in Teil 1 gezeigt wurden. Auch die drahtlose Steuerung von Ausgangspins eines ESP32-Boards ist möglich. Neben einer bidirektionalen Messwertübertragung sollen Praxisprojekte wie eine drahtlose Türklingel oder eine ESP-NOW-basierte Fernsteuerung für Modellbauzwecke genauer diskutiert werden.



Schalten von Verbrauchern: drahtloser (LED-)Schalter

Das drahtlose Schalten von Verbrauchern ist eine der wichtigsten Anwendungen des ESP-NOW-Systems. Die kostengünstigen Boards finden hier einen weiten Anwendungsbereich. Die Steuerung von WLAN-Steckdosen und drahtlosen Smart-Home-Funktionen war sogar eine der grundlegenden Motivationen für die Entwicklung des ESP32-Chips bzw. seines Vorgängers ESP8266.

Der Hardware-Aufbau für den drahtlosen Schalter ist sehr einfach gehalten. Am Sender-ESP sind nur zwei Taster (Bild 1), der Empfänger benötigt nur eine externe LED (Bild 2).

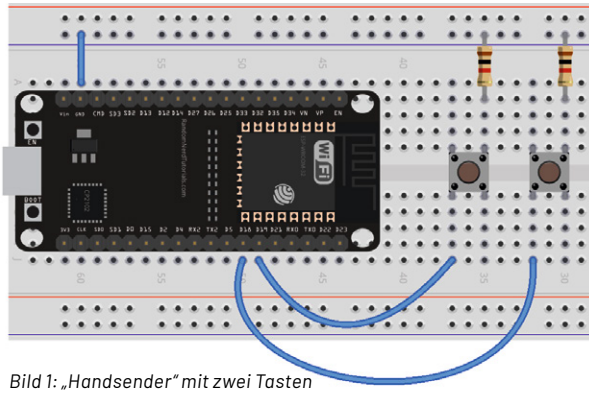


Bild 1: „Handsender" mit zwei Tasten

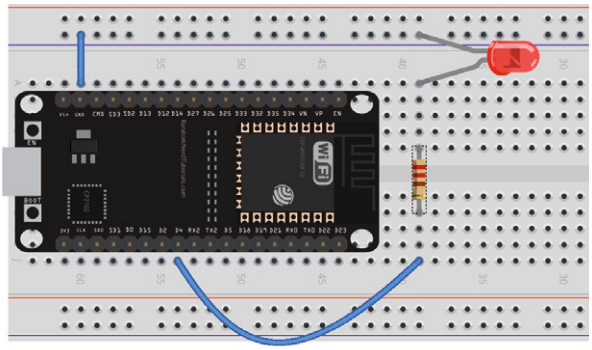


Bild 2: Empfänger mit LED an Port 4

Wichtiger Hinweis:
ESP32-Boards sind in verschiedenen Varianten auf dem Markt. Die Aufbauvorschläge können daher nur als Anhaltspunkte für eine Version dienen. Die Portnummern sind jedoch an allen Modulen identisch. Lediglich der Formfaktor oder die Lage einzelner Pins können abweichen. Sicherheitshalber werden daher die Anschlusspins zusätzlich in Tabellenform angegeben.

Für die beiden Taster gilt hier:
Taster 1 an GPIO18
Taster 2 an GPIO19
Die LED am Empfänger wird an Port 4 angeschlossen.

Im Sketch zu diesem Projekt werden die Schaltdaten für die LED übertragen. Alle Programme zu diesem Artikel sind als Download-Paket verfügbar [1]. Die Sketche werden wie üblich über die Arduino IDE geladen. Wenn die Grundlagen aus dem ersten Teil dieser Artikelserie bekannt sind, sollten dabei keine Verständnisprobleme auftreten.

Das Programm für den Sender sieht folgendermaßen aus:

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

```
// LED_switch_Tx.ino
// ESP32 @ IDE 1.8.16

#include <esp_now.h>
#include <WiFi.h>

uint8_t broadcastAddress[] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}; // receiver MAC adress

byte LEDonPin=18, LEDoffPin=19;

typedef struct struct_message
{bool sw;} struct_message;
struct_message myData;
esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{ Serial.print(status==ESP_NOW_SEND_SUCCESS?"Data Sent\t":"Send Fail\t");}

void setup()
{ Serial.begin(115200);  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) { Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0; peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {  Serial.println("Failed to add peer"); return;}
  pinMode(LEDonPin,INPUT_PULLUP); pinMode(LEDoffPin,INPUT_PULLUP);
}

void loop()
{ if (digitalRead(LEDonPin)==false)    myData.sw=true;
  if (digitalRead(LEDoffPin)==false)   myData.sw=false;
  Serial.println(myData.sw);
  esp_err_t result=esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
  Serial.println(result==ESP_OK?"Delivery Success":"Delivery Fail");
  delay(20);
}
```

In der Zeile

```
uint8_t broadcastAddress[] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}; // receiver MAC adress
```

muss wieder die korrekte MAC-Adresse des Empfängermoduls eingetragen werden. Details dazu finden sich im ersten Teil der Artikelserie.

Nach dem Einbinden der erforderlichen Librarys werden die beiden Pinnummern für die Eingabetaster (GPIO 18 und 19) festgelegt. Die Struktur für die Sendedaten besteht nun lediglich aus einer boolschen Variable „sw" für „switch".

Im Setup werden der Serielle Monitor für Kontrollzwecke und das WiFi-System gestartet. Danach wird der über die MAC-Adresse bekannt gegebene Peer (Empfänger) eingebunden. Falls das ESP-System nicht erfolgreich gestartet werden konnte oder der Peer nicht erreichbar ist, werden entsprechende Fehlermeldungen ausgegeben.

