

Manual IoT-Education- Board

Version 1.5

07.01.2022

Thomas Jäggi - [thomas.jaeggi\(at\)m121.ch](mailto:thomas.jaeggi(at)m121.ch)



Inhaltsverzeichnis

Inhaltsverzeichnis	2
1 Vorbemerkung	4
2 Full-Featured IoT-Education-Board	4
3 Einleitung	5
4 Board-Übersicht	6
5 Arduino IDE	8
5.1 Vorbereitung	8
5.2 Installation Arduino IDE	9
5.3 Einstellungen in der Arduino IDE	9
6 Inbetriebnahme des Boards	12
6.1 Komponenten ein- und ausschalten	12
7 Vorbemerkungen zu den Demo-Sketches	15
7.1 Allgemeines	15
7.2 Arduino Libraries	16
8 Demo-Sketches	18
8.1 LED blinken	18
8.2 LED dimmen	19
8.3 Analoges Signal von Potentiometer lesen	20
8.3.1 Ausgabe im Serial Monitor	20
8.3.2 Ausgabe im OLED Display	21
8.4 Buzzer mit Button aktivieren	23
8.4.1 Buzzertyp MF-FMD Low Level Trigger	23
8.4.2 Buzzertyp passiv	25
8.5 Rotary Encoder	26
8.6 NeoPixel-Ring mit Rotary Encoder steuern	30
8.7 NeoPixel-Choreografien	32
8.8 Buttons UP, DOWN, LEFT, RIGHT und SELECT auswerten	35
8.9 Motor 1: Typ JGA25-370 oder N20	38
8.10 Motor 1: Hall-Phasensignale PhA und PhB auswerten	40
8.11 Motor 2: Lüfter mit Propellermodul L9110	41
8.12 Motor 3: Schrittmotor 28BYJ-48 und ULN2003 Motor Driver	44
8.13 Analoger Temperatursensor TMP36	46
8.13.1 TMP36 mit ADC ADS1115 auslesen	48
8.14 Digitaler Feuchtesensor DHT22-AM2302	52
8.14.1 Digitaler Feuchtesensor DHT22-AM2302 - Ausgabe auf OLED-Display	54
8.15 Digitaler Temperatursensor DS18B20	56
8.16 Distanzsensor HC-SR04	58
8.17 Bewegungssensor HC-SR501	60
8.18 LDR - Fotowiderstand	63
8.19 Luftdrucksensor BMP280	67
8.20 Luftdrucksensor BMP388	69
8.21 Luftdrucksensor BME680	73
8.22 RTC DS3231 - Echtzeituhr	73



8.23 RFID RC522	76
8.24 Lichtsensor BH1750	82
8.25 IMU MPU9250-9-DOF	84
8.26 Infrarot LED - Infrarot Diode	87
8.27 TSOP4838-IR-Empfänger und Remote Control	87
8.28 NRF24L01-2.4GHz Transceiver	87
8.28.1 Einfache Meldung über Funk senden und empfangen	89
8.28.2 ESP32 als Funkempfänger und Gateway ins Internet	91
8.29 NI1000/PT1000 - OpAmp LM358	99
8.30 ESP32 als Webserver	99
9 Der M5StickC-Plus als Oszilloskop	103
10 Demo-Sketches M5StickC Plus	105
10.1 Watch	105
10.2 Stopwatch	106
10.3 Tetris Game	106
11 Anhang	107
11.1 Verwendete I ² C-Adressen auf dem IoT-Education-Board	107
11.2 GPIO-Map ESP32-Devkit-Lipo	107
11.2.1 Funktionen und verwendbare GPIOs	108
11.3 GPIO-Map M5StickC Plus	113



1 Vorbemerkung

Dieses Manual ist keine Anleitung in die Programmiersprache C oder C++. Dazu sei auf den Unterricht oder auf die E-Books von [RANDOM NERD TUTORIALS](#) verwiesen. Weiter ist auch das Buch [Mikrokontroller ESP32 - Das umfassende Handbuch](#) von Udo Brandes zu empfehlen.

Hinter den blauen Texten verbergen sich jeweils Links ins Internet. Rote Texte verweisen auf Stellen innerhalb dieses Dokumentes.

Mit den Demo-Sketches (siehe Abschnitt 8) kann die grundsätzliche Funktionsweise von Sensoren und Aktoren auf dem *IoT-Education-Board* getestet werden.

Dieses Dokument und andere Unterlagen befinden sich auf <https://gitlab.com/tjaeggi/iot-education-board>

2 Full-Featured IoT-Education-Board

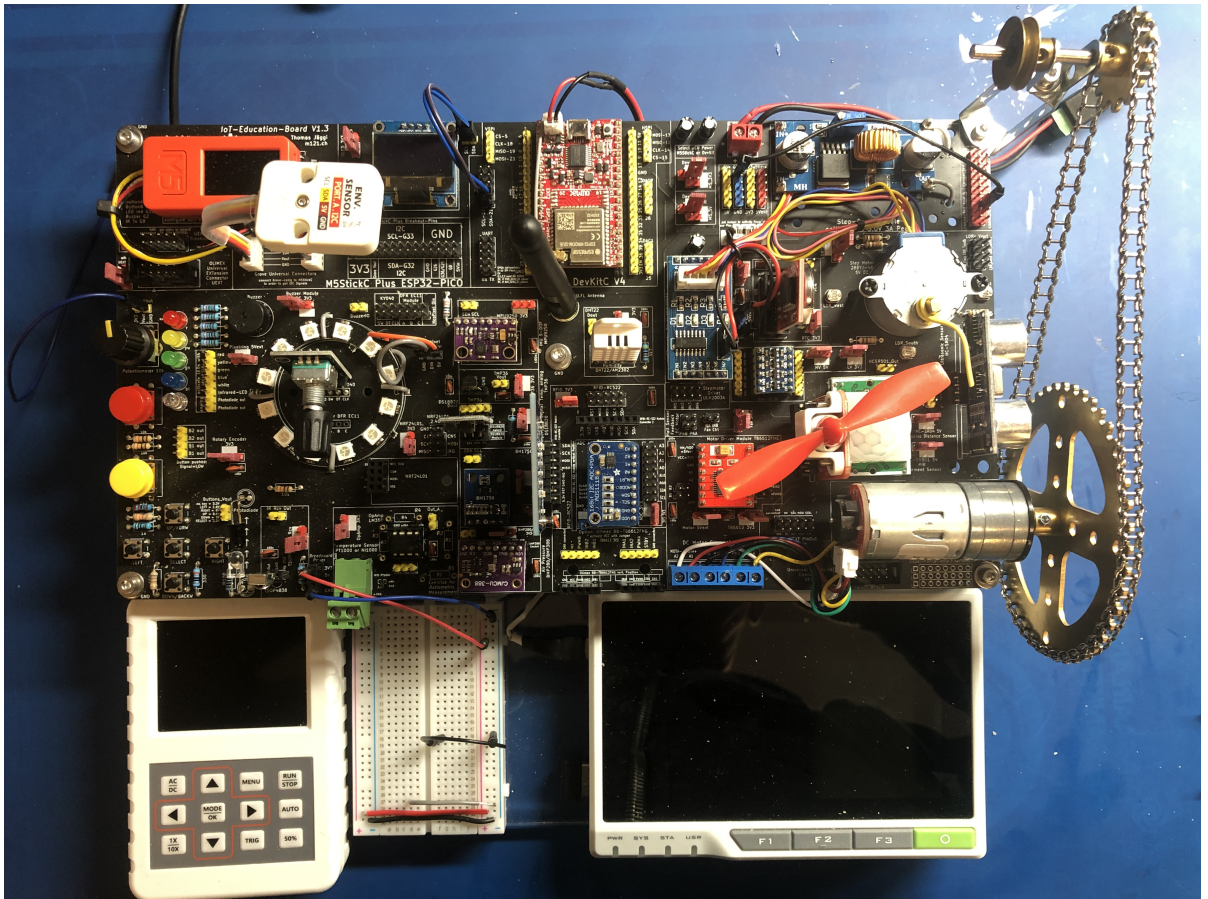


Abbildung 1: Full-Featured **IoT-Education-Board** mit zwei ESP32-Microcontroller, Oszilloskop (DSO FNIRISI PRO), Raspberry Pi CM4 (reTerminal von seed), M5Stack-Sensor an Grove-Connector und Antriebsrad für Spielereien... (<https://www.stokys.ch/>)



3 Einleitung

Das Board ist konzipiert für die Ausbildung im Bereich Regelung und Steuerung und als Lernplattform für Anwendungen im Bereich IoT. Neben den Sensoren und Aktoren befinden sich auf dem Board zwei Microcontroller:

- › M5StickC Plus. Siehe die [Detaildokumentation](#) des Herstellers.
- › Olimex ESP32-Dev-Kit-LiPo. Siehe auch [Detaildokumentation](#).

Beide Mikrocontroller basieren auf dem ESP32-Microprozessor.



Abbildung 2: M5StickC Plus

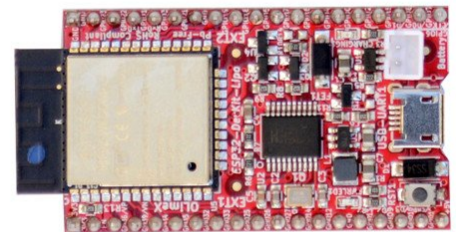


Abbildung 3: Olimex ESP32-Dev-Kit-LiPo

Die Sensoren und Aktoren werden direkt auf dem Board mit der notwendigen Spannung versorgt. Die Motoren erhalten die Spannung über das DC-DC Step-Down Modul. Das Modul kann mit bis zu 30V Eingangsspannung versorgt werden und liefert, je nach Modell, bis zu 3A Strom. Das Modul wird mit einer Ausgangsspannung V_{ext} von 5V betrieben. Die Spannung pro Sensor/Aktuator kann mit Jumper ein- und ausgeschaltet werden. Die Steuerpins der Komponenten sind auf männliche Pin-Header herausgeführt. Für Verbindungen zwischen Mikroprozessor und Sensor/Aktuator werden somit ausschliesslich weibliche Dupontkabel benötigt.



