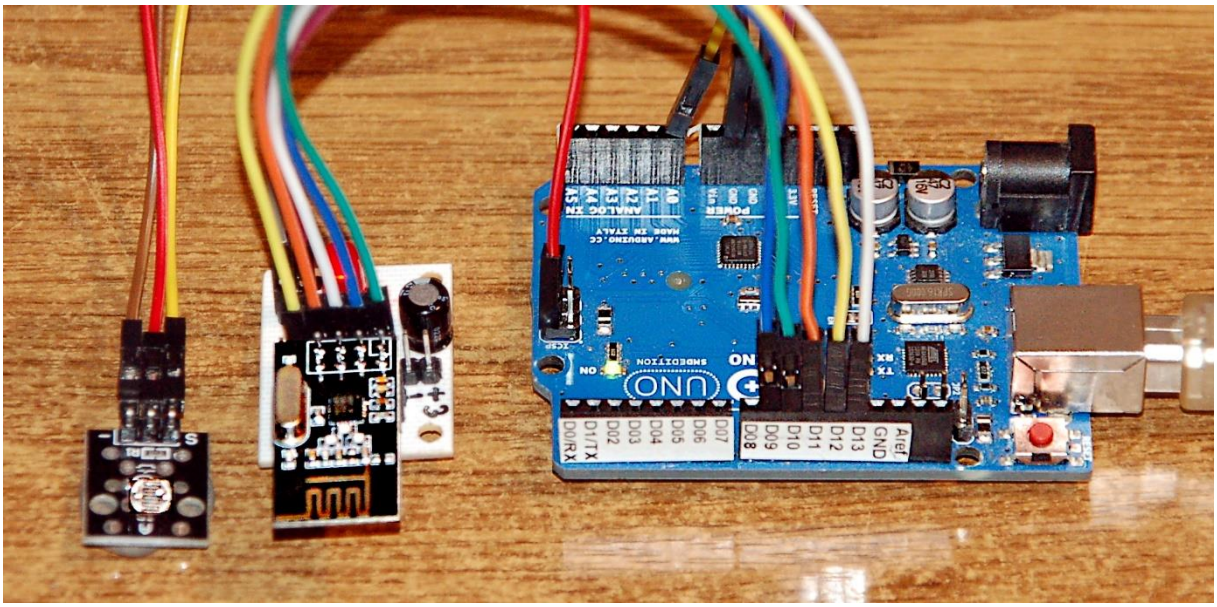

Abbildung 1: LDR, nRF24L01 und Arduino UNO

Diesen Beitrag gibt es auch als PDF-Datei zum Download.

MicroPython auf dem ESP32 und ESP8266

Heute ist Partytime angesagt. Die Arduinos werden fein gemacht, um sich gebührend beim ESP-Clan einzuführen. Wir sind natürlich ganz exklusive Paparazzi, die das Geschehen aus der ersten Reihe mitverfolgen. Willkommen zu

Arduino goes ESP8266/32 (Teil 2)

In der vorangegangenen Folge haben wir ESP32 und ESP8266 für den Empfang aufgehübscht. Heute sind die Arduinos dran. Alle mir bekannten AVR-Controller verfügen über einen SPI-Bus. Diese Verbindung kann genutzt werden, um die Chips nativ zu programmieren. Bevor ein Arduino über USB/RS232 von der Arduino-IDE aus programmiert werden kann, muss ihm ein Bootloader verpasst werden und genau das geschieht über den SPI-Bus. Die Boards, die wir heute verwenden wollen, besitzen diesen Bootloader bereits, daher können wir alle über die Arduino-IDE und den USB-Anschluss programmieren und den seriellen Monitor zur Steuerung und Ergebnisanzeige nutzen. Beim Arduino Pro mini ist dazu allerding noch ein USB-RS232 TTL-Adapter nötig. Den SPI-Bus benötigen wir, um den nRF24L01 an den Arduino anzudocken, so wie wir es in der letzten Folge beim ESP8266 getan haben.

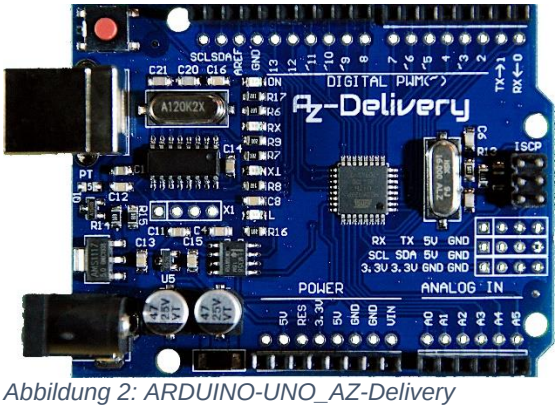

Abbildung 2: ARDUINO-UNO_AZ-Delivery

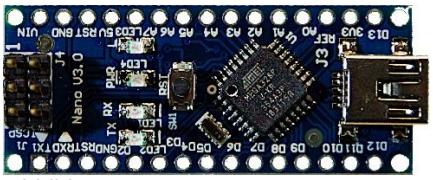

Abbildung 3: ARDUINO_NANO_V3

Abbildung 4: ARDUINO_PRO MINI

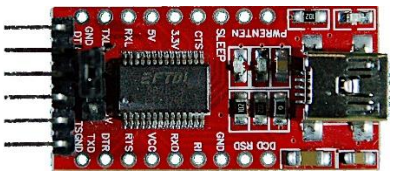

Abbildung 5: USB-RS232-Adapter _FTDI232RL

Die Hardware

Teile aus der ersten Folge

1	ESP8266 Node-MCU oder
1	ESP32 Node-MCU oder ESP32 D1 mini
1	FDTI-USB-RS232-TTL-Adapter
2	nRF24L01+
2	nRF24L10+ Breadboard-Adapter
1	LDR mit 10kΩ Trimmpoti oder
1	LDR-Modul
2	Minibreadboards

Teile für die aktuelle Folge

1	Arduino Uno oder Nano V3 oder Pro mini
1	Breadboard
diverse	Jumperkabel
2	passende USB-Kabel
1	evtl. Batterie 4,5V oder 5V-Steckernetzteil

Die Anschlussbelegungen für drei Arduino-Clones sind aus den folgenden Schaltbildern zu ersehen. Wählen Sie einfach nach Belieben eine Schaltung aus. Weil der Chip (ATMega 328P) bei allen der gleiche ist, gibt es auch keine Unterschiede im Programm.