Danach werden die Schalter-Pins als Eingänge definiert:

```
pinMode(LEDonPin,INPUT_PULLUP); pinMode(LEDoffPin,INPUT_PULLUP);
```

Da die internen Pull-up-Widerstände aktiviert werden, kann auf externe Komponenten verzichtet werden. Das Betätigen der Taster legt die - intern auf „High" gelegten - Ports auf Masse-Potential („LOW").

In der Hauptschleife werden die Taster abgefragt. Über den LEDonPin wird die LED auf der Empfängerseite ein-, über den LEDoffPin ausgeschaltet. Da die Schalter auf Masse schalten, sind sie LOW-aktiv, deshalb erfolgt die Abfrage auf „false" (= „LOW"). Anschließend werden die Daten via ESP-NOW übertragen. Im Fehlerfall wird wieder eine entsprechende Information ausgegeben.

Auf Empfängerseite muss lediglich eine LED an einen beliebigen in der Software festgelegten I/O-Pin (im Beispiel wurde Pin 4 gewählt) angeschlossen werden. Das zugehörige Empfängerprogramm sorgt für das schalten der LED an Port 4:

```
// LED_switch_Rx.ino
// ESP32 @ IDE 1.8.16

#include <esp_now.h>
#include <WiFi.h>

byte LEDpin=4;

typedef struct struct_message
{bool sw;} struct_message;
struct_message myData;

void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len)
{ memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Switch State: "); Serial.println(myData.sw);
  digitalWrite(LEDpin,myData.sw);
}

void setup()
{ Serial.begin(115200); WiFi.mode(WIFI_STA);
  pinMode(LEDpin,OUTPUT); digitalWrite(LEDpin,LOW);
  if (esp_now_init()!=ESP_OK){Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_recv_cb(OnDataRecv);
}

void loop() {}
```

Der Empfang der Schaltdaten erfolgt in der Routine OnDataRecv(). Für Kontrollzwecke wird die empfangene Information auf die Serielle Schnittstelle ausgegeben. Dann wird die LED über die Anweisung digitalWrite() in den jeweils aktuellen Schaltzustand (on/off) gebracht.

Nachdem beide Programme geladen sind, wird die LED am Empfänger aufleuchten, sobald der „ON"-Taster (an Port 18) gedrückt wird. Über den „OFF"-Taster (Port 19) kann die LED wieder ausgeschaltet werden. Natürlich kann die LED problemlos gegen ein Relais ausgetauscht werden, gegebenenfalls auch mit integriertem Treibertransistor. So entsteht eine universelle Fernbedienung, über die Verbraucher bequem drahtlos geschaltet werden können.

Drahtlose Türklingel, Garagentoröffner oder Fernbedienungen für Jalousien

Oft steht man vor der Situation, dass man eine Türklingel nicht nur von der Haus- oder Wohnungstür aus betätigen möchte. Ein Besucher soll beispielsweise auch vom Gartentor aus auf einen Klingelknopf drücken können. Wenn keine entsprechenden Leitungen vorhanden sind, kann ein Umbau ein kostspieliges Unterfangen sein. Umfangreiche Erdarbeiten wären nötig, um ein entsprechendes Kabel zu verlegen, eine Durchführung in das Gebäude müsste erstellt werden usw.

Mit einem Satz ESP32-Controllern kann die Aufgabe dagegen einfach, schnell und elegant gelöst werden. Der drahtlose LED-Schalter aus dem letzten Abschnitt muss dafür nur geringfügig modifiziert werden.

Zunächst kommt eine elektronische Türklingel mit nur einem einzigen Taster (Port 18) auf der Senderseite zum Einsatz. Am Empfänger wird die LED durch einen (aktiven) Buzzer ersetzt (Bild 3). Wird nun der Taster betätigt, ertönt ein lautes Tonsignal.

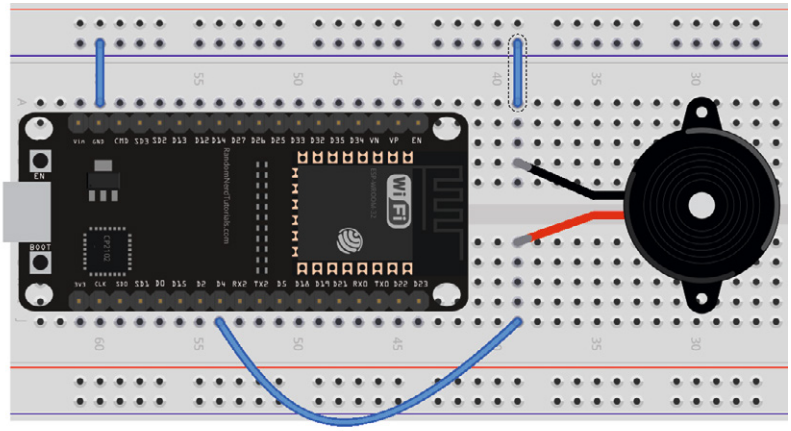


Bild 3: ESP32-Empfänger als drahtlose Türklingel

Sobald der Taster losgelassen wird, endet auch das Signal. Diese Version eignet sich bestens als drahtlose elektronische Türklingel.

In ähnlicher Weise können mit diesem System auch drahtlose Garagentoröffner oder Jalousien-Fernbedienungen umgesetzt werden. Das Programm dazu ist im Download-Paket unter „Wireless_Doorbell“ enthalten.