4 Board-Übersicht

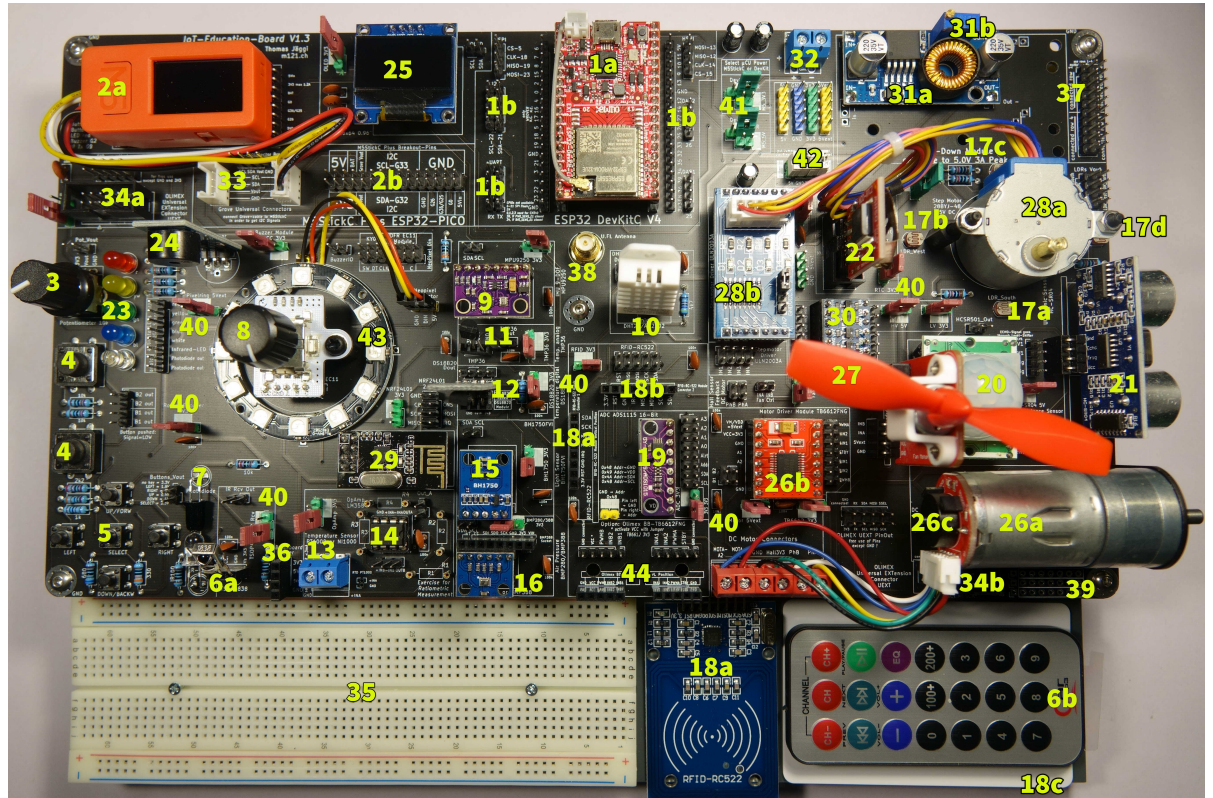


Abbildung 4: Komponenten IoT-Education-Board



Nr	Beschreibung
1a	Microcontroller Olimex ESP32-Dev-Kit-LiPo
1b	Anschlusspins des Microcontrollers Olimex ESP32-Dev-Kit-LiPo
2a	Microcontroller M5StickC Plus
2b	Anschlusspins des Microcontrollers M5StickC Plus
3	Drehpotentiometer 10k Ω
4	2 Buttons gross (12mm)
5	5 Buttons klein (6mm). Die Anordnung eignet sich z. Bsp. für eine Menusteuerung.
6a	Infrarot-Receiver TSOP4843 zum Verarbeiten der Signalen von handelsüblichen Fernbedienungen
6b	Fernbedienung
7	Infrarot Empfänger und Sender 950nm
8	Rotary Encoder EC11 oder KY-040
9	IMU MPU-9250
10	Feuchtigkeitssensor DHT22
11	analoger Temperatursensor TMP36
12	digitaler Temperatursensor DS18B20
13	Anschlussklemme für PT1000 oder NI1000 Temperatursensor
14	Operationsverstärker LM358P mit Pin-Sockets für die Widerstände R1 bis R4. Damit kann das Signal des PT1000 verstärkt werden, damit möglichst wenig Strom durch den Sensor fließen muss. Die Selbsterwärmung des Sensors wird so verhindert.
15	Lichtsensormodul BH1750. Der Sensor liefert die Helligkeit in Lux.
16	Luftdrucksensormodul BMP280
17a-d	4 Light-Dependant-Resistors (LDR). Der Widerstand eines LDR verändert sich abhängig vom Lichteinfall.
18a	RFID-RC522 Reader/Writer
18b	Steckplätze für RFID-RC522 Reader/Writer
18c	RFID-Karte MiFare
19	Analog Digital Converter ADC ADS1115
20	Bewegungssensormodul PIR HC-SR501
21	Ultraschall Distanzsensormodul HC-SR04
22	RTC Real Time Clock DS3231
23	5 LEDs in den Farben rot, gelb, grün, blau und weiss
24	passiver Buzzer
25	OLED Display mit SSD1306-Driver. 128x64 Pixel. I2C-Adresse: 0xC
26a	DC-Motor vom Typ N20 oder JGA25-370
26b	Motor Driver TB6612FNG
26c	Hall-Sensoren für MotorFeedback
27	Fan-Modul mit Motortreiber L9110H
28a	Schrittmotor 28BYJ-48
28b	Motortreiber ULN2003 für Schrittmotor
29	Funkmodul NRF24L01
30	Level-Shifter 4 Kanal
31a	Step-Down Modul mit 5V Ausgangsspannung
31b	Potentiometer, um die Ausgangsspannung des Step-Down-Moduls einzustellen. Diese soll auf 5V eingestellt werden. Höhere Spannungen können die Motoren zerstören.



weitere Anschlüsse:

32	Anschlussklemme für AC-DC Adapter (max 30V DC). Diese Spannung wird auf den Eingang des Step-Down-Moduls geführt.
33	Grove-Connector für Sensoren/Aktoren. Der Stecker (SDA, SCL, 5V, GND) ist mit dem M5StickC Plus verbunden.
34a	Universeller Stecker 1 (UEXT) für externe Sensoren/Aktoren
34b	Universeller Stecker 2 (UEXT) für externe Sensoren/Aktoren
35	Breadboard
36	Power-Anschlüsse für Breadboard
37	4x Pin-Reihen, um elektrische Punkte zu multiplizieren
38	Befestigung für optionale U.FL-Antenne. Diese kann verwendet werden, wenn der Microcontroller über einen entsprechenden Anschluss verfügt
39	Lötpunkte, um zusätzliche Komponenten zu fixieren. Die vertikalen Reihen sind miteinander verbunden.
40	Jumper mit Fahne: Damit wird die Spannung für die entsprechende Komponente ein- bzw. ausgeschaltet. Wenn der Jumper neben dem weissen Feld eingesteckt wird, wird das Bauteil mit Spannung versorgt.
41	Mit diesen Jumpers wird festgelegt, von welchem Microcontroller 3.3V bzw. 5V bezogen wird.
42	Mit diesen vier Jumpers erfolgt die Spannungsversorgung der Komponenten auf dem Board (3.3V, 5V vom Microcontroller, 5Vext vom Step-Down-Modul und GND). Diese sollten immer gesteckt sein, ausser wenn der Strom gemessen werden soll. Dabei kann anstelle des Jumpers ein Multimeter in Ampèrestellung an die Pins geklemmt werden.
43	Neopixel-Ring 12x oder 16x
44	Alternativer Footprint für Motortreiber TB6612FNG von Olimex. Artikel: BB-TB6612FNG

5 Arduino IDE

5.1 Vorbereitung

Damit mit dem *Olimex ESP32-DevKit* gearbeitet werden kann, muss der USB-Treiber CH341SER installiert werden. Dieser Treiber ist für Windows und Mac notwendig, damit die Verbindung PC - Microcontroller funktioniert. Der Treiber kann hier heruntergeladen werden: [CH341SER-Driver](https://github.com/tjaeggi/iot-education-board/-/tree/master/Software)

<https://github.com/tjaeggi/iot-education-board/-/tree/master/Software>

Verwenden Sie für Windows Betriebssysteme: CH341SER.zip

und für Mac Betriebssysteme: CH341SER_MAC.ZIP

Bei Linux ist keine Installation notwendig. Der Treiber ist bereits im Kernel vorhanden.

Nachdem der PC mit dem Olimex-Mikrocontroller mit einem MicroUSB-Kabel verbunden wurde, sieht die Verbindung im Gerätemanager unter Windows so aus:

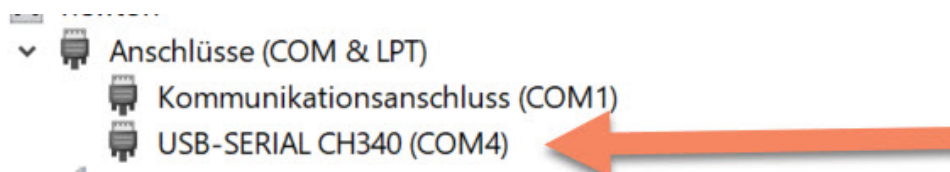


Abbildung 5: ESP32-Dev-Kit Verbindung im Gerätemanager



Um den *M5StickC Plus* in Betrieb zu nehmen, wird kein Treiber benötigt. Ein kurzes Einführungsvideo inkl. Arduino-Installation finden Sie unter:
<https://www.youtube.com/watch?v=ppXkl0046dc>

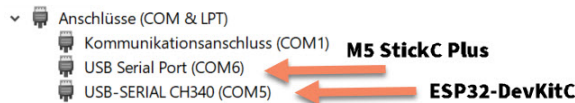


Abbildung 6: Verbindungen mit ESP32-Dev-Kit und M5StackC Plus sind im Gerätemanager sichtbar

5.2 Installation Arduino IDE

Die Arduino Entwicklungsumgebung ist für den Start sehr geeignet. Sie ist verfügbar für Windows, Mac und Linux und kann unter folgendem Link (Dezember 2021: Version 1.8.19) heruntergeladen werden:

<https://www.arduino.cc/en/software>

Zum Zeitpunkt der Erstellung dieses Dokumentes ist die Arduino IDE in der Version 2.0.0-rc3 als *Release Candidate* verfügbar. Diese Version bietet neue Funktionalitäten und entspricht einer modernen und einfach zu bedienenden IDE. Verwenden Sie diese, sobald der Release stabil ist.