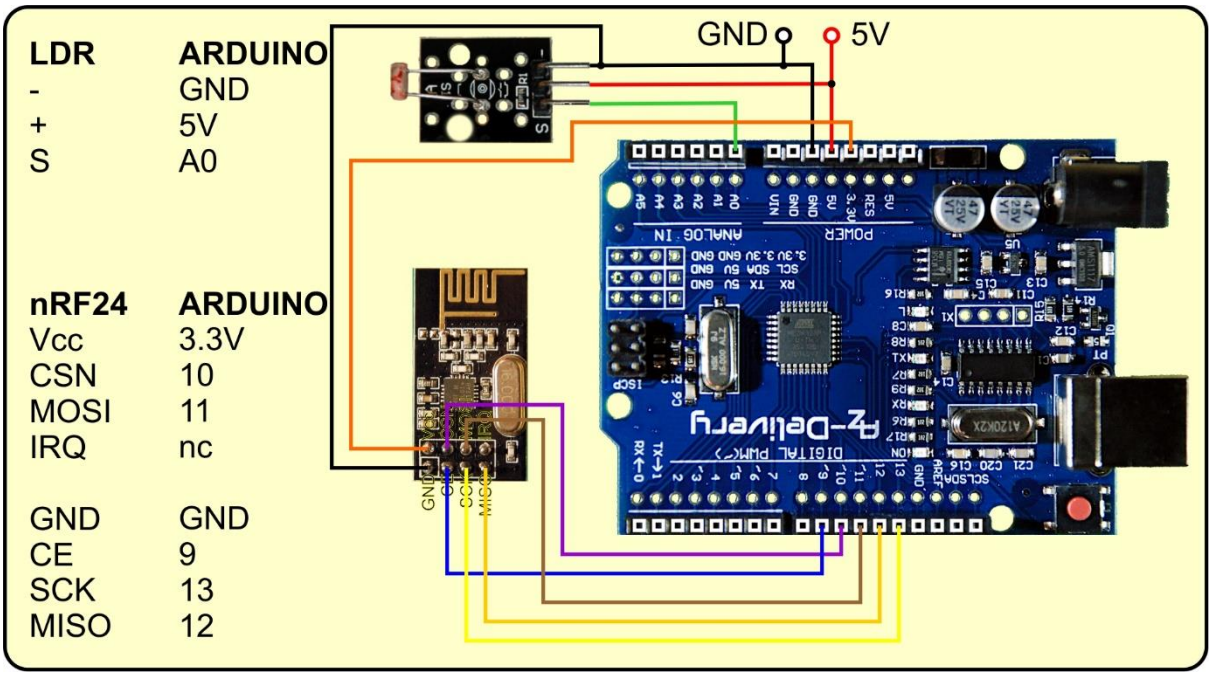

Abbildung 6: nRF24L10 am Arduino Uno-Clone

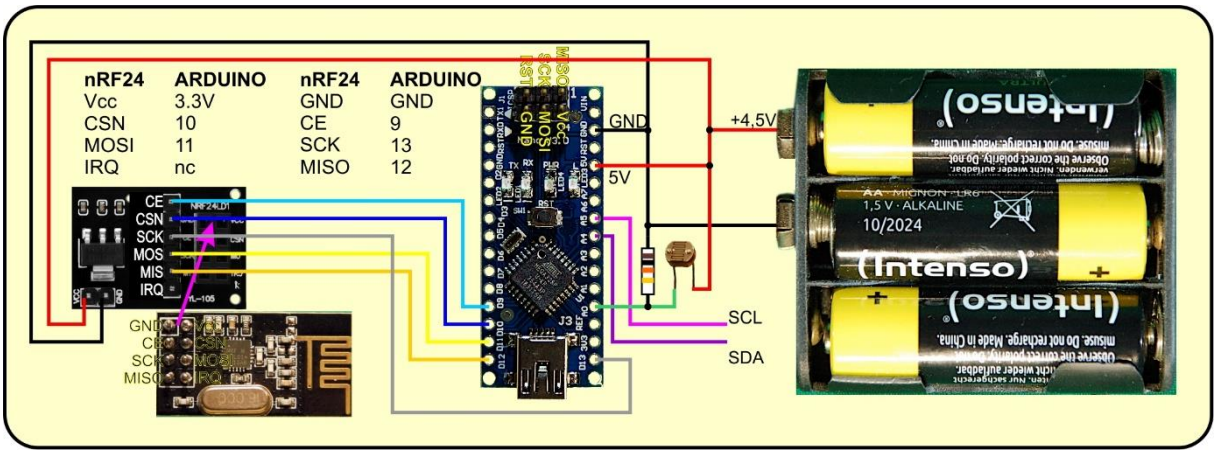

Abbildung 7: Schaltung mit Arduino Nano V3 - 5V-Versorgung

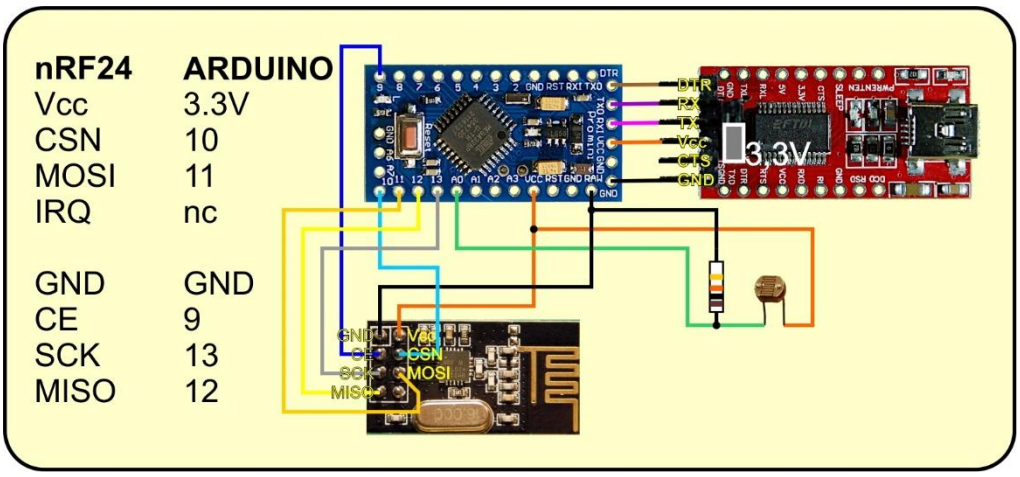

Abbildung 8: Schaltung mit Arduino Pro mini - 3,3V aus dem FTDI-Adapter

Bei der Energieversorgung aus dem USB-Anschluss über den FTDI-Adapter müssen wir beachten, dass der Jumper in der 3,3V-Position gesteckt ist, bevor wir den nRF24L01 anschließen. Das **nRF24L01-Board darf in keinem Fall direkt mit 5V** versorgt werden. Die Logikleitungen sind allerdings 5V-fest. Das nutzt die Schaltung in Abbildung 7 aus.