Drahtloser LED-Dimmer

Dass nicht nur einfache Schaltsignale mit dem ESP-NOW-System übertragbar sind, zeigt die folgende Anwendung. Anstelle eines ON/OFF-Signals wird hier ein quasi-analoger Wert übertragen.

Im einfachsten Fall wird dazu ein analoger Spannungswert über ein Potentiometer erzeugt. Dieser Wert wird über einen internen AD-Wandler des ESP32 digitalisiert. Der so gewonnene Digitalwert wird dann auf die bekannte Weise an den Empfänger übertragen.

Für den Sender ergibt sich folgender Code:

```
// ESP_now_LED_ctrl_analog_Tx.ino
// ESP32 @ IDE 1.8.16

#include <esp_now.h>
#include <WiFi.h>
uint8_t broadcastAddress[] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}; // receiver MAC adress

byte analogInPin=34;

typedef struct struct_message
{ int analogVal;} struct_message;
struct_message myData; esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{ Serial.print(status==ESP_NOW_SEND_SUCCESS?"Data Sent\t":"Send Fail\t");}

void setup()
{ Serial.begin(115200);  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK)   { Serial.println("Error initializing ESP-NOW"); return;  }
  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;    peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK)  {  Serial.println("Failed to add peer"); return; }
}

void loop()
{ myData.analogVal = analogRead(analogInPin);
  Serial.print("Pot voltage: "); Serial.println(myData.analogVal);
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
  Serial.println(result==ESP_OK?"Delivery Success":"Delivery Fail");
  delay(200);
}
```

Das Programm liest analoge Spannungswerte vom Potentiometer ein. Die Struktur „struct_message" enthält nur das Element „analogVal" vom Typ Integer, das die Spannungswerte aufnimmt. In der „setup()"-Funktion wird die serielle Kommunikation initialisiert, der WLAN-Modus auf „WIFI_STA" gesetzt und die ESP-NOW-Bibliothek initialisiert. Dann wird eine „OnDataSent()" Funktion registriert, die aufgerufen wird, wenn ein Sendevorgang abgeschlossen ist. Die MAC-Adresse des Empfängers wird in die „peerInfo"-Struktur kopiert, und ein Peer wird mit der Funktion „esp_now_add_peer()" hinzugefügt.

In der „loop()"-Funktion wird der Analogwert des Potentiometers gelesen und in der „myData"-Struktur gespeichert. Der Analogwert wird testweise auf der seriellen Konsole ausgegeben und dann mit der „esp_now_send()"-Funktion an den Empfänger gesendet. Der Status des Sendevorgangs wird ebenfalls auf der seriellen Konsole ausgegeben. Eine Verzögerung von 200 Millisekunden sorgt für eine klar getrennte Übertragung der einzelnen Daten.

Das Empfängerprogramm sieht so aus:

```
// ESP_now_LED_ctrl_Rx.ino
// ESP32 @ IDE 1.8.16

#include <esp_now.h>
#include <WiFi.h>

byte LEDpin=4;
int frequency=1000, resolution=8, channel=0;

typedef struct struct_message
{ int brightness;} struct_message;
struct_message myData;

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{ memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Received Data: "); Serial.println(myData.brightness);
  ledcWrite(channel,myData.brightness/16);
}

void setup()
{ Serial.begin(115200); WiFi.mode(WIFI_STA);
  pinMode(LEDpin,OUTPUT);
  ledcSetup(channel,frequency,resolution); ledcAttachPin(LEDpin,channel);
  if (esp_now_init() != ESP_OK)
  { Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_recv_cb(OnDataRecv);
}

void loop() {}
```

Das Programm entspricht in weiten Teilen dem Empfängerprogramm für den LED-Schalter. Der wesentliche Unterschied besteht darin, dass nun nicht nur boolesche Signale empfangen werden, sondern Integer-Werte. Diese werden dann über die PWM-Funktion „ledcWrite()" an die LED weitergeleitet. Die Pulsweiten-Modulation (PWM) sorgt dann für die kontinuierlichen Helligkeitsänderungen.

Der Hardware-Aufbau des Empfängers kann unverändert aus Bild 2 (LED an Port 4) übernommen werden. Am Sender ist anstelle der Taster ein Potentiometer an Analog-Port 34 anzuschließen (Bild 4).

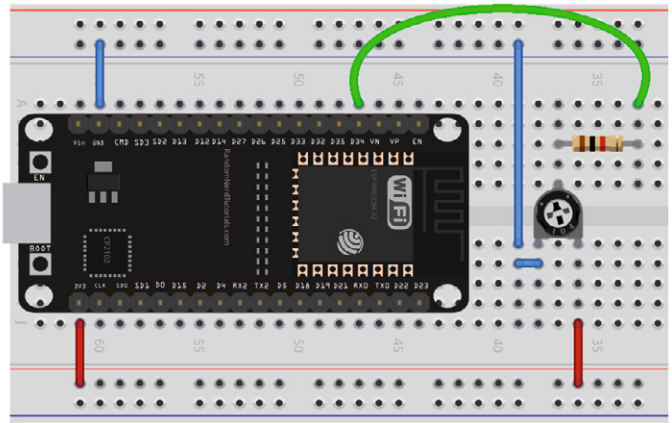


Bild 4: Sender zum drahtlosen LED-Dimmer

Nachdem die Programme auf die Controller geladen wurden, kann durch Drehen am Poti auf der Senderseite die Helligkeit der LED am Empfänger kontinuierlich verändert werden.