5.3 Einstellungen in der Arduino IDE

Mit der Arduino IDE können viele verschiedene Typen von Microcontrollern programmiert werden. Damit bei der Kompilierung Binary-Code für den ESP32 generiert wird, müssen folgende Einstellungen gemacht werden:

Unter FILE/PREFERENCES muss unter ADDITIONAL BOARD MANAGER URLS folgender Eintrag gemacht werden:

Der String kann von [hier](#) kopiert werden.

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/Software/esp32link.txt>

https://dl.espressif.com/dl/package_esp32_index.json

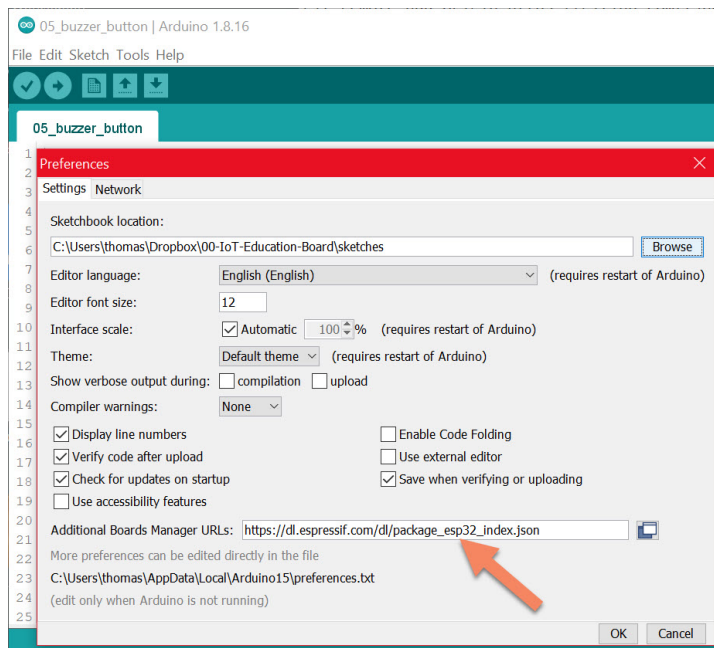


Abbildung 7: Board Manager Eintrag in der Arduino IDE

Anschließend suchen Sie unter TOOLS/BOARD/BOARDS MANAGER die ESP32-Packages und klicken auf INSTALL.

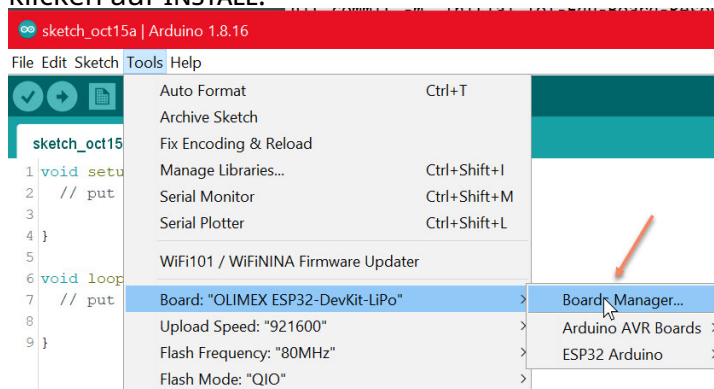


Abbildung 8: ESP32 Boardmanager starten

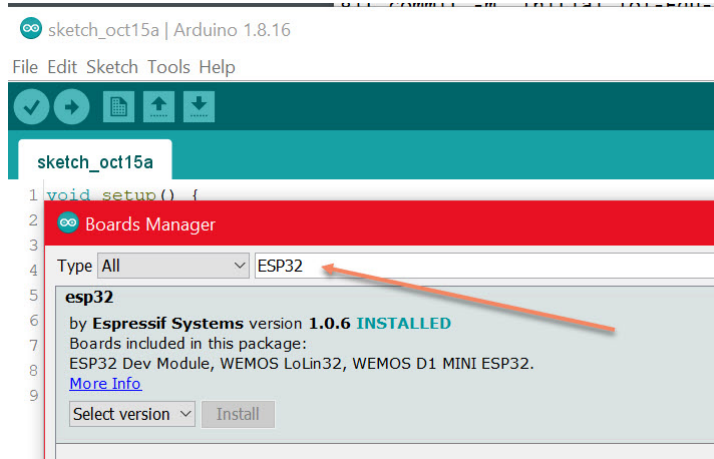


Abbildung 9: ESP32-Package suchen und installieren

Danach kann das Board, mit welchem gearbeitet werden soll, ausgewählt werden. Wichtig ist auch, dass der entsprechende Port, mit welchem der Microcontroller verbunden ist, gesetzt wird. Sie sehen, dass beide Controller verfügbar werden:

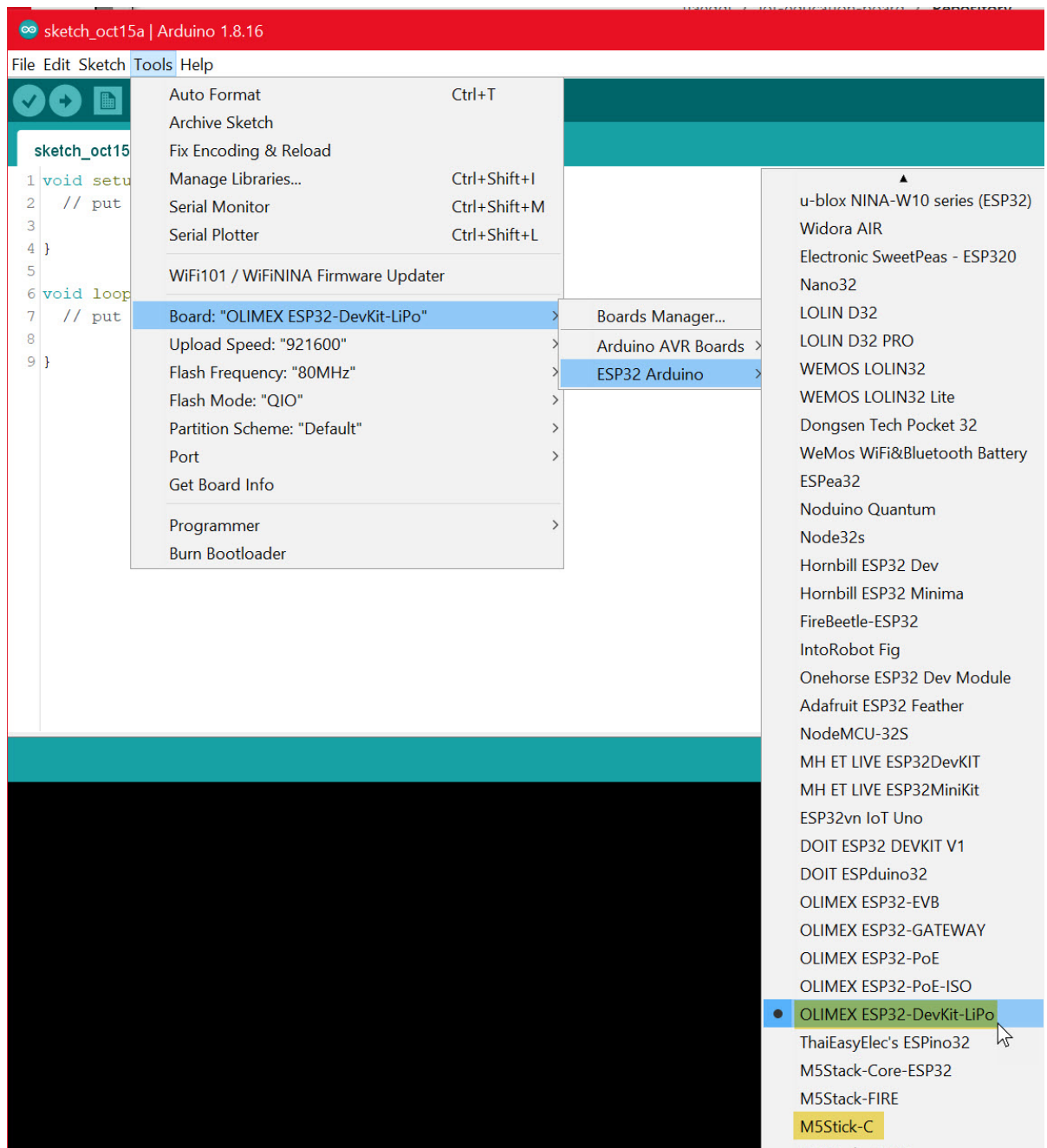


Abbildung 10: Auswahl des Microcontrollers

In der Folge werden wir mit dem Olimex ESP32-DevKit-Lipo arbeiten. Stellen Sie sicher, dass die Einstellungen so aussehen:

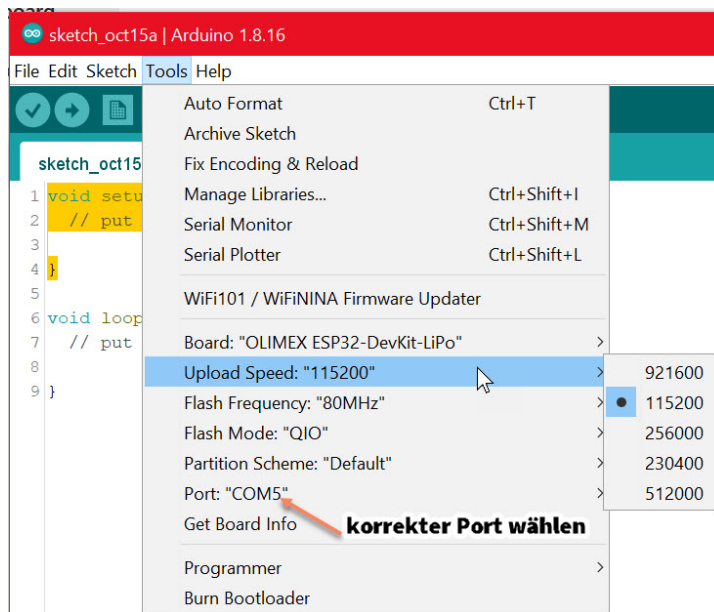


Abbildung 11: Board-Einstellungen für OLIMEX ESP32-DevKit-LiPo

6 Inbetriebnahme des Boards

Das Board wird in Betrieb genommen, indem der Microcontroller mit dem PC mit einem USB-Kabel verbunden wird. Der USB-Port des PCs versorgt die Komponenten via dem Microcontroller mit der notwendigen Spannung. Die allermeisten Komponenten benötigen 3.3V. Alle Komponenten, welche 5V benötigen, erhalten diese ebenfalls vom Microcontroller.