Die Pin-Nummern des SPI-Busses auf dem Arduino-Board können auch direkt auf diverse andere Boards der Familie übertragen werden. Die Anschlüsse bei den "Minis" befinden sich zum Beispiel jeweils an anderen Positionen auf dem Board, haben aber dieselben Nummern und dieselbe Funktion.

Die Software

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft

Für den Arduino
Arduino-IDE
Library für den nRF24L01+
Sketch für den nRF24L01+

Verwendete Firmware für den ESP8266/ESP32:

MicropythonFirmware
Bitte eine Stable-Version aussuchen

Die MicroPython-Programme zum Projekt:

nRF24-L01-Modul für den ESP8266/ESP32
master+slave.py: Demoprogramm für die ESP8266 RX- und TX-Funktion des nRF24L01

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung. Darin gibt es auch eine Beschreibung, wie die MicropythonFirmware (Stand 03.02.2022) auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, damit der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton, oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein kompiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

Einrichten der Arduino-IDE

Falls Sie noch nicht mit der Arduino-IDE gearbeitet haben und sich dieses Werkzeug noch nicht auf Ihrem Rechner befindet, folgen Sie bitte dieser Kurzanleitung.

Beginnen Sie mit dem Download der Installationsdatei über diesen Link.

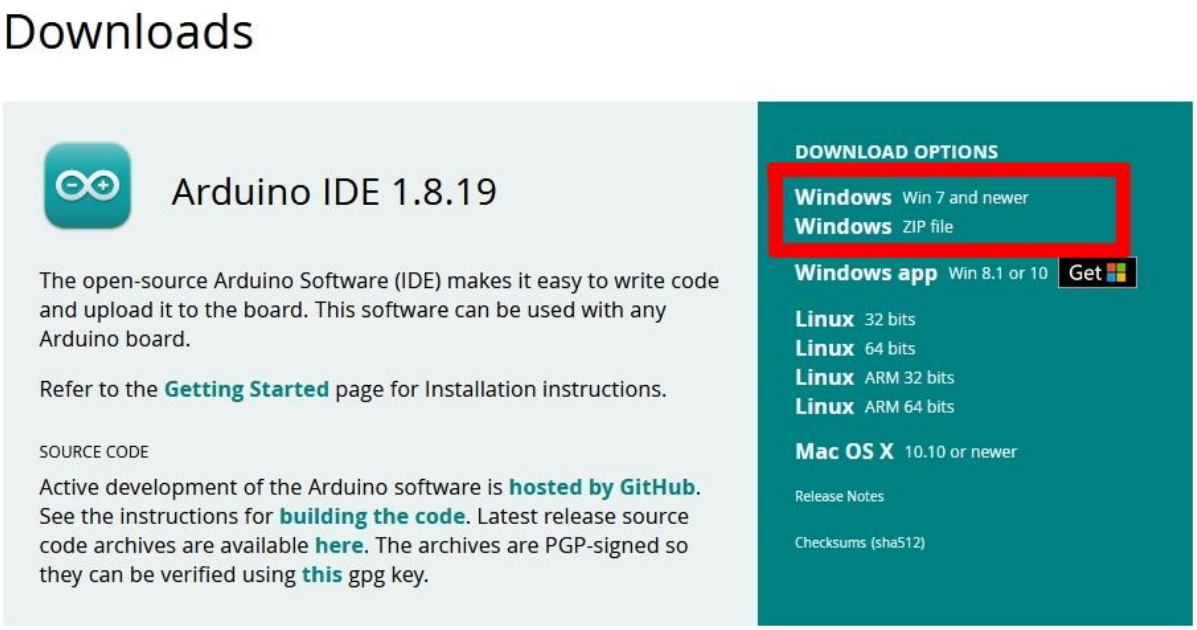

Abbildung 9: Arduino-IDE herunterladen

Klicken Sie die Version, die Ihrem Betriebssystem entspricht und speichern Sie die Datei in einem Verzeichnis Ihrer Wahl.

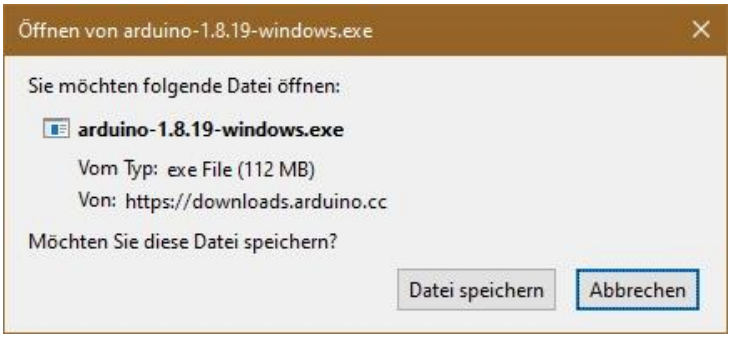

Abbildung 10: Installationsdatei speichern

Starten Sie die Installationsdatei und folgen Sie der Benutzerführung.

Im nächsten Schritt laden Sie bitte die Bibliotheksdatei RF24-master.zip direkt über diesen Link oder von der GitHub-Seite herunter und speichern Sie diese in einem Verzeichnis Ihrer Wahl.