Ferngesteuerter Servo

Mit dem ESP-NOW-System geht der Traum vieler Modellbauer in Erfüllung. Man kann damit eine Fernsteuerung mit nahezu beliebig vielen Kanälen realisieren. Eine sichere Reichweite von ca. 100 m in freiem Gelände ist zudem für viele Anwendungen ausreichend. Man kann so beispielsweise Elektroautos oder Modellboote steuern. Bei Flugzeugen oder Drohnen sollte man allerdings doch besser auf kommerziell verfügbare RC-Systeme umsteigen. Das folgende Projekt zeigt exemplarisch, wie man mehrere Modellbauservos über ESP-NOW-Steuern kann.

Dieses Mal kann der Sender nach Bild 4 unverändert bleiben. Für den Empfänger kann der Aufbau analog zu Bild 5 verwendet werden. Hier wurde der Übersichtlichkeit halber nur ein Servo an Pin 13 eingezeichnet. Die Software unterstützt allerdings bis zu vier Kanäle.

Die folgende Tabelle zeigt die Zuordnung der Kanäle zu den ESP32-Pins:

Kanal	ESP 32
Servo 1	13
Servo 2	12
Servo 3	15
Servo 4	18

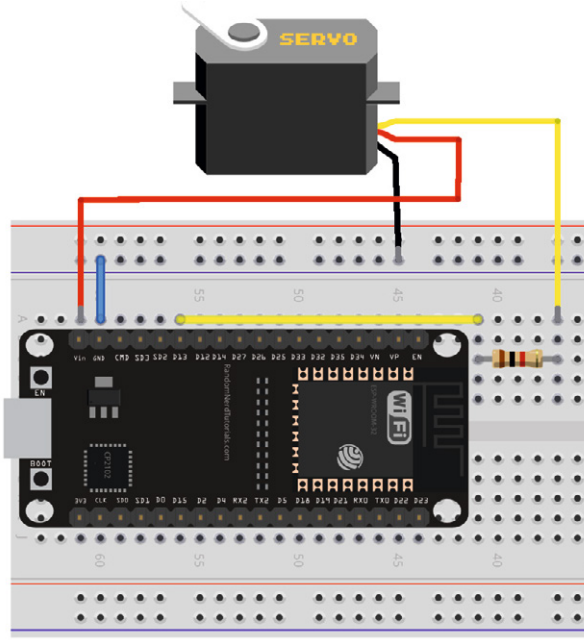


Bild 5: Ferngesteuerter Servo

Mit den oben angegebenen Anschlussbelegungen ergibt sich das folgende Empfängerprogramm:

```
// ESP_now_4_channel_RC_Rx.ino
// ESP32 @ IDE 1.8.16

#define ADCtoDEG 180/4095 // convert ADC readings to Servo degrees

#include <esp_now.h>
#include <WiFi.h>
#include <Servo.h>

Servo myservo1; Servo myservo2; Servo myservo3; Servo myservo4;// 12 servo objects max
int servoPos01=0, servoPos02=0, servoPos03=0, servoPos04=0;  // servo positions

typedef struct struct_message
{ int servoPos01; int servoPos02; int servoPos03; int servoPos04;} struct_message;
struct_message myData;

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{ memcpy(&myData, incomingData, sizeof(myData));
  Serial.println(myData.servoPos01*ADCtoDEG); Serial.println(myData.servoPos02*ADCtoDEG);
  Serial.println(myData.servoPos03*ADCtoDEG); Serial.println(myData.servoPos04*ADCtoDEG);
  Serial.println("====================");

  myservo1.write(myData.servoPos01*ADCtoDEG); myservo2.write(myData.servoPos02*ADCtoDEG);
  myservo3.write(myData.servoPos03*ADCtoDEG); myservo4.write(myData.servoPos04*ADCtoDEG);
}

void setup()
{ Serial.begin(115200); WiFi.mode(WIFI_STA);
  if (esp_now_init()!= ESP_OK) {Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_recv_cb(OnDataRecv);
  myservo1.attach(13); myservo2.attach(12); myservo3.attach(15); myservo4.attach(18);
}

void loop() {}
```

Im Programm wird die Servo-Bibliothek verwendet. Falls diese noch nicht geladen ist, kann dies über
Sketch → Bibliothek einbinden
in der Arduino IDE nachgeholt werden.

Das Programm für den Sender sieht so aus:

```
// ESP_now_4_channel_RC_Tx.ino
// ESP32 @ IDE 1.8.16

#include <esp_now.h>
#include <WiFi.h>
// uint8_t broadcastAddress[] = {0x80, 0x7D, 0x3A, 0xDD, 0x12, 0x84}; // receiver MAC adress
uint8_t broadcastAddress[] = {0x30,0xAE,0xA4,0xF5,0x7A,0xE8}; // receiver MAC adress
byte ch01=34, ch02=35, ch03=32, ch04=33;

typedef struct struct_message
{int servoPos01; int servoPos02; int servoPos03; int servoPos04;} struct_message;
struct_message myData; esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{ Serial.print(status==ESP_NOW_SEND_SUCCESS?"Package sent\t":"Send Fail\t");}

void setup()
{ Serial.begin(115200);  WiFi.mode(WIFI_STA);
  if (esp_now_init()!= ESP_OK) { Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6); // 6 digit MAC
  peerInfo.channel=0; peerInfo.encrypt=false;
  if (esp_now_add_peer(&peerInfo)!=ESP_OK) {Serial.println("Failed to add peer"); return;}
}

void loop()
{ myData.servoPos01=analogRead(ch01);  myData.servoPos02=analogRead(ch02);
  myData.servoPos03=analogRead(ch03);  myData.servoPos04=analogRead(ch04);
  esp_err_t result=esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
  Serial.println(result==ESP_OK?"Delivery success":"Error sending data");
  delay(20);
}
```

Auf der Senderseite sind ebenfalls bereits vier Kanäle vorbereitet. Bei Bedarf können also bis zu vier Potentiometer nach dem folgenden Schema angeschlossen werden:

Potentiometer	ESP32-Pin
Kanal 1	34
Kanal 2	35
Kanal 3	32
Kanal 4	33

Anstelle von vier einzelnen Potentiometern können auch zwei Joysticks (siehe Bild 6) mit jeweils zwei Potentiometerfunktionen (x/y-Kreuzhebel) verwendet werden. Bei der Fernsteuerung von Modellen erfreuen sich die Joysticks großer Beliebtheit, da sie eine wesentlich intuitivere Steuerung zulassen als einzelne Potentiometer.

