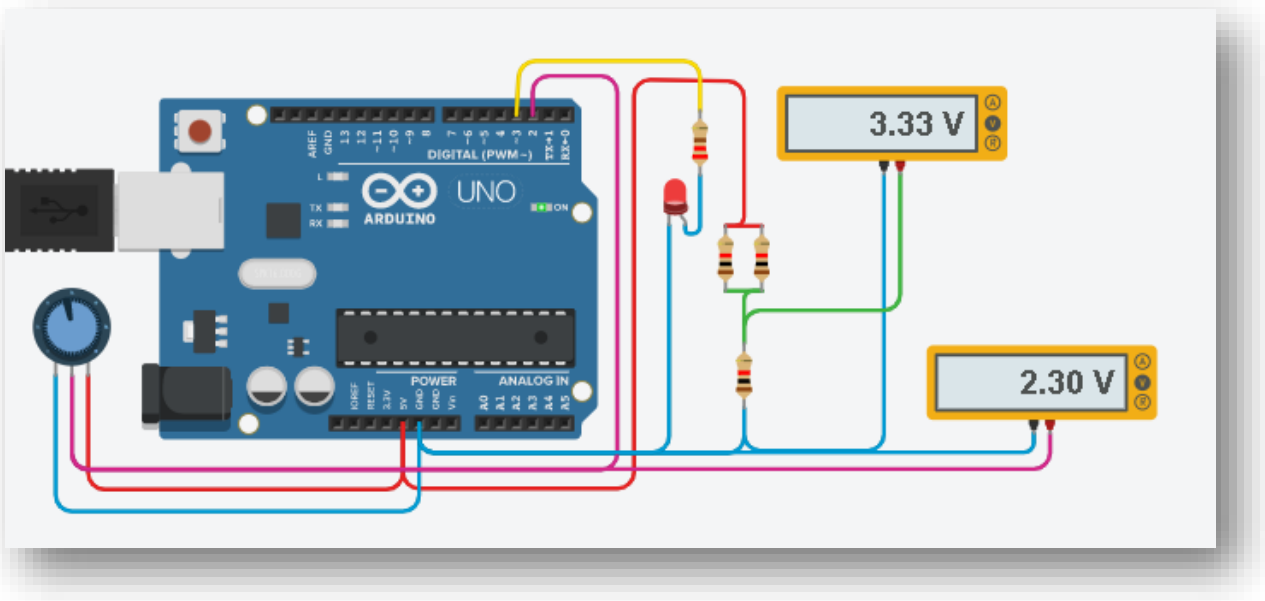
Serielle Kopplung von Arduino-Modulen mit ESP8266 / ESP32 und Datenübertragung per ESP Now

Die serielle Kopplung von Arduino- und ESP-Modulen ist eine sinnvolle Angelegenheit, wenn:

- die analogen und digitalen Pins der ESP-Module für die Lösung einer Aufgabe nicht ausreichen,
- ein Arduino sich mit dem WLAN verbinden soll,
- die Rechenleistung und/oder Speicher eines Arduino nicht ausreichend sind,
- die Programmierung des Arduino in C/C++ durch die Programmierung eines ESP-Moduls in MicroPython erweitert werden soll.

Ein Problem sind die verschiedenen Ein- und Ausgangspannungen von 5V bzw. 3.3V eines Arduino bzw. eine ESP-Moduls.

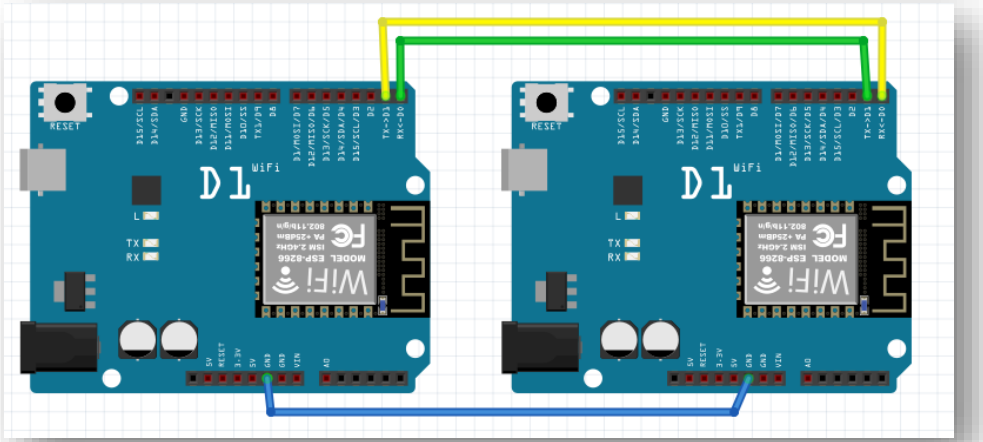
Problem: Spannungsteiler 5V auf 3.3V auf Tinkercad



Die Pegelanpassung vom 5V-Ausgang des Arduino auf einen 3.3V-Eingang eines ESP-Moduls kann mit einem Spannungsteiler aus drei gleichen Widerständen (im kOhm Bereich) sauber gelöst werden.