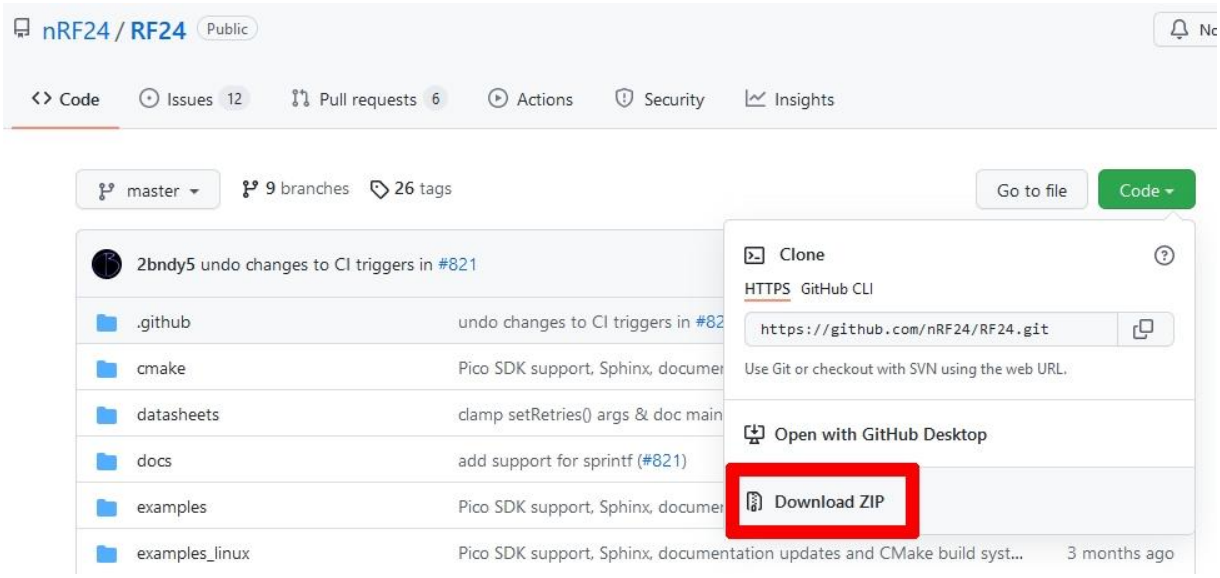

Abbildung 11: nRF24-Library herunterladen

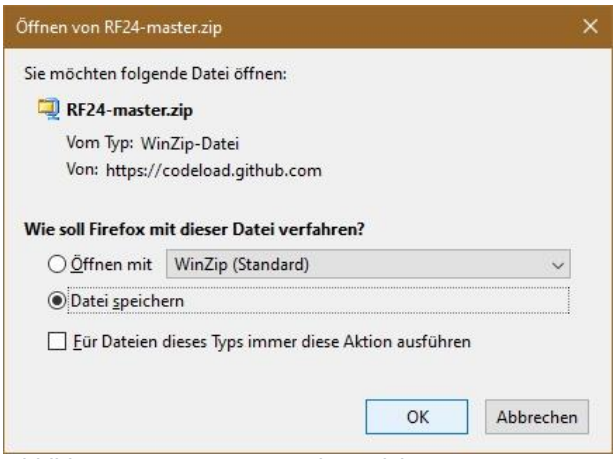

Abbildung 12: RF24-master.zip speichern

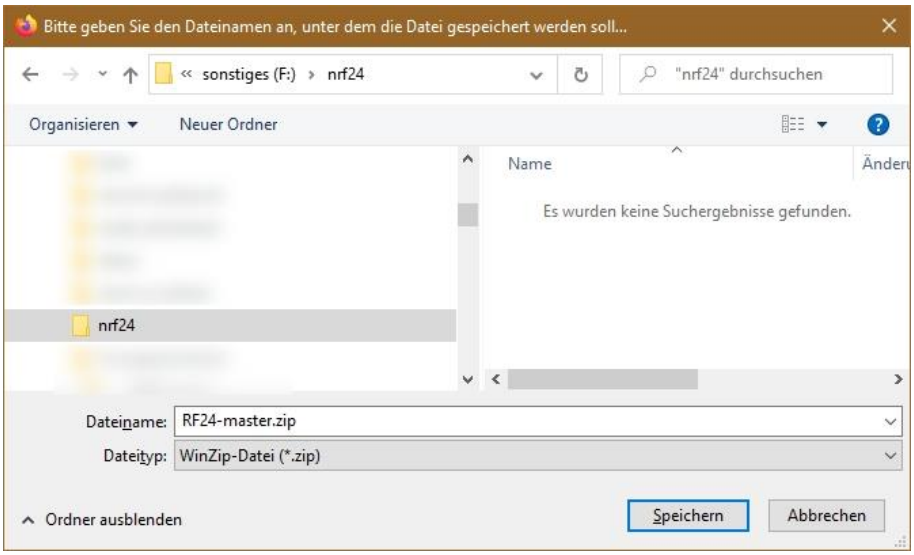

Abbildung 13: Verzeichnis auswählen

Starten Sie jetzt die Arduino-IDE, um die neue Bibliothek einzubinden. Im Menü **Sketch** wählen Sie dazu den Punkt **Bibliothek einbinden** und dann .**Zip-Bibliothek hinzufügen**.

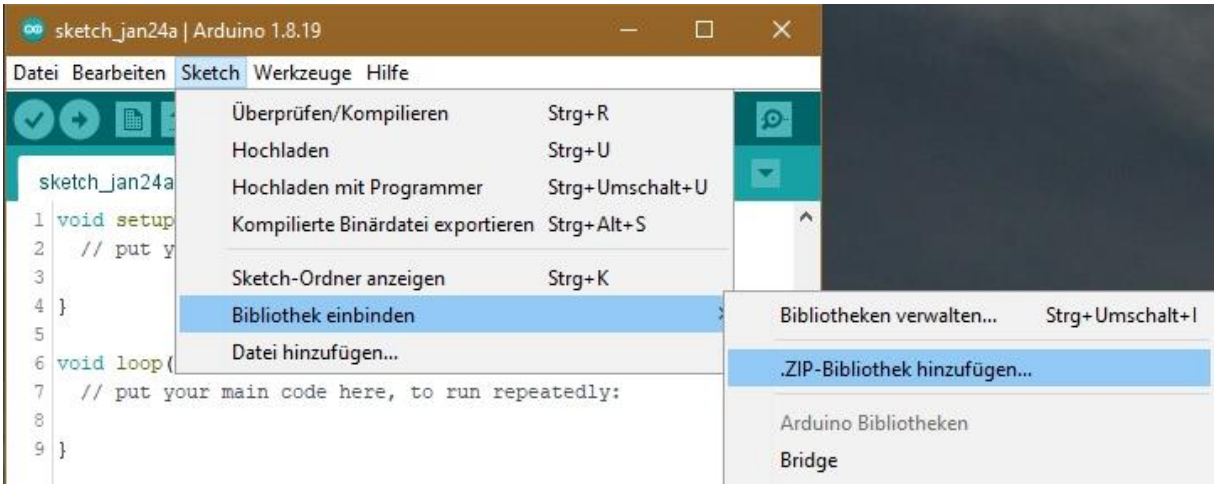

Abbildung 14: Bibliothek einbinden

Navigieren Sie zu der heruntergeladenen ZIP-Datei, markieren Sie diese und klicken Sie dann auf öffnen.