Nach dem Anschluss der Potentiometer bzw. Joysticks und den Servos können die Programme geladen werden. Sobald eine Joystick-Position bzw. eine Potentiometerstellung verändert wird, folgt der zugehörige Servo präzise den Steuerbefehlen. Natürlich kann der ESP32 zusätzlich weitere Funktionen übernehmen, wie etwa die Ansteuerung blinkender Positionslichter oder sogar die Auswertung von Lagesensoren etc.

Ein besonderer Vorteil dieser Eigenbau-Fernsteuerung ist neben den geringen Kosten ihre hohe Flexibilität. Im Beispiel wurden vier Servo-Kanäle vorgesehen. Über die PWM-Ausgänge des ESP32 kann man aber auch die beliebten „Fahrtenregler" integrieren. Damit lassen sich dann auch Elektromotoren für Fahrzeuge oder Boote direkt ansteuern.

Eine ideale Anwendung für die hier vorgestellte RC-Anlage sind Modellyachten. Durch deren geringe Geschwindigkeit und den Wind-

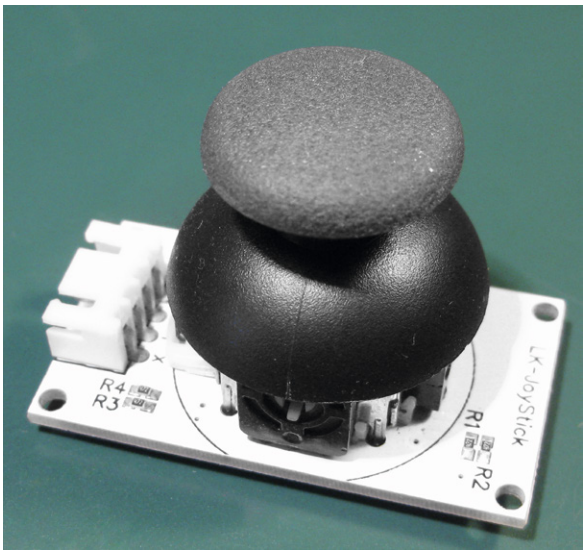


Bild 6: Joystick-Modul

Bild 7: Mit ESP-NOW ferngesteuerte Modellsegelyacht

antrieb sind die Anforderungen an die Zuverlässigkeit der Anlage relativ gering. Modellsegelschiffe (s. Bild 7) sind daher gut geeignet, um erste Erfahrungen mit dem ESP-NOW-System im Modellbau zu gewinnen. Wenn dann ausreichend Kenntnisse über Reichweite und Zuverlässigkeit der selbst erstellten Anlage vorliegen, kann man sich auch an anspruchsvollere Projekte wie Rennboote oder schnelle RC-Cars wagen.

Übertragung von Sensorwerten

Das Dimmen einer Beleuchtung oder das Steuern eines Servos sind nur einige mögliche Anwendungen für die Übertragung von digitalisierten analogen Daten. Eine noch größere Bedeutung hat die drahtlose Übermittlung von Messwerten. Rüstet man die Senderseite mit einem Sensor aus, entsteht ein Datenübertragungssystem mit beachtlicher Reichweite. Damit lassen sich z. B. Temperaturen in einem Gewächshaus, die Beleuchtungsintensität einer Solarzelle oder die Feuchtigkeit in einer Garage, einem Keller oder einem Abstellraum bequem erfassen. Der Sensor wird einfach im betreffenden Bereich positioniert. Die dort herrschenden Werte können dann zentral an einem Rechner im Haus oder in der Wohnung abgelesen werden. Bild 8 zeigt den Aufbau für die Erfassung von Helligkeitswerten. Als Sensor kann eine beliebige Photodiode oder ein Phototransistor (z. B. BPW40) eingesetzt werden.

Das Potentiometer dient zur Einstellung der Empfindlichkeit. Damit lassen sich die Helligkeitswerte bis zu einem gewissen Maß kalibrieren.

Für die Messung von absoluten Helligkeiten in Lux-Einheiten wäre allerdings die Kalibration mit einem kommerziellen Luxmeter erforderlich. Für die Erfassung von relativen Helligkeitswerten kann der Sensor allerdings auch ohne Kalibration verwendet werden.

Das Sendeprogramm dazu ist praktisch identisch mit dem des Servo-Senders und soll daher hier nicht nochmals wiedergegeben werden. Der vollständige Sketch findet sich im Download-Paket.