Umgekehrt liest ein Arduino-Eingang eine Eingangsspannung von ca. 2.5 V und höher als H-Pegel.
Der Ausgang eines ESP-Moduls liefert aber etwa 3.3V. Das ergibt sicher H-Pegel am Eingang des Arduino.

Kopplung zweier ESP8266 D1 Boards über die Serielle Schnittstelle (TxD/RxD)



ESP8266-Serial-Example-links.ino

```ino
// Serielle Kopplung zweier ESP8266-Module
//  (Eingang) RXD <- gelb <- TXD (Ausgang) des Partners
//  (Ausgang) TXD -> grün -> RXD (Eingang) des Partners

const byte led = 14;

void setup() {
  Serial.begin(9600);
  Serial.println("\nESP8266_Serail-Example-links");
  pinMode(led, OUTPUT);
}

void loop() {
  Serial.println("Hallo Rechts");
  Serial.flush();
  digitalWrite(led, not digitalRead(led));
  delay(2000);
}
```

ESP8266-Serial-Example-rechts.ino

```ino
// Serielle Kopplung zweier ESP8266-Module
//  (Eingang) RXD <- grün <- TXD (Ausgang) des Partners
//  (Ausgang) TXD -> gelb -> RXD (Eingang) des Partners

const byte led = 14;

void setup() {
  Serial.begin(9600);
  Serial.println("\nESP8266-Serial-Example-rechts");
  pinMode(led, OUTPUT);
}

void loop() {
  if (Serial.available()) {
    char ch;
    while (Serial.available()) {
      ch=Serial.read();
    }
    digitalWrite(led, not digitalRead(led));
    delay(500);
  }
}
```

Die Verbindung der Boards darf erst nach dem Aufspielen der Programme hergestellt werden. Das rechte Board blinkt nach dem Empfang der Daten vom linken Board kurz auf. Wird der Abstand der gesendeten Nachricht verändert oder wird die Übertragung unterbunden, ändert sich das Verhalten des rechten Boards entsprechend. Jede Programmänderung verlangt, diese Verbindung wieder zu entfernen, um die Kommunikation mit dem PC wieder zu ermöglichen.

Nun zu den Programmen:

Über die „serielle Schnittstelle“ und deren Nutzung kannst du unter Serial mehr erfahren.

Das linke Modul agiert als Sender der Nachricht „Hallo Rechts“, die regelmäßig aller 2 Sekunden über den seriellen Port gesendet wird. Nach jedem Senden, wird die Onboard LED ein- bzw. ausgeschalten. Die Daten werden in einen Sendepuffer geschrieben und versendet. Mittels **`Serial.Flush()`** wird gewartet, bis alle Daten aus dem Puffer gesendet werden. Der Puffer wird „geleert“ (flush).

Das rechte Modul arbeitet als Empfänger der Nachricht. Wenn Daten auf dem seriellen Port verfügbar (available) sind, werden diese solange zeichenweise abgeholt, bis keine Zeichen mehr vorhanden sind. Danach wird der Zustand der Onboard LED gewechselt und eine halbe Sekunde gewartet.
Warum zum Kuckuck wird die LED danach über sofort wieder zurückgeschalten? Dafür gibt es doch erst nach dem nächsten Empfang einer neuen Nachricht einen Grund.

Das muss noch geklärt werden.

Durch die Nutzung der seriellen Schnittstelle für die Kommunikation zwischen unseren beiden µControllern, haben wir aber keine Verbindung mehr zum PC.

Geben wir nun auf der Empfängerseite die „Hardware Serial“-Schnittstelle für den PC wieder frei, um auf dem Seriellen Monitor der Arduino IDE verfolgen zu können, was eigentlich empfangen wird. Wir richten auf dem Empfänger eine „Software Serial“-Schnittstelle ein.