Abbildung 15: Library auswählen und öffnen

Der Sketch für den Arduino-Slave

Die hier vorgestellten Arduino-Varianten haben zwar den gleichen Controller an Bord, einen ATMega328P, aber dennoch muss jede der drei mit einer eigenen Einstellung bedient werden. Wichtig sind die richtige Auswahl des Boards, die Einstellung des Prozessors und natürlich des richtigen COM-Ports.

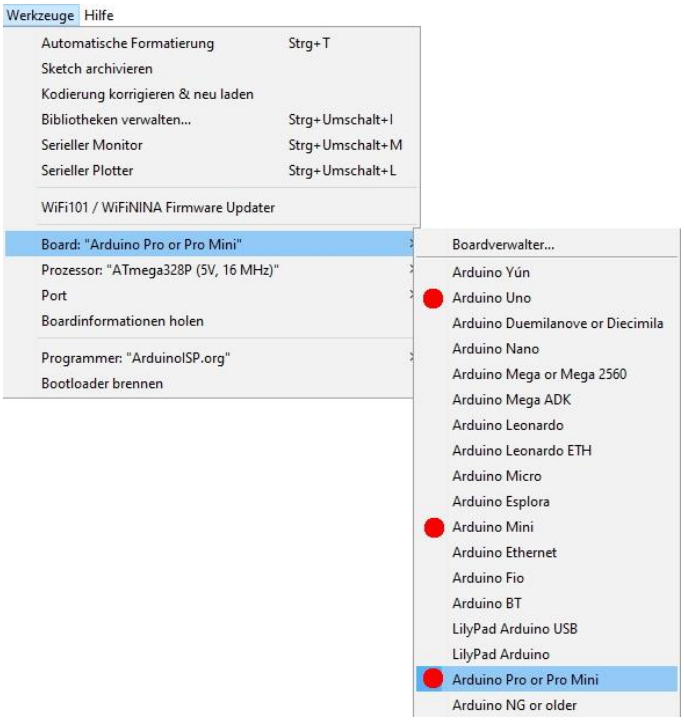

Abbildung 16: Arduino Pro mini - Boardauswahl

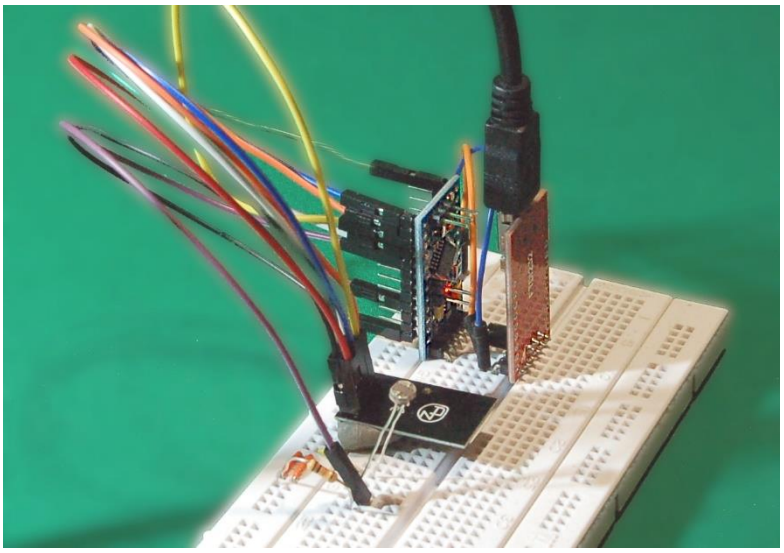
Abbildung 17: Arduino Pro mini mit FTDI-Adapter

Die Interfacepins des Pro mini passen gegengleich zu jenen des FTDI-Adapters, sodass die beiden Boards direkt aufeinander oder, wie hier face to face, auf ein Breadboard gesteckt werden können. Die Leitungen RXD und TXD sind dadurch bereits gekreuzt.

Das waren dann auch schon die Vorbereitungen für die Arduino-IDE. Wir schließen jetzt den Arduino an eine freie USB-Schnittstelle an und laden den Sketch in den Editor.

Lassen Sie uns das Programm näher anschauen, bevor wir es starten. Es gliedert sich in drei wesentliche Teile.

Includes und Deklarationen

```
#include <SPI.h>
#include "RF24.h"
#include "printf.h"
RF24 radio(9,10); // CE=9; CSN=10
uint8_t address0[]={0x54, 0x5A, 0x5A, 0x5A, 0x5A};
uint8_t address1[]={0x52, 0x5A, 0x5A, 0x5A, 0x5A};
uint8_t kanal=50;
const int PLLength = 8;
char payload[PLLength];
bool radioNumber=1; //
uint16_t ldr=0; //Integerwert vom LDR
int ldrPin = A0;
```

Wir importieren zuerst drei Bibliotheken, für die Bedienung des SPI-Busses, die Ansteuerung des nRF24L01+ und für die Ausgabe seiner Konfigurationsdaten.

Der Konstruktoraufruf der Klasse RF24 erzeugt eine nRF24L01-Instanz, die wir **radio** nennen. Damit unterschiedliche Einheiten angesprochen werden können, braucht jede Einheit zwei eindeutige Adressen, eine für das Senden und eine für den Empfang. Es reicht, wenn sich die Adressen in einem Byte unterscheiden. Weitere Informationen zur Adressvergabe können Sie in der vorangegangenen Episode näher studieren. Adressen dürfen zwischen drei und fünf Bytes enthalten. Wir hatten beim ESP8266 eine Adresslänge von 5 Bytes benutzt und tun hier desgleichen. Allerdings sind im MicroPython-Programm master+slave.py die Adressen als 10-stellige Hexadecimal-Werte definiert, während im Arduino-Sketch dafür Byte-Arrays der Länge 5 verwendet werden.