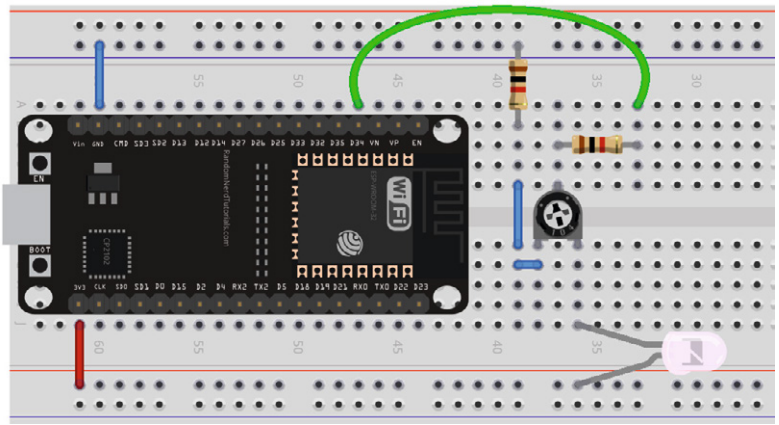


Bild 8: Sender zur drahtlosen Übermittlung von Helligkeitswerten

Auf der Empfängerseite kann das Programm so aussehen:

```
// ESP_now_LED_ctrl_Rx.ino
// ESP32 @ IDE 1.8.16

#include <esp_now.h>
#include <WiFi.h>

typedef struct struct_message
{ int brightness;} struct_message;
struct_message myData;

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{ memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Received Data: "); Serial.println(myData.brightness);
}

void setup()
{ Serial.begin(115200); WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK)
  { Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_recv_cb(OnDataRecv);
}

void loop() {}
```

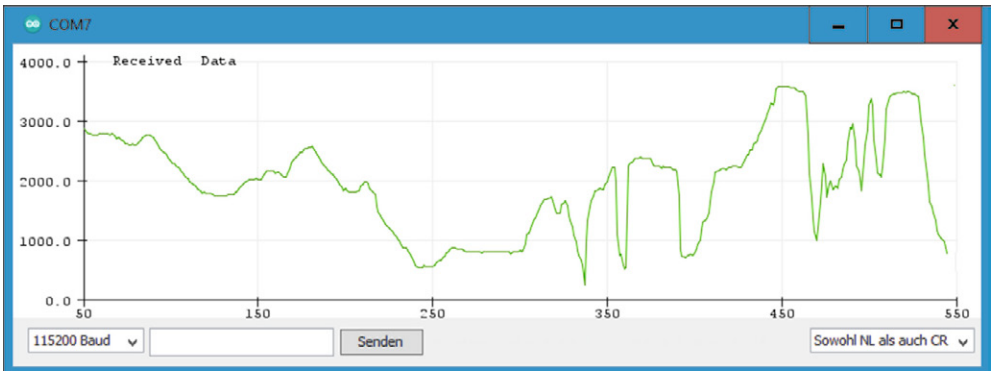


Bild 9: Grafische Aufzeichnung von Helligkeitswerten mit dem ESP-NOW-System

Verwendet man in der Arduino IDE nicht den Seriellen Monitor zur Ausgabe der Werte, sondern den Plotter, erhält man eine grafische Aufzeichnung der Helligkeitswerte. Bild 9 zeigt ein Beispiel dazu.

Bidirektionale Kommunikation und Messwertübertragung

Bereits im ersten Beitrag zu dieser Serie wurde klar, dass über das ESP-NOW-System eine bidirektionale Kommunikation aufgebaut wird. Über die Callback-Funktion erfolgte bereits eine Rückmeldung, vom Empfänger zum Sender. Darin wurde mitgeteilt, ob die Nachricht korrekt empfangen wurde. Der Rückkanal lässt sich nicht nur für die Datenquittierung nutzen. Vielmehr ist auch eine echte bidirektionale Verbindung zwischen zwei ESP32-Controllern möglich.

In diesem Abschnitt soll gezeigt werden, wie Sensordaten zwischen zwei ESP32-Boards ausgetauscht werden können. Für die Anzeige der Daten wird jeder Controller mit einem OLED-Display ausgerüstet. Die Daten werden von einem SHT20-Sensor (s. Materialliste am Ende des Artikels) gelesen. Für einen Datenaustausch sind natürlich zwei Hardwareeinheiten erforderlich.

Möchte man nicht nur Temperaturen erfassen, können auch andere Sensoren verwendet werden. Kombisensoren für Temperatur und Luftfeuchte wie der SHT20 leisten hier oftmals gute Dienste. Diese Messwandler liefern digitale Temperatur- und Feuchtigkeitswerte. Sie verwenden das I2C-Kommunikationsprotokoll, um die Messwerte an den ESP32-Controller zu übertragen. Der Sensor ist in der Lage, Temperaturen im Bereich von -40 bis +125 Grad Celsius und relative Luftfeuchtigkeiten von 0 bis 100 Prozent zu messen. Dank seiner hohen Genauigkeit und Zuverlässigkeit wird der SHT20-Sensor häufig in verschiedenen Anwendungen wie Klimatisierung, Wetterstationen, Industrieautomatisierung und anderen Bereichen eingesetzt.

Jede Station kann Temperatur- und Feuchtigkeitsmesswerte erfassen. Zusätzlich sendet jeder Controller seine Messwerte über ESP-NOW an das jeweils andere Board. Wenn dieses die Messwerte empfängt, zeigt es sie ebenfalls auf seinem OLED-Display an. Zudem wird nach dem Senden der Messwerte angegeben, ob die Daten erfolgreich übermittelt wurden. Bei dieser Anwendung muss jedes Board die MAC-Adresse des anderen kennen, um die Nachricht senden zu können. Später kann dieses Setup um weitere Boards ergänzt werden. Dabei können dann alle Boards miteinander kommunizieren und Daten austauschen. Auf diese Weise ist es dann möglich, ganze Messnetze aufzubauen. Im dritten Artikel zu dieser Serie wird diese Variante genauer vorgestellt werden.