Ausnahme: Neopixel-Ring und Motoren. Diese erhalten die 5V-Spannung vom DC-DC-Stepdown-Modul, weil der Microcontroller nicht genügend Strom (max. 40mA) liefern kann.

Das Stepdown-Modul wird über einen externen AC/DC-Adapter mit Spannung versorgt (max 9-20VDC/2-3A). Siehe Nr.32, in Abbildung 4).

6.1 Komponenten ein- und ausschalten

Jede Komponente kann ein- und ausgeschaltet werden. Dies ist mit einem roten Jumper (mit Griff) realisiert. Wenn der Jumper **über** dem weißen Feld platziert wird, ist die Komponente **eingeschaltet**. Wenn der Jumper nicht direkt über dem weißen Feld eingesteckt wird, ist die Komponente spannungslos. Abbildung 12 zeigt die beiden Jumperpositionen exemplarisch bei drei Sensoren:

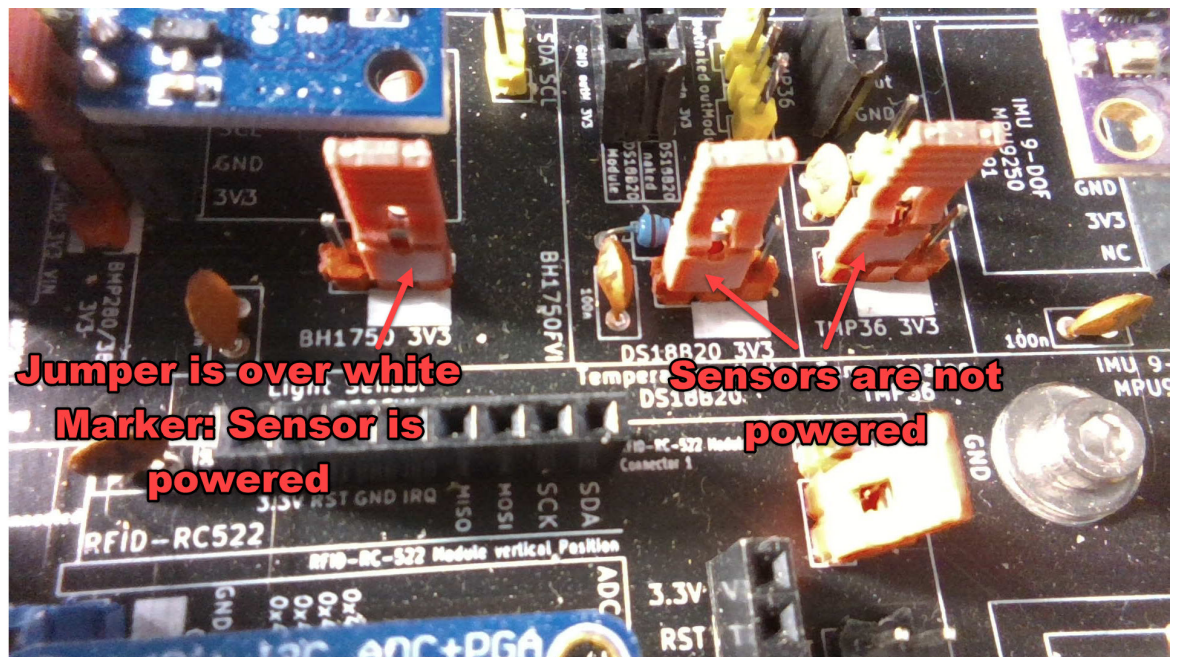


Abbildung 12: Jumperpositionen für Spannung on/off



Jumperstellung

Es ist zu empfehlen, dass sich die Jumperstellung **neben** der weissen Fläche befindet, wenn die Komponente nicht verwendet werden soll. So wird vermieden, dass der Microcontroller bezüglich Strombedarf weniger belastet wird. Grundsatz: nur diejenigen Komponenten einschalten, welche auch verwendet werden. Alles andere ist ausgeschaltet: Jumper neben und nicht über dem weissen Feld.



Wahl des Microcontrollers

Siehe Nr. 41 der Abbildung 4. Mit diesen beiden Jumpfern wird festgelegt, mit welchem Microprozessor gearbeitet werden soll, bzw. von welchem Microcontroller die Komponenten die Spannung beziehen. Setzen Sie beide Jumper entweder auf DevK_3V3 und DevK_5V (Olimex DevKit-Lipo, roter Microcontroller) **oder** M5_3V3 und M5_5V (M5StickC Plus, oranger Microcontroller). Mischen Sie nicht! Trotzdem kann selbstverständlich mit beiden Microcontrollern gleichzeitig gearbeitet werden. Mit den Jumperpositionen entscheiden Sie, von welchem Controller die Sensoren ihre Spannung beziehen sollen. **Beide Controller besitzen denselben GND.** Das vorliegende Manual behandelt mit den Demo-Sketches den roten Olimex DevKitC-LiPo-Microcontroller.

Abbildungen 13, 14 zeigen die zulässigen Jumperstellungen. Eine Jumperstellung wie in Abbildung 15 sollte vermieden werden.

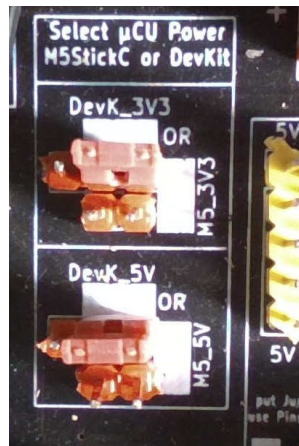


Abbildung 13: Spannungsversorgung durch DevKitC V4 (Olimex) Microcontroller

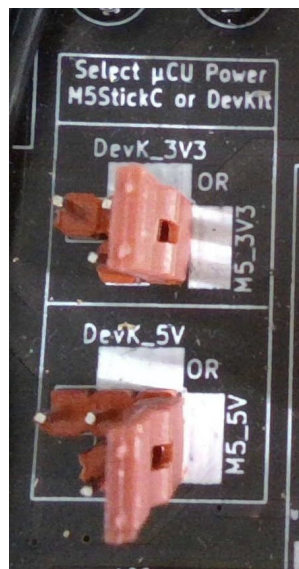


Abbildung 14: Spannungsversorgung durch M5Stick



Abbildung 15: gemischte Spannungsversorgung vermeiden



7 Vorbemerkungen zu den Demo-Sketches

7.1 Allgemeines

Bei jedem Beispiel müssen eine oder mehrere Verbindungen zwischen Sensor/Aktor und Microcontroller hergestellt werden. Verwenden Sie dazu weibliche Dupont-Kabel. Die Demosketches befinden sich auf <https://gitlab.com/tjaeggi/iot-education-board>. Ein direkter Link zum Source-Code auf *gitlab* ist bei jedem Sketch vorhanden und kann nach dem Klick auf den Link direkt aus dem Browser-Fenster kopiert werden. **Source-Code nicht aus diesem pdf-Dokument kopieren!**



Abbildung 16: Dupontkabel female-female

Bitte beachten:

Ein Arduino-Programm wird **SKETCH** genannt!

Ein Pin eines Microcontrollers wird **GPIO** genannt. GPIO = General Purpose I/O. Jeder GPIO hat eine Nummer.

Fast alle Demo-Sketches verwenden **Libraries**. Hinweise dazu sind im Text oder im Source-Code vorhanden. Libraries werden im Source-Code mit **#include** eingebunden. Installieren Sie diese in der Arduino-IDE unter SKETCH/INCLUDE LIBRARY/MANAGE LIBRARIES, bevor Sie den Code kompilieren. Wenn die Library nicht installiert ist, erhalten Sie Fehlermeldungen. Im Abschnitt **7.2** wird erklärt, wie Libraries in der Arduino-IDE installiert werden.

Kopieren Sie den Source-Code stets vom gitlab und **nicht** aus dem pdf-Dokument. Damit vermeiden Sie, dass ungewollte Character in den Source-Code gelangen und Probleme verursachen. Zu jedem Demo-Sketch ist der Link zum Source-Code auf gitlab vorhanden. Kopieren Sie diesen in den Arduino-Editor mit ctrl-a, ctrl-c und ctrl-v.



Machen Sie sich mit den einzelnen Sensoren und Aktoren vertraut. Nehmen Sie diese einzeln in Betrieb, indem Sie die hier vorgestellten Sketches verwenden. Wenn Sie die einzelnen Komponenten softwaremässig kontrollieren können, ist es relativ einfach, komplexe Steuerungen und Regelungen zu realisieren, indem Sie die Softwareteile miteinander kombinieren.

Technische **Datasheets** zu einzelnen Komponenten, welche auf dem Board verbaut sind, finden Sie hier:

<https://gitlab.com/tjaeggi/iot-education-board/-/tree/master/DatasheetsComponents>

Wenn Sie Komponenten auf das Board stecken, **IMMER** die Signalbezeichnungen des Bauteils beachten! Die Bezeichnung auf dem Board muss mit der Bezeichnung des Bauteil-Pins übereinstimmen.

7.2 Arduino Libraries

Libraries, in Deutsch "*Bibliotheken*", sind Programmteile eines Arduino-Sketches. Mit dem Befehl `#include` können wir *libraries* nutzen. Die *libraries* ergänzen unseren Programmcode.

Warum werden Libraries verwendet?