In dem Array steht das niederwertigste Byte an der Position 0, die auch als erste bei der Übertragung an den nRF24L01 über den SPI-Bus geht. Der Kanal 50 wird unser Nachrichtenkanal. Möglich sind Werte zwischen 0 und 125. Falls es im Betrieb zu Übertragungsproblemen kommt, kann es daran liegen, dass der gewählte Kanal auch von anderen Stationen genutzt wird, zum Beispiel von WLAN-Geräten, die das gleiche Frequenzband von 2,4 GHz benutzen. Dann ist es ratsam zu höheren Kanälen auszuweichen. Die beteiligten Stationen müssen aber denselben Kanal verwenden.

Als Nutzlast oder Payload dürfen wir bis zu 32 Zeichen übermitteln. Wir legen die Anzahl hier auf 8 fest und erzeugen ein char-Array dieser Länge als Puffer-Speicher. Der Arduino arbeitet als Slave. Er bekommt Aufträge vom ESP8266 und sendet Messwerte als Antwort zurück. Den LDR haben wir an **A0** angeschlossen. Dessen Wandlerwerte nimmt die 16-Bit-Variable **ldr** auf.

Konfiguration des Arduino

```
void setup() {

  Serial.begin(115200);
  while (!Serial) {
```

```
  }
  if(!radio.begin()){
    Serial.println(F("nRF24 not found"));
    while(1){}  // Halt on error
  }
  Serial.println(F("nRF24 started in Slave mode on
channel:"));
  radio.setChannel(kanal);
  Serial.println(kanal);
  radio.setPALevel(RF24_PA_LOW);
  radio.setAddressWidth(5);
  radio.setPayloadSize(PLLength);
  radio.openWritingPipe(address1);
  radio.openReadingPipe(1,address0);
  radio.startListening();
  printf_begin();
  radio.setDataRate(RF24_250KBPS);
  radio.printPrettyDetails();
}
```

Und schon sind in der Setup-Abteilung angelangt. Die serielle Schnittstelle für den Monitor wird eingerichtet. Wir versuchen, den nRF24L01+ zu starten. Hat das geklappt, dann bekommen wir die Startmeldung im seriellen Monitor. Andernfalls fährt sich das Programm in der while-Schleife fest.

Dann werden die Arbeitsparameter an den nRF24L01+ geschickt, die Kanalnummer, die Anzahl von Adressbytes und die Anzahl Bytes für die Nutzlast, wir senden und empfangen stets 8 Zeichen. Ein Master kann bis zu sechs Verbindungen zu Slaves auf einem Kanal unterhalten. Diese Verbindungen heißen Pipes. Unsere Sendeleitung legen wir auf Adresse 1, die Empfangsleitung Pipe 1 auf Adresse 0. Damit haben wir die Adressen zum Master auf dem ESP8266 gekreuzt. Der ESP8266 sendet auf Adresse 0, der Arduino muss also auf einer seiner Empfangspipes, hier die Pipe 1, auf dieser Adresse lauschen. Der Arduino sendet auf Adresse 1, weil der ESP8266 auf der Eingangsleitung 1 auch auf Adresse 1 lauscht. Die Sendeleistung drosseln wir auf niedrigstes Niveau, weil unsere beiden Stationen während der Tests sehr nahe aneinander liegen und zu viel Sendeleistung den Empfangsteil übersteuern kann, was zu Störungen führt.

Nachdem wir den nRF24L01+ angewiesen haben, am Radio zu lauschen, richten wir die Ausgabeeinheit ein, stellen die niedrigste Übertragungsrate (für beste Empfängerempfindlichkeit) auf 250kBit/s und lassen uns die gesamte Konfiguration am seriellen Monitor ausgeben.

Die Hauptschleife macht den Job

```
void loop() {
  uint8_t pipe;
  if (radio.available(&pipe)){
    radio.read(&payload,PLLength);
    Serial.print(F("Empfangen:"));
```

```
    Serial.print(PLLength);
    Serial.print(F(" Bytes on Pipe: "));
    Serial.print(pipe);
    Serial.print(F(": "));
    Serial.println(payload);
    for (int i=0; i<8;i++){
      Serial.println(payload[i],DEC);
    }
    String recv=(String((char *)payload));
    recv.trim();
    int colonAt=recv.indexOf(":");
    String command=recv.substring(0,colonAt);
    String rest=recv.substring(colonAt+1);
    // int val=rest.toInt();
    if (command=="send"){
      radio.stopListening();
      ldr=analogRead(A0);
      Serial.println(ldr);
      String wert = rest+":"+String(ldr);
      char reply[PLLength]={0,0,0,0,0,0,0,0};
      wert.toCharArray(reply,PLLength);
      bool report = radio.write(&reply, PLLength);
      Serial.println(report);
      radio.startListening();
    }
  }
}
```

Die Hauptschleife prüft, ob ein Paket mit Daten angekommen ist. Wenn ja, wird die Nutzlast in das char-Array **payload** eingelesen. Wir erhalten einige Meldungen mit den Empfangsdaten. Die folgenden print-Befehle und die for-Schleife dienen nur der Evaluierung während der Entwicklung und können im späteren Produktionssystem weggelassen werden.

Wir wandeln jetzt das char-Array in einen String um, entfernen nichtdruckbare Zeichen und prüfen auf den Text des angekommenen Befehls. Dazu suchen wir nach der Position des ":", denn der trennt den Befehl von der nachfolgenden Seriennummer. Alles davor ist Befehl, alles danach Seriennummer. Gegebenenfalls können wir die Seriennummer auch in eine Ganzzahl umwandeln, um daraus zur Absicherung des Transfers eine Prüfziffer oder Verschlüsselung zu erzeugen.

War der Befehl "**send**", dann beenden wir das Zuhören, lesen den analogen Wert an A0 und formen ihn in einen String um. In das gelöschte char-Array **reply[]** lassen wir die Seriennummer und, getrennt durch einen ":", den Inhalt des LDR-Wert-Strings übertragen, um die Daten im folgenden Befehl an den Master zu senden. Ist der Inhalt von **report** gleich 1, dann hat die Sendung geklappt und wir gehen wieder auf Lauschposten. Wie oder ob die Antwort beim Master (ESP8266) bearbeitet wird, überlasse ich gerne Ihnen. Ideen wären zum Beispiel das Versenden ins Netz oder Schreiben in eine Log-Datei.