Neben der ESP32-NOW-Library sind für dieses Projekt die folgenden Bibliotheken erforderlich:

- OLED-Bibliotheken: Adafruit_SSD1306-Bibliothek und Adafruit_GFX-Bibliothek
- SHT20-Bibliothek

Auf der Hardware-Seite werden diese Teile benötigt:

- 2x ESP32-Entwicklungsboards
- 2x SHT20-Sensoren
- 2x 0,96 Zoll OLED-Displays

Die MAC-Adressen der beiden Boards können wieder mit dem aus dem ersten Artikel bekannten Programm (ESP_now_get_MAC.ino) ausgelesen werden. Nach dem Auslesen sollten die Adressen wieder umgehend in die MAC-Datenbank der Boards eingetragen werden.

Die Tabelle 1 kann als Referenz verwendet werden, um die SHT20-Sensoren und die OLED-Displays mit den Boards zu verdrahten. Beide Komponenten kommunizieren über den I2C-Bus.

Tabelle 1

SHT20	**ESP32**	**Leitungsfarbe in Bild 10**
VCC	3,3 V	rot
GND	GND	schwarz
SCL	GPIO 22	gelb
SDA	GPIO 21	blau
OLED Display	**ESP32**	**Leitungsfarbe in Bild 10**
GND	GND	schwarz
VCC	3,3 V	rot
SCL	GPIO 22	gelb
SDA	GPIO 21	blau

Die Verdrahtung für jeweils ein System ist in Bild 10 dargestellt, Bild 11 zeigt einen Aufbauvorschlag.

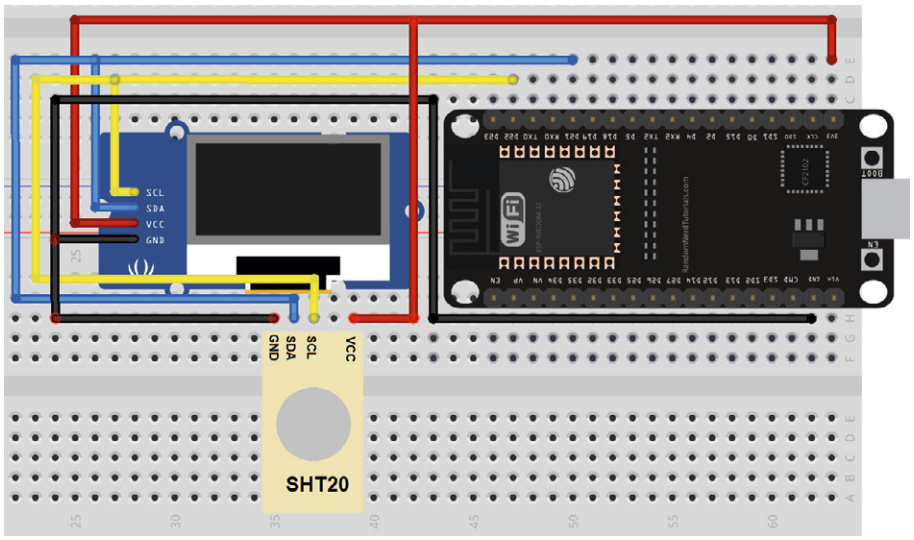


Bild 10: Sender/Empfänger für bidirektionale Datenübertragung

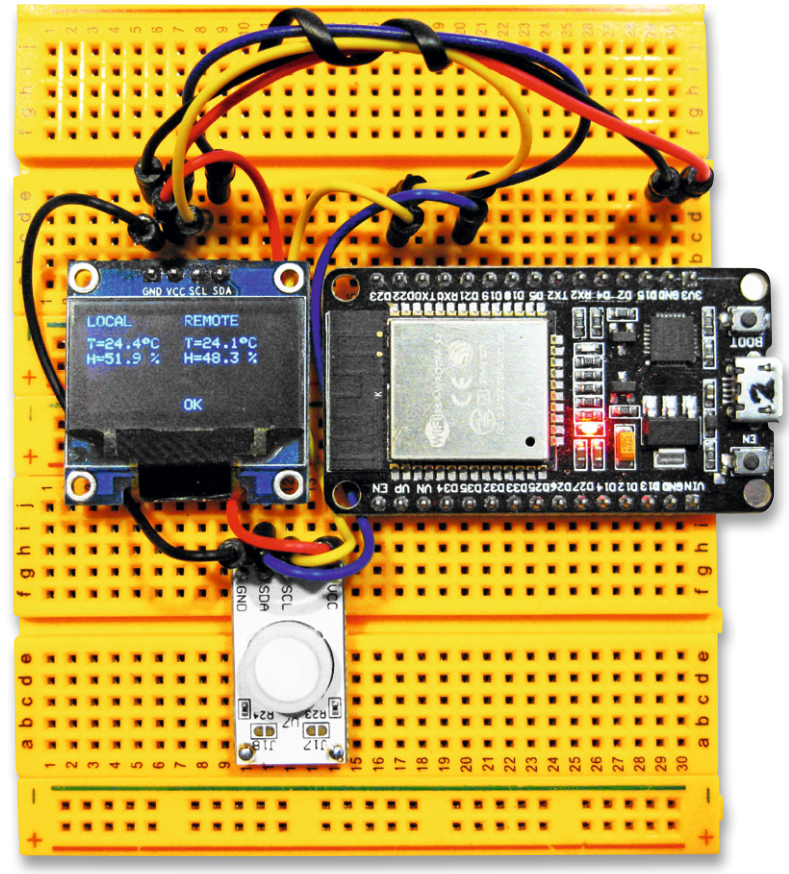


Bild 11: Sender/Empfänger für bidirektionale Datenübertragung

Das Programm ist für beide Einheiten identisch. Es muss lediglich jeweils die korrekte MAC-Adresse, also die Adresse des jeweiligen „Empfänger-Boards" in der Zeile

```
uint8_t broadcastAddress[]={0xFF,0xFF,0xFF, 0xFF,0xFF,0xFFC}; // INSERT RECEIVER ADDRESS HERE!
```