Libraries werden oft für Komponenten angeboten, um komplexe Programmteile universell verfügbar zu machen. Beispiel: Die Abfrage von Sensorwerten eines bestimmten Sensortyps ist komplex und funktioniert nur richtig, wenn die physikalischen Eigenschaften des Sensors im Detail berücksichtigt werden. Diese Details sind meist in den Datasheets der Komponenten beschrieben. Jeder Sensor hat seine ganz spezifischen Eigenschaften und sind auch je nach Hersteller unterschiedlich. Wenn es nun eine Library gibt für einen spezifischen Sensor, kann das Ansprechen und die Abfrage der Sensorwerte massiv vereinfacht werden, indem im eigenen Programmcode mit einem `#include` die Programmteile der Library integriert werden. Manchmal stellt der Hersteller eines Sensors eine Arduino-Library zur Verfügung. Oft ist es aber so, dass jemand aus der OpenSource-Community eine Library schreibt und diese öffentlich zur Verfügung stellt.

Wenn es eine Library gibt, muss man als Programmierer nur noch wissen, mit welchen Parametern diese angesprochen werden muss.

Viele Libraries sind in der Arduino-IDE bereits integriert, bzw. werden heruntergeladen, wenn diese ausgewählt werden. Libraries findet man in der IDE unter:

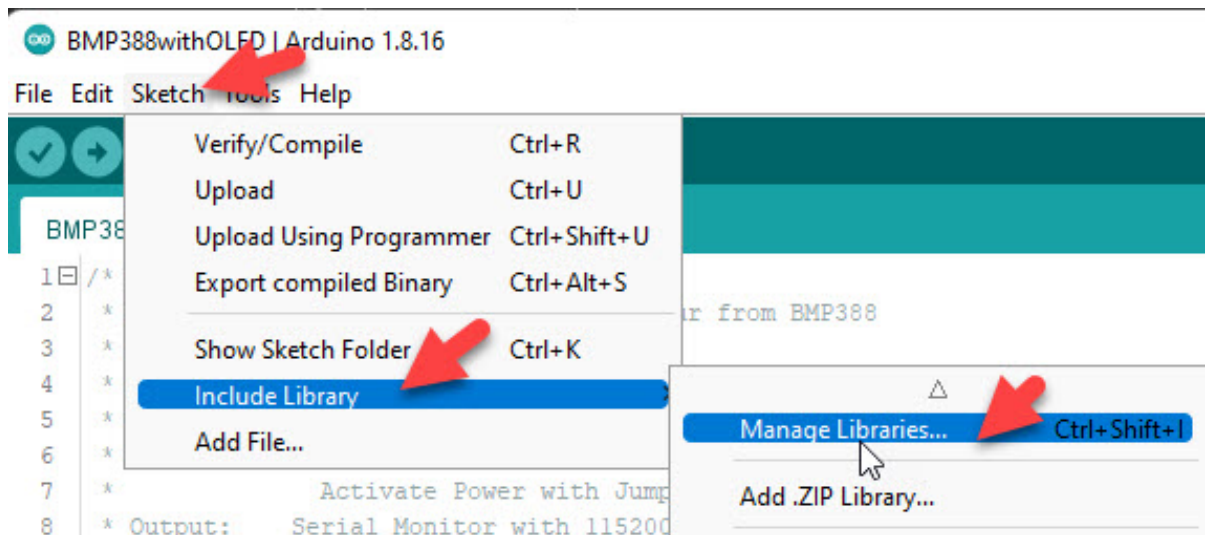


Abbildung 17: Arduino-Library Menu

Die Suche, Auswahl und das Installieren der Library erfolgt unter MANAGE LIBRARIES:

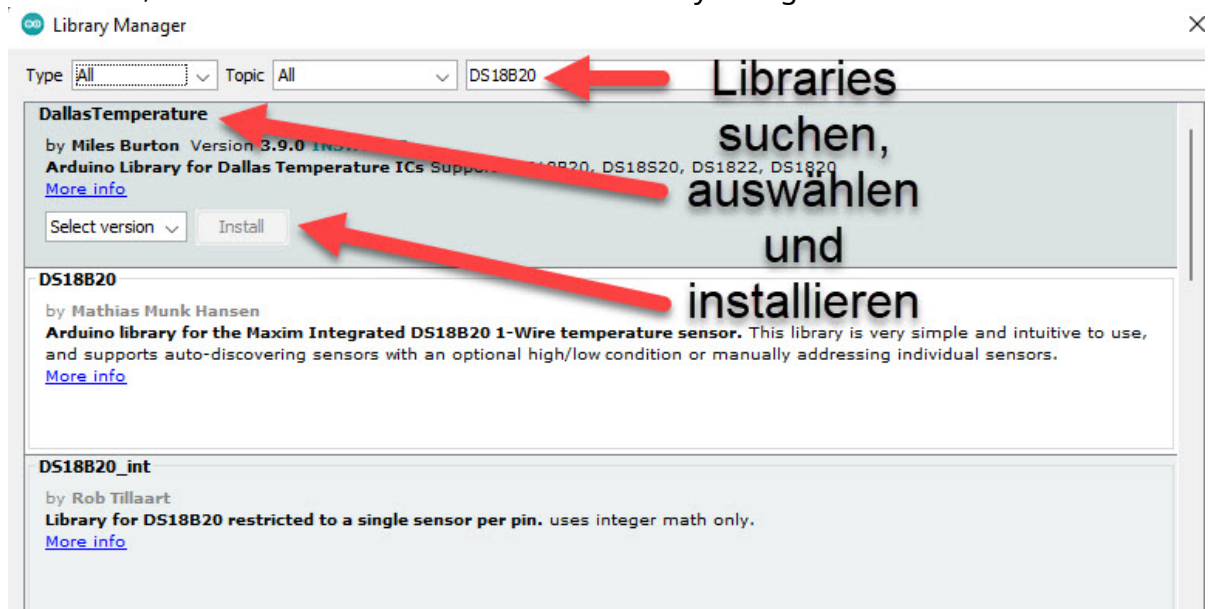


Abbildung 18: Hier wird die Library des Temperatursensors DS18B20 ausgewählt und installiert. Auch updates werden von hier aus gemacht.

Der Temperatursensor DS18B20 wird von der Firma *Dallas Semiconductor* hergestellt und hat ganz spezifische Eigenschaften, welche von der Library berücksichtigt werden. Oft ist es auch so, dass eine bestimmte Library auf anderen Libraries basiert. Dies ist auch hier der Fall. Die DS18B20-Library basiert auf der 1wire-Library, welche für das Kommunikationsprotokoll zwischen Microcontroller und Sensor verantwortlich ist.

Man kann auch sagen, dass eine Library das physische Bauteil so abstrahiert, dass der Programmierer nur noch wissen muss, mit welchen Programmbefehlen er welche Funktionen des Bauteils aktivieren muss. Beim Durchspielen der Demo-Sketches werden Sie verstehen, was damit gemeint ist.

Mehr Details zu Arduino-Libraries finden Sie unter:

<https://www.arduino.cc/en/Guide/Libraries>

<https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use>



8 Demo-Sketches

8.1 LED blinken

Stellen Sie zuerst eine Verbindung zwischen dem roten LED und dem Microcontroller her: Verbinden Sie dazu Pin red mit GPIO23 des Microcontrollers, wie auf dem Bild dargestellt.

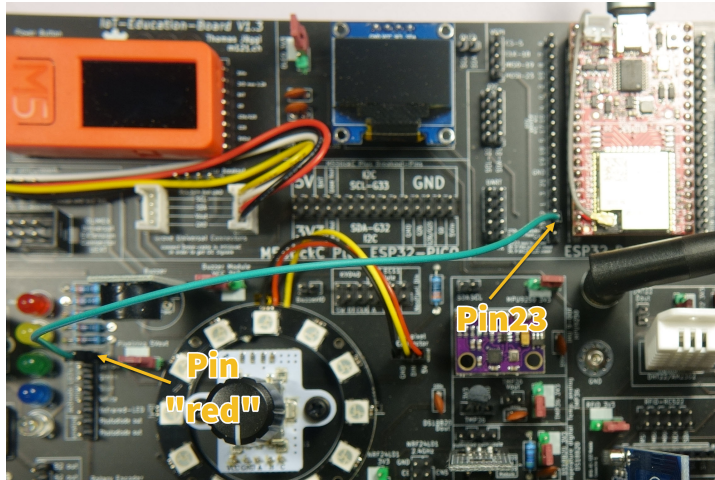


Abbildung 19: Verbindung Pin "red" mit Pin23

Öffnen Sie nun das Blink-Skript mit der Arduino IDE. Kopieren Sie dazu den Inhalt unter dem [Link](#) in ein leeres Sketch-Fenster und speichern Sie es ab. Anschliessend laden Sie das Programm auf den Mikroprozessor mit UPLOAD.

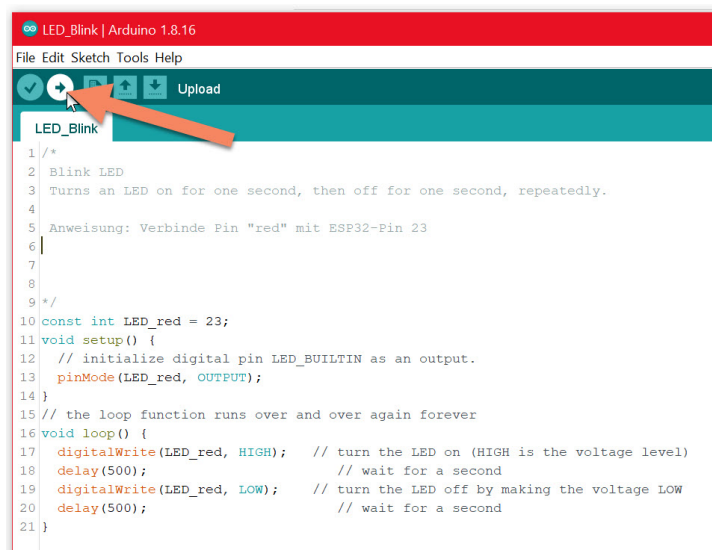


Abbildung 20: Kompiliert den Blink-Sketch gefolgt von einem Upload

Sketch: [LEDBlink.ino](#)

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/LEDBlink/LEDBlink.ino>

Wenn alles geklappt hat, wird jetzt das rote LED blinken. Verbinden Sie Pin23 auch mit dem gelben, grünen, blauen und weissen LED. Alle LEDs sollten blinken.