Ist alles am Arduino aufgebaut und angeschlossen? Ist der korrekte USB-Port zum Arduino eingestellt und das richtige Board ausgewählt? Dann können wir den Sketch compilieren und hochladen. Der Arduino läuft als Slave und sollte in der Lage sein, Anfragen vom Master empfangen und beantworten zu können.

Der ESP8266 als Master

Damit der Arduino das kann, brauchen wir die Ausführung des Master-Programms auf dem ESP8266. Dafür gibt es zwei Möglichkeiten. Entweder ein ESP8266 ist bereits nach den Angaben aus der vorangegangenen Episode des Blogs vorbereitet und startklar oder es muss eine Station gemäß den Ausführungen dieser Folge erstellt werden. Für den letzteren Fall, hier eine kurze Anleitung.

Die Schaltung für den Master zeigt Abbildung 16. Für den Fall, dass der Master außer der Bedienung des nRF24L01 noch andere Dinge erledigen soll, ist statt dem ESP8266 vielleicht besser ein ESP32 geeignet. Programm und Treibermodul sind für den großen Bruder identisch.

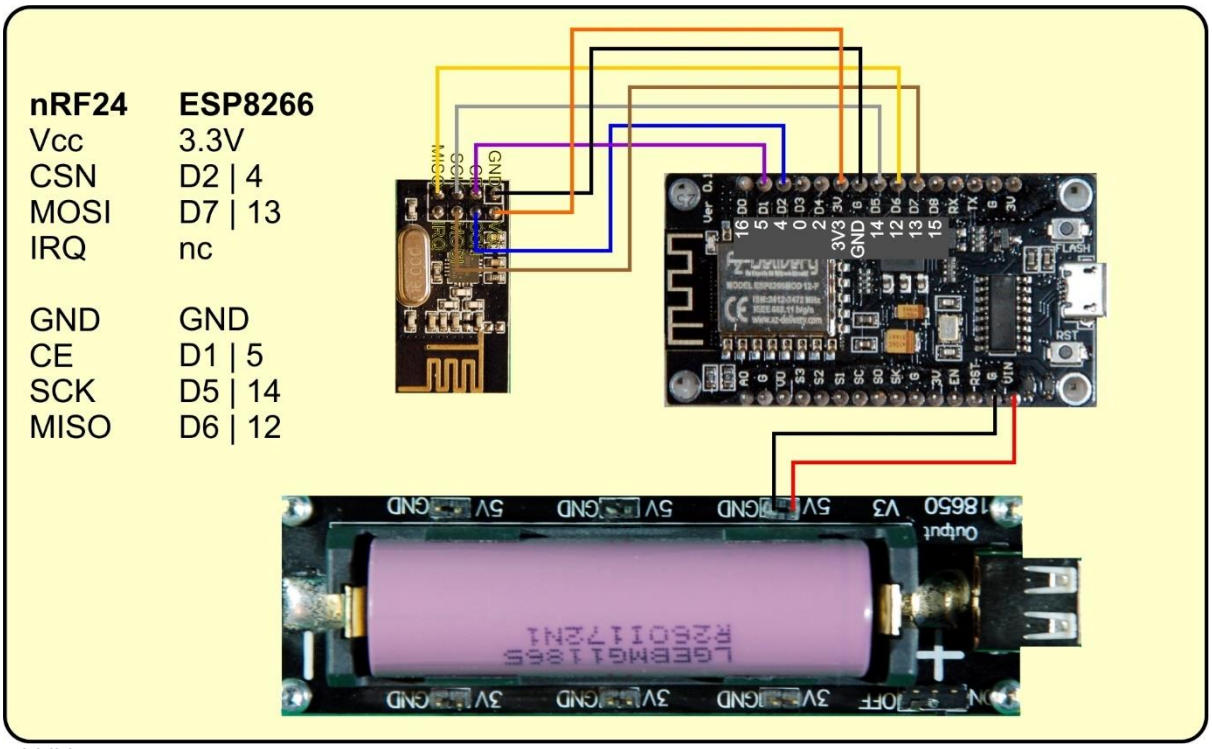

Abbildung 18: nRF24L10 am ESP8266

Mittels Thonny wird der ESP8266/ESP32 mit der MicroPython-Firmware versehen. Eine Anleitung dazu finden Sie im Kapitel **MicroPython - Sprache - Module und Programme** weiter oben in diesem Beitrag oder sie folgen einfach gleich diesem Link.

Ist das erledigt, laden wir die Datei **nrf24simple**.py zum ESP8266 hoch und öffnen das Programm **master+slave**.py in einem Editorfenster von Thonny. Nachdem es mit F5 gestartet wurde, werden fünf Anfragen an den Arduino gesendet, dann gibt es die Kontrolle an den MicroPython-Prompt ">>>" im Terminalbereich von Thonny

zurück. Solange der Arduino noch nicht läuft, werden wir Timeout-Meldungen erhalten, die wir im Moment getrost ignorieren können.

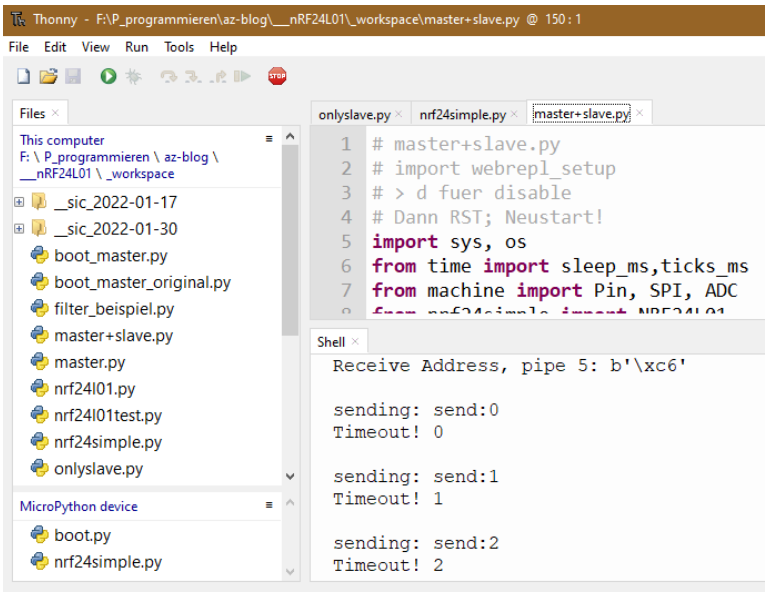

Abbildung 19: Erster Start des ESP8266 als Master

In der Arduino-IDE starten wir jetzt den seriellen Monitor. Er empfängt die Ausgaben unseres Programms auf 115200 Baud.