angegeben werden. Da das Programm bereits recht umfangreich ist, werden im Folgenden nur die wichtigsten Ausschnitte angegeben. Der vollständige Code findet sich im Download-Paket.

```
// Bi-directional.ino
// ESP32 @ IED 1.8.16

#include <esp_now.h>
...

uFire_SHT20 sht20;

#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 64  // OLED display height, in pixels

// SSD1306 display @ I2C (SDA->22, SCL->21 pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// MAC Address of receiver(s)
uint8_t broadcastAddress[]={0xFF,0xFF,0xFF, 0xFF,0xFF,0xFFC}; // INSERT RECEIVER ADDRESS HERE!


// Define variables to store readings
…

void setup()
{ Serial.begin(115200);
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  { Serial.println(F("SSD1306 allocation failed")); return; }
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {Serial.println("Error initializing ESP-NOW");return;}
  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0; peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK){Serial.println("Failed to add peer");return;}
  esp_now_register_recv_cb(OnDataRecv);
  sht20.begin();
}

void loop()
{ …
  getReadings();
  SHT20Readings.temp=temperature;
  SHT20Readings.hum=humidity;
  SHT20Readings.pres=pressure;

  // Send message via ESP-NOW

}
```

Nachdem die Programme mit der jeweils korrekten MAC-Adresse auf die Module geladen wurden, erscheinen sowohl die Daten des lokalen SHT20-Sensors als auch die drahtlos übertragenen Werte des zweiten Moduls auf dem OLED-Display.

Die Callback-Funktion erlaubt es auch hier, Rückinformationen der Gegenstation anzuzeigen. Sobald diese ausfällt (etwa durch eine Unterbrechung der Stromversorgung), wechselt die Anzeige in den Displays von „OK" auf „FAIL" (Bild 12).

Damit steht ein zuverlässiges und praxistaugliches System zur Verfügung, das die Übertragung von Sensorwerten auch über größere Entfernungen hinweg zuverlässig ermöglicht.

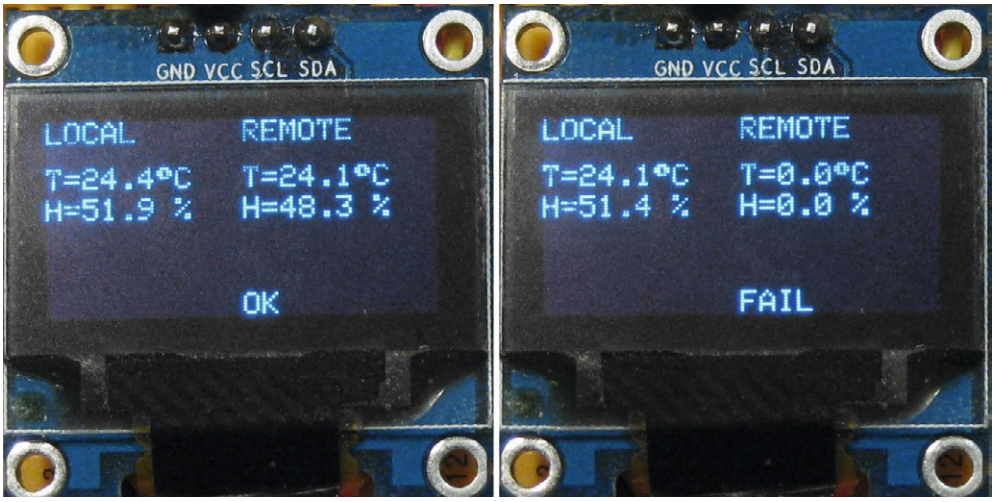


Bild 12: Ein Ausfall der Gegenstation wird durch „FAIL" angezeigt.

Ausblick

In diesem Artikel wurde gezeigt, wie das ESP-NOW-System in vielfältiger Weise in der Praxis eingesetzt werden kann. Vom einfachen Funkschalter über LED-Dimmer bis hin zur bidirektionalen Datenübertragung spannt sich dabei der Bogen möglicher Applikationen.

Allerdings wurden bislang immer nur Zweipunkt-Verbindungen mit einem „Sender" und einem „Empfänger" verwendet. Im dritten Artikel dieser Serie wird es um den Aufbau von kompletten Netzwerken gehen. Diese erlauben es, beliebige Werte in einem universellen Datennetzwerk auszutauschen. Dabei kann ein Sender an mehrere Empfänger Daten übertragen oder ein Empfänger Messdaten von verschiedenen Sensorstationen übernehmen. Aber auch der bidirektionale Datenaustausch unter allen beteiligten ESP-Boards ist möglich.

Anwendungen im Bereich des Internets der Dinge (Internet of Things - IoT) und der Heimautomatisierung sind damit kaum noch Grenzen gesetzt. ELV

Weitere Infos

[1] Download-Paket: Artikel-Nr. 253606

Material	Artikel-Nr.
Joy-IT Entwicklungsplatine NodeMCU mit ESP32	145164
Breadboard	251467
SHT20-Sensoren sind im Prototypenadapter-Set PAD 4 („Digital") enthalten	155107
OLED-Display	251189
Kleinmaterial (LEDs, Verbindungsdrähte, Potentiometer)	