Source-Code:

```
1 /*
   Title:      LED blink
```



```
3 Sketchfile: LEDBlink.ino
5 Function: Turns an LED on for one second, then off for one second, repeatedly.
   Connections: Connect Pin "red" with ESP32-Pin 23
7
9 Date: tja/15.10.2021
11 */
13 const int LED_red = 23;
14 void setup() {
15     // initialize digital pin LED_BUILTIN as an output.
16     pinMode(LED_red, OUTPUT);
17 }
18 // the loop function runs over and over again forever
19 void loop() {
20     digitalWrite(LED_red, HIGH); // turn the LED on (HIGH is the voltage level)
21     delay(500); // wait for a second
22     digitalWrite(LED_red, LOW); // turn the LED off by making the voltage LOW
23     delay(500); // wait for a second
24 }
```

Listing 1: LEDBlink.ino

8.2 LED dimmen

Verbinde Pin red mit GPIO23 auf dem Mikrocontroller. Sketch: [LEDdimmen.ino](#)

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/LEDdimmen/LEDdimmen.ino>

Source-Code:

```
2 /*
3 Title: LED fading
4 Sketchfile: LEDdimmen.ino
5
6 Function: Fading a LED using PWM
7 Connections: Connect Pin "red" with ESP32-Pin 23
8
9 Date: tja/15.10.2021
10 */
11
12 const byte led_gpio = 23; // the PWM pin the LED is attached to
13 int brightness = 0; // how bright the LED is
14 int fadeAmount = 5; // how many points to fade the LED by
15
16 // the setup routine runs once when you press reset:
17 void setup() {
18     ledcAttachPin(led_gpio, 0); // assign a led pins to a channel
19
20     // Initialize channels
21     // channels 0-15, resolution 1-16 bits, freq limits depend on resolution
22     // ledcSetup(uint8_t channel, uint32_t freq, uint8_t resolution_bits);
23     ledcSetup(0, 5000, 8); // 12 kHz PWM, 8-bit resolution
24 }
25
26 // the loop routine runs over and over again forever:
27 void loop() {
28     ledcWrite(0, brightness); // set the brightness of the LED
29
30     // change the brightness for next time through the loop:
31     brightness = brightness + fadeAmount;
32 }
```



```
34 // reverse the direction of the fading at the ends of the fade:
35 if (brightness <= 0 || brightness >= 255) {
36     fadeAmount = -fadeAmount;
37 }
38 // wait for 30 milliseconds to see the dimming effect
39 delay(30);
40 }
```

Listing 2: LEDdimmen.ino

8.3 Analoges Signal von Potentiometer lesen

8.3.1 Ausgabe im Serial Monitor

Verbinde PotVout mit GPIO34 auf dem Mikrocontroller. Sketch: [ReadPotentiometer.ino](#).

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/ReadPotentiometer/ReadPotentiometer.ino>

Durch Drehen am Potentiometer kann im Serial Monitor der Arduino IDE die Signalveränderung beobachtet werden.

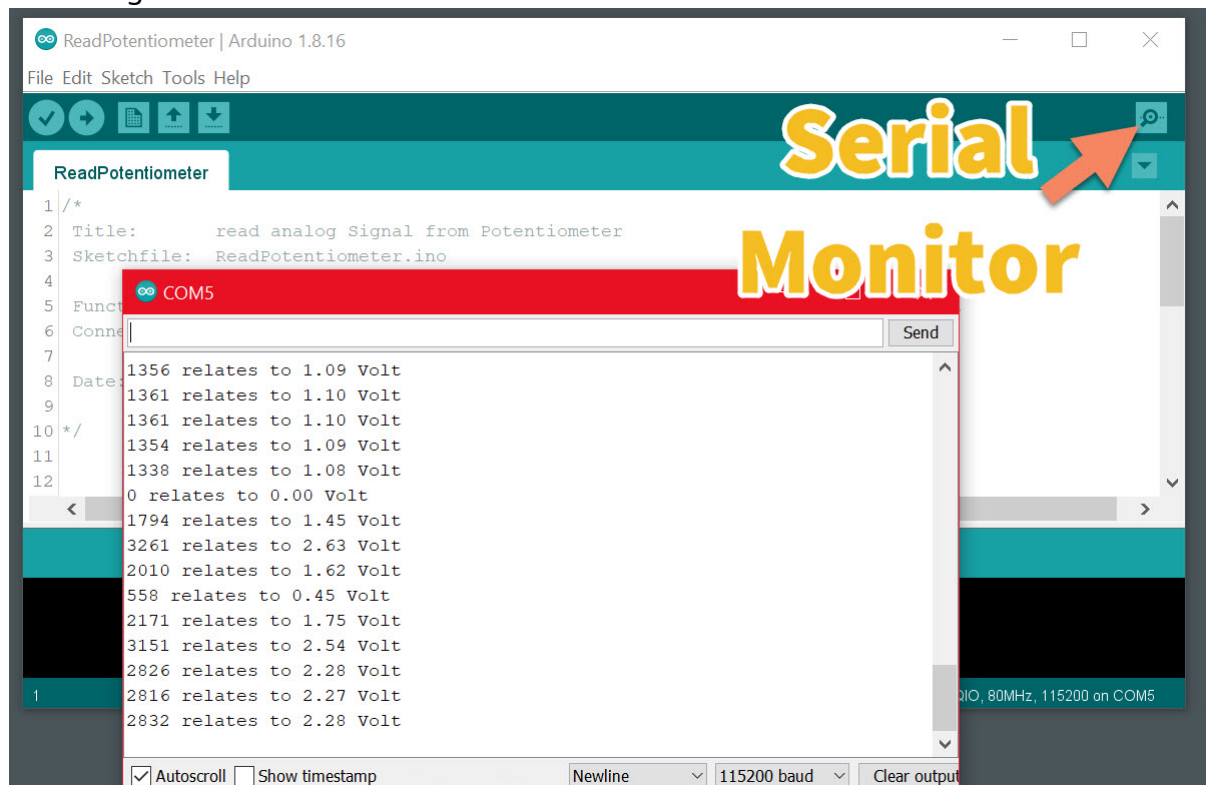


Abbildung 21: Ausgabe der analogen Signalwerte im Serial Monitor

Source-Code:

```
1 /*
2  Title:      read analog Signal from Potentiometer
3  Sketchfile: ReadPotentiometer.ino
4
5  Function:   Read analog Signal from Poti and convert it into Voltage, using ADC.
6  Connections: Potentiometer is connected to GPIO 34 (Analog ADC1_CH6)
7
8  Date:      tja/15.10.2021
9
10 */
11
12
```




```
14 const int potPin = 34;
15 // variable for storing the potentiometer value
16 int potValue = 0;
17
18 void setup() {
19     Serial.begin(115200);
20     delay(1000);
21     analogSetPinAttenuation(potPin, ADC_11db);
22     analogReadResolution(12);
23     analogSetWidth(12);
24 }
25
26 void loop() {
27     // Reading potentiometer value
28     potValue = analogRead(potPin);
29     float voltage = potValue * (3.3 / 4095.0);
30     Serial.print(potValue); Serial.print(" relates to "); Serial.print(voltage); Serial
31     .println(" Volt");
32
33     delay(500);
34 }
35
36 \end{flushleft}}}
```

Listing 3: ReadPotentiometer.ino

8.3.2 Ausgabe im OLED Display

Verbinde PotVout mit GPIO34 und SCL- und SDA-Pins des OLED-Displays mit SCL- und SDA-Pins GPIO22 und GPIO21 des Microcontrollers. Sketch:

[ReadPotiOLEDout.ino](#).

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/ReadPotiOLEDout/ReadPotiOLEDout.ino>

Beachten Sie die verwendeten Libraries. Die Adafruit-Libraries Adafruit_GFX und Adafruit_S müssen installiert sein. Die Installation erfolgt über die Menüpunkte SKETCH/INCLUDE LIBRARY/MANAGE LIBRARIES.

Durch Drehen am Potentiometer kann auf dem OLED-Display die Signalveränderung beobachtet werden.