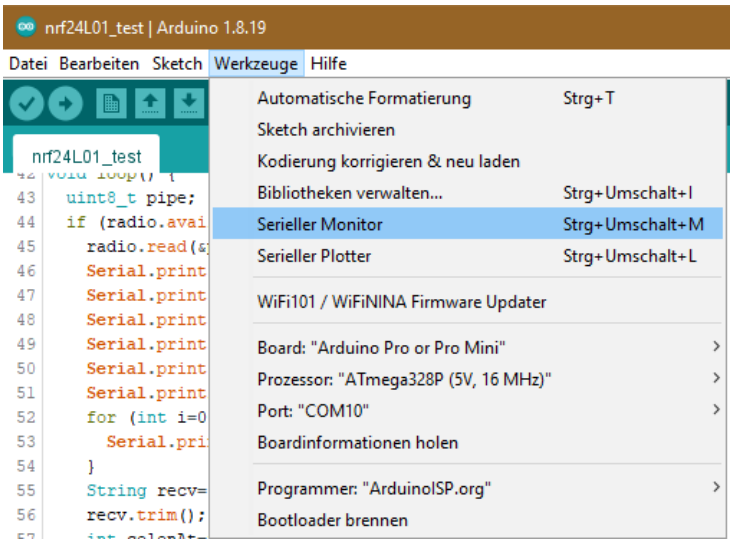

Abbildung 20: Den seriellen Monitor in der Arduino-IDE starten

Nach ein paar Sekunden erhalten wir die Startmeldungen von unserem Sketch.

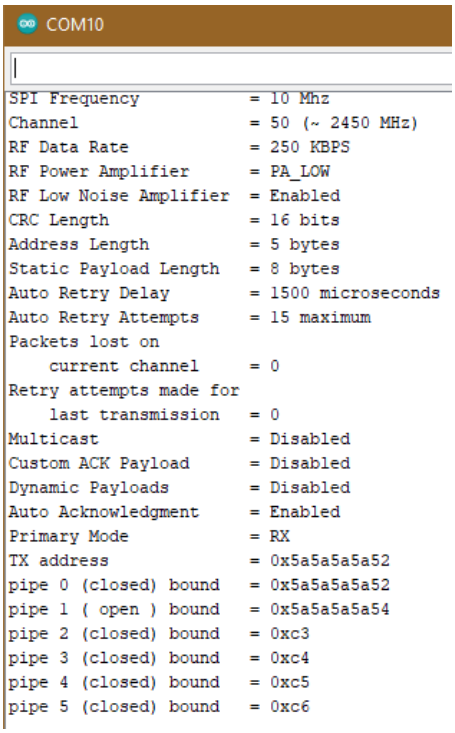

Abbildung 21: Startmeldungen des Arduino-Slaves

Der Slave ist jetzt empfangsbereit, starten wir also mit F5 das Masterprogramm in Thonny.

Im seriellen Monitor erscheinen in 2-Sekunden-Abständen die Anfragen des ESP8266 in aufgedröselter Form mit dem LDR-Wert in der vorletzten Zeile. Die 1 sagt uns, dass die Rückmeldung an den Master erfolgreich den Slave verlassen hat.

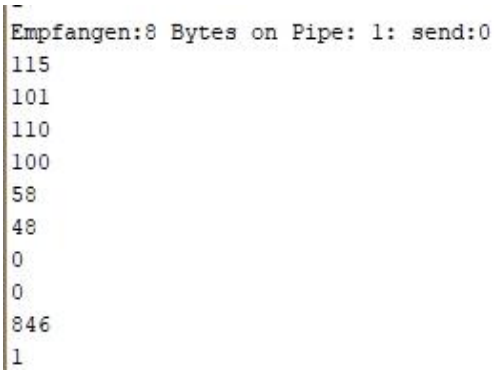

Abbildung 22: Output im seriellen Monitor beim Slave

Auch im Terminal von Thonny tauchen die Werte ohne bemerkbare Verzögerung auf.

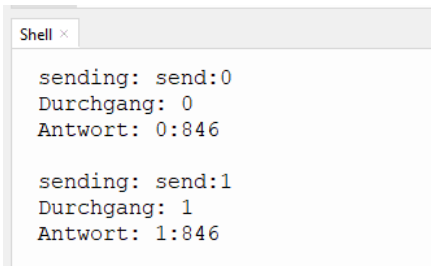

Abbildung 23: Shell-Output in Thonny

In einzelnen Fällen wird die Anforderung vom Master zwar versandt, kommt aber am Slave nicht oder verstümmelt an, sodass in der Folge auch keine Rückmeldung verschickt wird. Das äußert sich dann beim Master durch einen Timeout. Das Programm kann sich dann entsprechend um den Fehler kümmern, indem es zum Beispiel sofort eine weitere Anfrage sendet.

Damit sind wir am Ende des zweiteiligen Posts zum nRF24L01 angekommen, fassen wir zusammen.

- Der nRF24L01 stellt eine Funkverbindung zwischen Arduino und ESP8266/32 zur Verfügung, die abseits der üblichen WLAN-Frequenzen liegt.
- Durch das SPI-Interface ist der nRF24L01 für beide Controllerfamilien verwendbar.
- Das SPI-Interface auf dem ESP32/ESP8266 ist auch bequem im MicroPython programmierbar.
- Bestehende Arduino-Projekte lassen sich über ESP32 und ESP8266 einfach mit WLAN-Funktionalität erweitern.
- Ein ESP32 oder ESP8266 kann bis zu 6 logische Verbindungen zu Arduino-Projekten oder nRF24L01-Einheiten in Verbindung mit anderen Controllern verwalten.

Für uns ergeben sich damit weiterführende Grundlagen für spannende neue Projekte. Vielleicht finden Sie ja auch schon etwas in der Übersicht der bereits auf AZ-Delivery erschienen Beiträge zum Thema MicroPyton und ESP32/ESP8266.