Mehr zum Thema Serielle Datenübertragung:

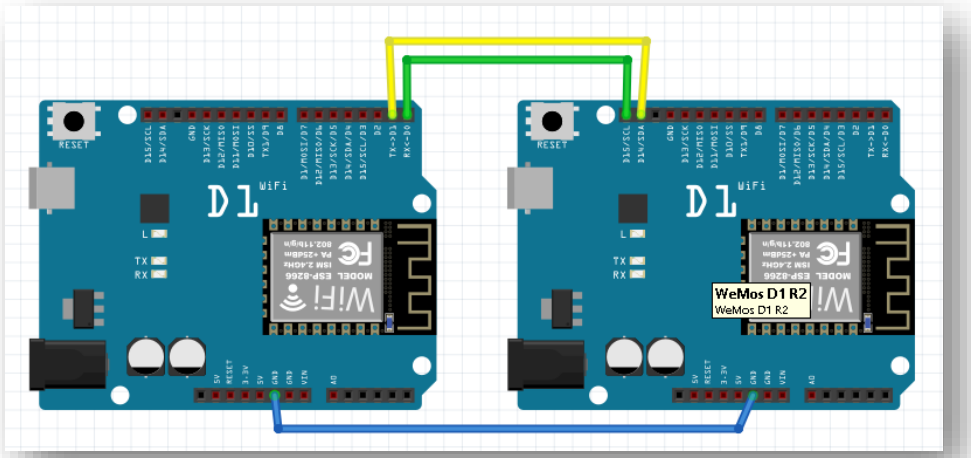
UART - Universal Asynchronous Receiver Transmitter

UART verstehen

Die UART Schnittstelle

Wie funktioniert UART – einfach erklärt!

SoftSerial – eine eigene serielle Schnittstelle




```
14 Bytes:
Hallo Rechts

9 Bytes:
Hallo Rec
5 Bytes:
hts

14 Bytes:
Hallo Rechts
```

```
14 Bytes:
14 Bytes:
7 Bytes:
7 Bytes:
14 Bytes:
```

```
14 Bytes:
0 Bytes:
0 Bytes:
0 Bytes:
14 Bytes:
0 Bytes:
0 Bytes:
5 Bytes:
9 Bytes:
0 Bytes:
14 Bytes:
0 Bytes:
```

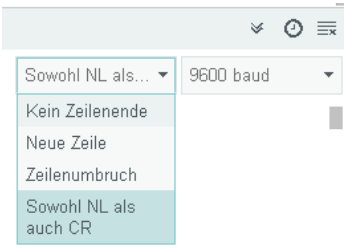
Je nach Programmänderung kann folgende Ausgabe erreicht werden.
Dabei tauchen aberhin und wieder kleine „Aussetzer“ auf.

ESP8266-SoftSerial-Example-rechts.ino

```
1  // Serielle Kopplung zweier ESP8266-Module - SoftSerial beim Empfänger
2
3  #include <SoftwareSerial.h>
4  #define swSerial_TX 5  // SCL/D15 - GPIO5  ->  grün  -> RXD der Gegenstelle
5  #define swSerial_RX 4  // SDA/D14 - GPIO4  ->  gelb  -> TXD der Gegenstelle
6  SoftwareSerial swSerial;
7
8  const byte led = 14;
9
10 void setup() {
11   Serial.begin(9600);
12   Serial.println("\nESP8266-SoftSerial-Example-rechts");
13   swSerial.begin(9600, SWSERIAL_8N1, swSerial_RX, swSerial_TX, false);
14   pinMode(led, OUTPUT);
15 }
16
```

```
17 void loop() {
18   if (swSerial.available()) {
19     Serial.println( String(swSerial.available()) + " Bytes:  ");
20     char ch;
21     while (swSerial.available()) {
22       ch=swSerial.read();
23       Serial.print(ch);
24     }
25     Serial.println();
26     digitalWrite(led, not digitalRead(led));
27     delay(500);
28   }
29   delay(400);
30 }
```

In dieser Programmversion ist das unerklärbare Verhalten der Onboard LED nicht mehr zu beobachten. Die Zeile 27 kann somit entfallen. Warum hat die Nachricht eine Länge von 14 Zeichen. Es wird doch „Hallo Rechts“ gesendet.
Das sind nur 12 Zeichen (mit dem Leerzeichen).
Die Zeilenende-Einstellung des Senders ist die Ursache