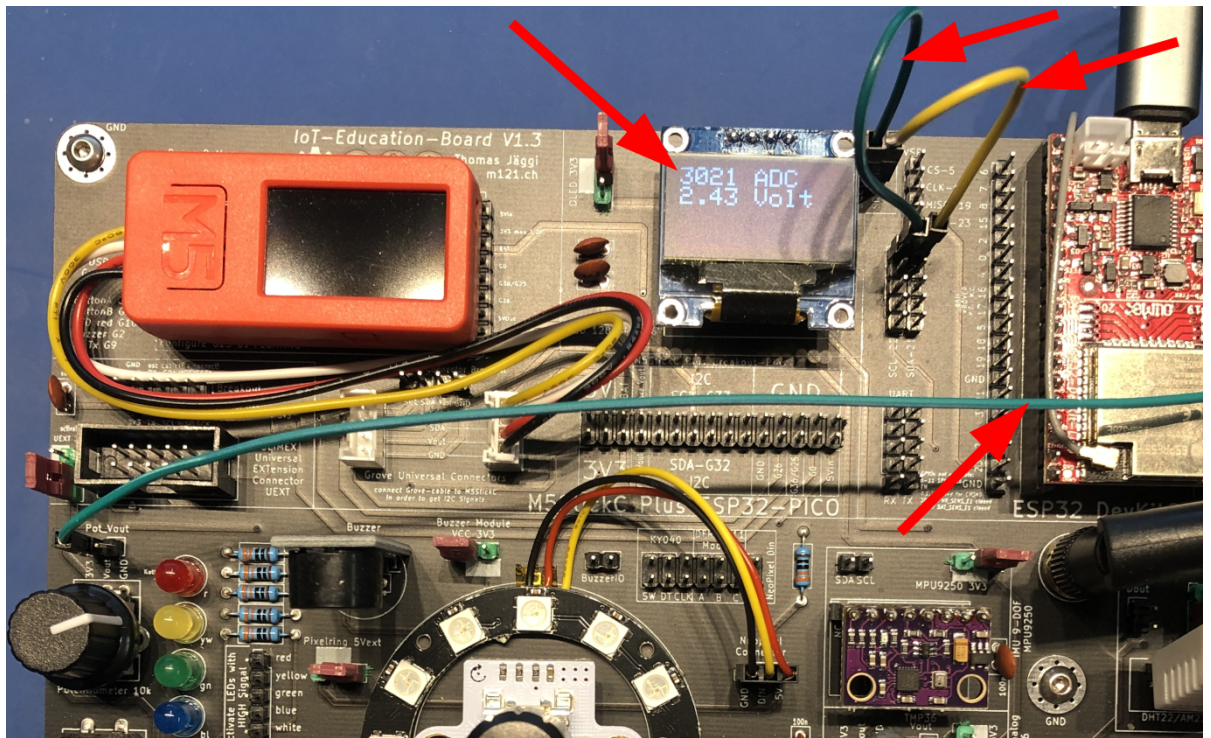


Abbildung 22: Ausgabe der analogen Signalwerte auf dem OLED-Display
Source-Code:

```
1 /*  
2 Title:      read analog Signal from Potentiometer and display ADC-Value and  
              Voltage onto OLED Display  
              Sketchfile: ReadPotiOLEDout.ino  
4  
5 Function:   Read analog Signal from Poti and convert it into Voltage, using ADC.  
6 Connections: Potentiometer is connected to GPIO 34 (Analog ADC1_CH6)  
              OLED SDA-Pin to SDA Pin Microcontroller GPIO21  
              OLED SCL-Pin to SCL Pin Microcontroller GPIO22  
              Jumper "OLED 3V3" is active: Place Jumper over white field.  
8  
9 Date:      tja/15.10.2021  
12 */  
14  
15 /* OLED Settings  
16 *  
17 */  
18 #include <Wire.h>  
20 #include <Adafruit_GFX.h>  
21 #include <Adafruit_SSD1306.h>  
22  
23 #define SCREEN_WIDTH 128 // OLED display width, in pixels  
24 #define SCREEN_HEIGHT 64 // OLED display height, in pixels  
26 #define OLED_RESET    4 // Reset pin # (or -1 if sharing Arduino reset pin)  
27 #define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64, 0x3C  
28                               for 128x32  
29 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);  
30  
31  
32 const int potPin = 34;  
34
```



```
// variable for storing the potentiometer value
36 int potValue = 0;

38 void setup() {
    Serial.begin(115200);
    // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
40         Serial.println(F("SSD1306 allocation failed"));
42         for(;;); // Don't proceed, loop forever
44     }
    // Clear the buffer
46     display.clearDisplay();
    delay(1000);
48     analogSetPinAttenuation(potPin, ADC_11db);
    analogReadResolution(12);
50     analogSetWidth(12);

52 }

54 void loop() {
    // Reading potentiometer value
56     potValue = analogRead(potPin);
    float voltage = potValue * (3.3 / 4095.0);
58     display.clearDisplay();
    display.setTextSize(2); // Normal 1:1 pixel scale
60     display.setTextColor(SSD1306_WHITE); // Draw white text
    display.setCursor(0, 0); // Start at top-left corner
62     display.cp437(true); // Use full 256 char 'Code Page 437' font
    display.print(potValue); display.println(" ADC");display.print(voltage);display.
        println(" Volt");
64     display.display();
    delay(500);
66 }
```

Listing 4: ReadPotiOLEDout.ino

8.4 Buzzer mit Button aktivieren

Auf dem Board können verschiedene Modelle von Buzzern verwendet werden. Hier zwei Beispiele für das Modul MH-FMD mit Low Level Trigger und einem passiven Buzzer. Vergleichen Sie die Bilder, um festzustellen, welcher Buzzer bei Ihnen verwendet wird. Für beide Typen gilt:

Verbinde BuzzerIO mit GPIO21 und den Button Pin B1 out mit GPIO16 des Microcontrollers. Sketch:

[BuzzerLowLevelTriggerButton.ino](#). Mit Button B1 gedrückt, ertönt der Buzzer.

8.4.1 Buzzertyp MF-FMD Low Level Trigger

Sketch: [BuzzerLowLevelTriggerButton.ino](#). Mit Button B1 gedrückt, ertönt der Buzzer.

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/BuzzerLowLevelTriggerButton/BuzzerLowLevelTriggerButton.ino>

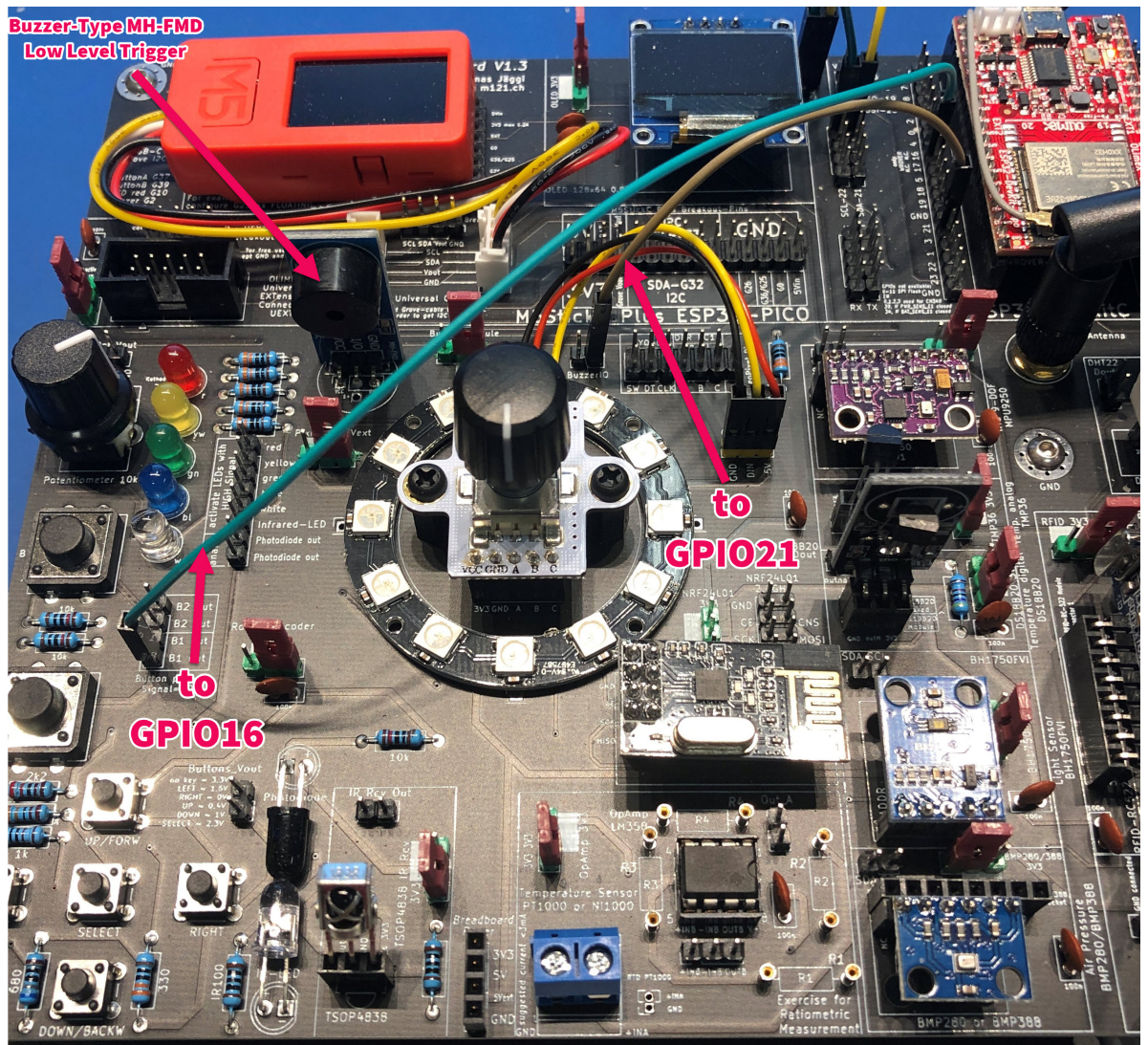


Abbildung 23: active Buzzer MH-FMD Low Level Trigger

Source-Code:

```
1  /*
3  * Sketch: BuzzerLowLevelTrigger_mit_Button.ino
4  * This Sketch works with the Buzzer-Module MH-FMD Low Level Trigger
5  * Signal = LOW Buzzer is activated
6  *
7  */
9
11 const int BUTTON_PIN = 16; // GPIO16 pin connected to button's pin
12 const int BUZZER_PIN = 21; // GPIO21 pin connected to Buzzer's pin "BuzzerIO"
13
14 void setup() {
15     Serial.begin(115200); // initialize serial
16     pinMode(BUTTON_PIN, INPUT); // set ESP32 pin to input
17     pinMode(BUZZER_PIN, OUTPUT); // set ESP32 pin to output mode
18 }
19
20 void loop() {
21     int buttonState = digitalRead(BUTTON_PIN); // read new state
22
23     if (buttonState == LOW) {
24         Serial.println("The button is being pressed");
25     }
26 }
```




```
25     digitalWrite(BUZZER_PIN, LOW); //Low-Level-Trigger: Buzzer on
    }
    else
27     if (buttonState == HIGH) {
        Serial.println("The button is unpressed");
29     digitalWrite(BUZZER_PIN, HIGH); //Low-Level-Trigger: Buzzer off
    }
31 }
```

Listing 5: BuzzerLowLevelTriggerButton.ino

8.4.2 Buzzertyp passiv

Der Sketch spielt eine Melodie, wenn Button B1 gedrückt ist.

Sketch: [PlaymelodyButtonPressed.ino](#).