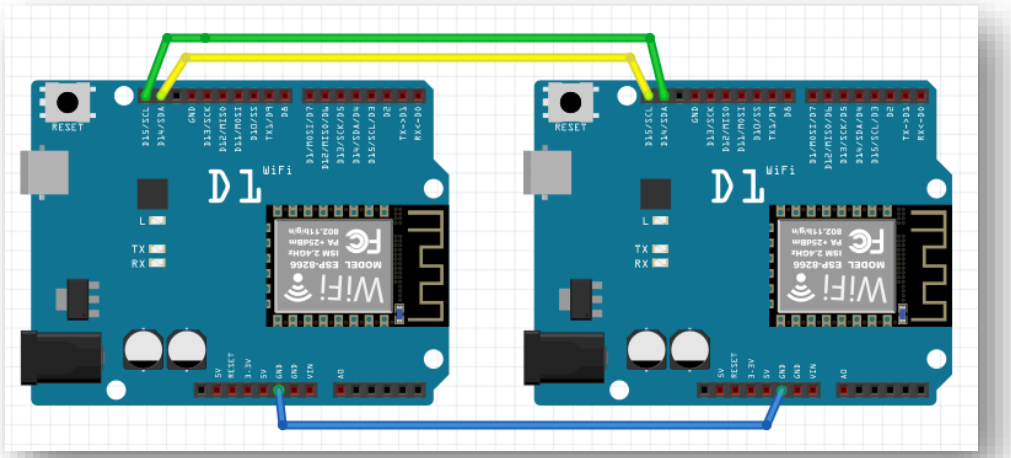


Softserial auf Sender und Empfänger nutzen

Auf beiden ESP8266-Modulen wird SoftSerial zum gegenseitigen Datenaustausch genutzt. Da die **Hardware Serial Ports** frei sind, können diese mit je einer Instanz der Arduino IDE mit dem PC per USB verbunden und deren Serieller Monitor genutzt werden.

Nun könen auf beiden Seiten über die Befehlszeile des Seriellen Monitors Daten eingeben, versendet und beim Epfänger ausgegeben werden.

Nun können jeweils auf der Empfängerseite diese Daten ausgewertet werden.



SoftwareSerialExample-links.ino

```ino
#include <SoftwareSerial.h>
//  TXD - GPIO5 - D15/SCL    -->   grün    ---> RXD
//  RXD - GPIO4 - D14/SDA    -->   gelb    ---> TXD
//                       RXD, TXD
SoftwareSerial mySerial(  4,  5 );

void setup() {
  Serial.begin(9600); Serial.println("\nSoftwareSerial-Test");
  Serial.println("Ich  bin links");

  mySerial.begin(9600);
  mySerial.println("\nHallo Rechts.");
}

void loop() {
  while (mySerial.available()) {   // Solange Daten im Empfangspuffer stehen,
    Serial.write(mySerial.read()); // werden diese gelesen und ausgegeben.
  }
  while (Serial.available()) {     // Solange Daten im Sendepuffer stehen,
    mySerial.write(Serial.read()); //werden diese gelesen und ausgegeben.
  }
}
```

SoftwareSerialExample-rechts.ino

```ino
#include <SoftwareSerial.h>
//  TXD - GPIO5 - D15/SCL    -->   gelb    ---> RXD
//  RXD - GPIO4 - D14/SDA    -->   grün    ---> TXD
//                       RXD, TXD
SoftwareSerial mySerial(  4,  5 );

void setup() {
  Serial.begin(9600); Serial.println("\nSoftwareSerial-Test");
  Serial.println("Ich  bin rechts");

  mySerial.begin(9600);
  mySerial.println("\nHallo Links.");
}

void loop() {
  while (mySerial.available()) {   // Solange Daten im Empfangspuffer stehen,
    Serial.write(mySerial.read()); // werden diese gelesen und ausgegeben.
  }
  while (Serial.available()) {     // Solange Daten im Sendepuffer stehen,
    mySerial.write(Serial.read()); //werden diese gelesen und ausgegeben.
  }
}
```

ESP32 D1 R32 als Sender per UART2 und ESP8266 als Empfänger testen

Der Sender ist per Thonny und der Empfänger per Arduino IDE mit PC verbunden.

```
            ESP32 D1 R32     ----       ESP8266 D1
                  Gnd        blau           Gnd
U2-RX - GPIO17 - IO16        grün      D15/SCL GPIO5 swTX
U2-TX - GPIO16 - IO17        gelb      D14/SDA GPIO4 swRX
```

Serial2-test.py

```python
# Serielle Verbindung zu einem ESP8266
# ESP32 sendet aller 2 Sekunden Daten
# ESP8266 als Empfänger auf SoftwareSerial Tx=5 und Rx=4
#
## https://www.engineersgarage.com/micropython-esp8266-esp32-uart/
import time
from machine import UART, Pin

uart = UART(2, baudrate=9600, tx=17, rx=16)

#led = machine.Pin(2,machine.Pin.OUT)
led = Pin(2,Pin.OUT)
while 1:
    uart.write('hello')
    time.sleep(2.0)
    led.value(not led.value())
```

```
□□□□□□□□□□□□□□□□□□□□□□□□□
hello
5 Bytes:
hello
5 Bytes:
hello
5 Bytes:
hello
5 Bytes:
hello
5 Bytes:
hello
```