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/PlaymelodyButtonPressed/PlaymelodyButtonPressed.ino>

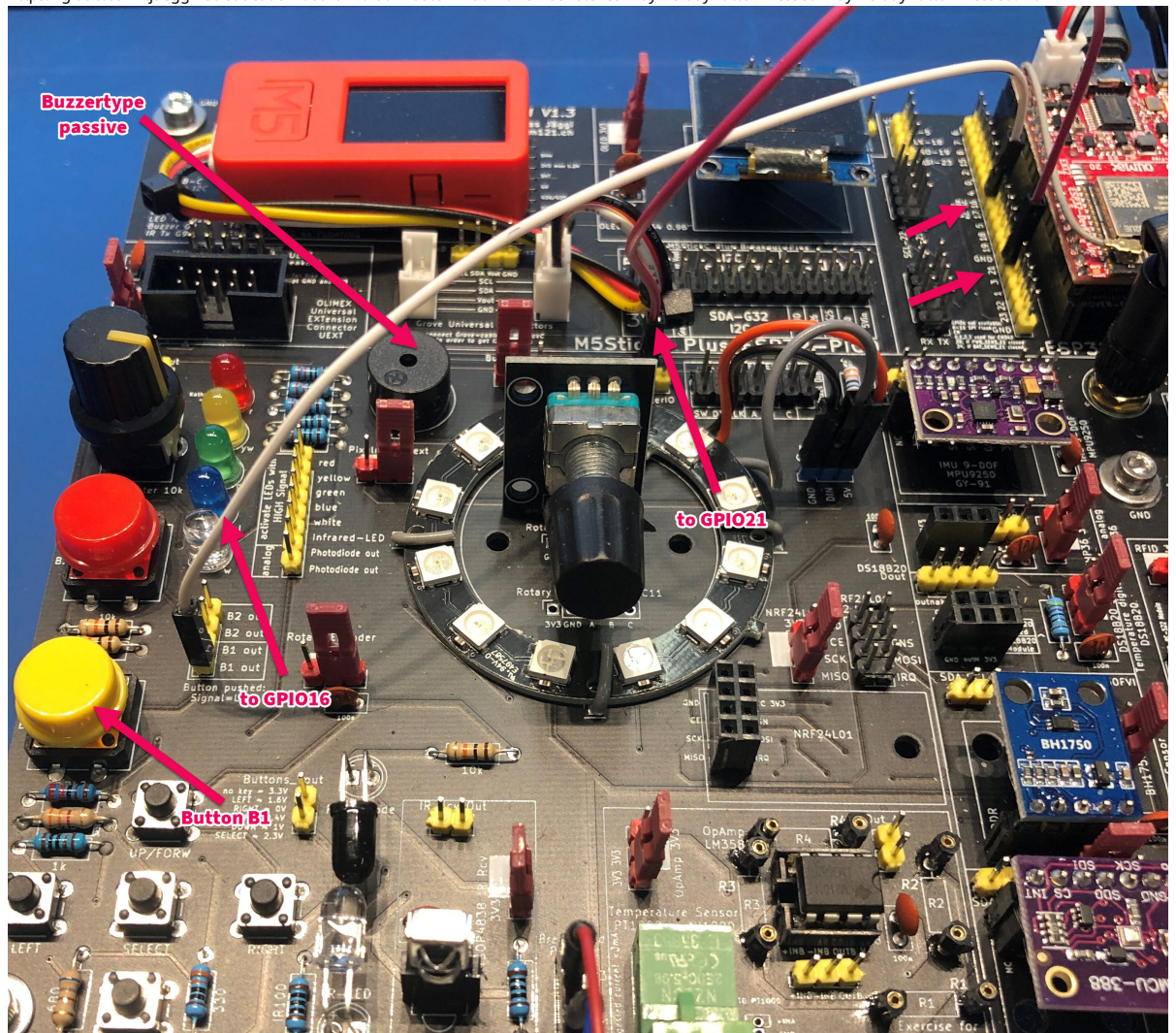


Abbildung 24: passive Buzzer control with Function ledcWriteNote()

Source-Code:

```
1 // Spielt Melodie, wenn B1 gedrückt wird.
3 //
4 // Verbinde Button-Pin B2 mit GPIO23
5 // Verbinde Buzzer-Pin BuzzerIO mit GPIO32
6 //
```




```
7 // Teilweise aus ESP32-Handbuch, Seite 296
// Siehe auch Kapitel zu PWM, Seite 291
9 // Siehe auch: https://www.thomascourtz.com/2021/02/21/arduino-tone-for-esp32

11 #define LEDC_KANAL_0      0          // Kanal 0; einer der 16-LEDC-Kanäle
#define LEDC_AUFLOESUNG    8          // 8-Bit-Tastverhältnisauflösung (0-255)
13 #define LEDC_FREQ        5000       // 5.000 Hz Frequenz
#define BUZZER_PIN        32          // GPIO32 mit Buzzer
15 #define BUTTON_PIN       23          // GPIO23 pin connected to button's pin

17 // 13 Noten bestehend aus Ton, Oktave, Dauer
typedef struct {note_t ton; uint8_t oktave; int laenge;} note;
19 note melodie [13] = {
    {NOTE_F, 3, 500}, {NOTE_D, 3, 500}, {NOTE_D, 3, 1000},
21    {NOTE_E, 3, 500}, {NOTE_C, 3, 500}, {NOTE_C, 3, 1000},
    {NOTE_B, 2, 500}, {NOTE_C, 3, 500}, {NOTE_D, 3, 500}, {NOTE_E, 3, 500},
23    {NOTE_F, 3, 500}, {NOTE_F, 3, 500}, {NOTE_F, 3, 1000} };

25 void buzzer() {
    for (int i = 0; i < 13; i++) {
27        ledcWriteNote(LEDC_KANAL_0, melodie[i].ton, melodie[i].oktave);
        delay (melodie[i].laenge);
29        ledcWrite(LEDC_KANAL_0, 0);
        delay (30);
31    }
    ledcWrite(LEDC_KANAL_0, 0);          // Tastverhältnis 0 %
33    delay (1000);                      // Pause zwischen zwei Noten
}

35 void setup() {
37    Serial.begin(115200);
    ledcSetup(LEDC_KANAL_0, LEDC_FREQ, LEDC_AUFLOESUNG);
39    ledcAttachPin(BUZZER_PIN, LEDC_KANAL_0);
}

41 void loop() {
43    int buttonState = digitalRead(BUTTON_PIN); // read new state
45
    if (buttonState == LOW) {
47        Serial.println("The button is being pressed");
        buzzer(); // turn on
49    }
    else
51    if (buttonState == HIGH) {
        Serial.println("The button is unpressed");
53    }
55 }
```

Listing 6: PlaymelodyButtonPressed.ino

8.5 Rotary Encoder

Hier wird der Rotary Encoder angesteuert und ausgelesen. Im Serial Monitor wird je nach Drehrichtung die Zahl hoch- oder runtergezählt. Beim Drücken des Knopfes wird der Zähler auf Null gesetzt. Physikalisch wird dabei der Pin SW von 3.3V auf GND gezogen. Darauf achten, dass der Jumper ROTARY ENCODER 3V3 über dem weissen Feld eingesteckt ist.

Source-Code [RotEncoder.ino](#):



```
1  /*
2  File: RotEncoder.ino
3  Siehe: https://garrysblog.com/2021/03/20/reliably-debouncing-rotary-encoders-
4  with-arduino-and-esp32/
5
6  https://chome.nerpa.tech/mcu/reading-rotary-encoder-on-arduino/ and here
7  https://chome.nerpa.tech/mcu/rotary-encoder-interrupt-service-routine-for-avr-
8  micros/
9
10 This example does not use the port read method. Tested with Nano and ESP32
11
12 Connections
13 =====
14 Encoder | ESP32 |
15 -----
16 A-DT      | 21 |
17 B-CLK      | 32 |
18 C-SW      | 25 |
19 */
20 // Define rotary encoder pins
21 #define ENC_A_DT 21
22 #define ENC_B_CLK 32
23 #define ENC_C_SW 25
24
25 volatile int counter = 0;
26
27 void setup() {
28     // Set encoder pins
29     pinMode(ENC_A_DT, INPUT_PULLUP);
30     pinMode(ENC_B_CLK, INPUT_PULLUP);
31     pinMode(ENC_C_SW, INPUT_PULLUP);
32     // Start the serial monitor to show output
33     Serial.begin(115200); // 115200 for ESP32
34     delay(500); // Wait for serial to start
35     Serial.println("Start");
36 }
37
38 void loop() {
39     static int lastCounter = 0;
40
41     read_encoder();
42
43     // If count has changed print the new value to serial
44     if(counter != lastCounter){
45         Serial.println(counter);
46         lastCounter = counter;
47     }
48 }
49
50 void read_encoder() {
51     // Encoder routine. Updates counter if they are valid
52     // and if rotated a full indent
53
54     static uint8_t old_AB = 3; // Lookup table index
55     static int8_t encval = 0; // Encoder value
56     static const int8_t enc_states[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0}; //
57     // Lookup table
58
59     old_AB <=> 2; // Remember previous state
60 }
```



```
62 if (digitalRead(ENC_A_DT)) old_AB |= 0x02; // Add current state of pin A
64 if (digitalRead(ENC_B_CLK)) old_AB |= 0x01; // Add current state of pin B
66 encval += enc_states[( old_AB & 0x0f )];
68 // Update counter if encoder has rotated a full indent, that is at least 4 steps
69 if( encval > 3 ) { // Four steps forward
70     counter--; // Increase counter
71     encval = 0;
72 }
73 else if( encval < -3 ) { // Four steps backwards
74     counter++; // Decrease counter
75     encval = 0;
76 }
77
78 if (digitalRead(ENC_C_SW)== LOW) {Serial.println("Pushed - Counter Reset");
79     counter=0;Serial.println(counter);};
80 }
```

Listing 7: RotEncoder.ino

andere Variante mit Library *SimpleRotary.h*: [RotEncoderCW-CCW-CounterSerMon.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/RotEncoderCW-CCW-CounterSerMon/RotEncoderCW-CCW-CounterSerMon.ino>

```
1 /*
2 =====
3
4 SimpleRotary.h Library Options Example Sketch
5 Learn more at [https://github.com/mprograms/SimpleRotary]
6 This example shows how to set various options.
7 =====
8 PREPARATION
9 =====
10 File: RotEncoderCW-CCW-CounterSerMon.ino
11 Pixelring needs 5V from DC-DC Step-Down Module
12
13 activate Power: Place red Jumper over white Field on:
14
15     RotaryEncoder 3V3
16
17 Connections with KY-040:
18     DT to GPIO32
19     CLK to GPIO21
20     SW to GPIO25
21
22 Connections with EC11:
23     A to GPIO32
24     B to GPIO21
25     C to GPIO25
26
27
28 Function: By turning the rotaryEncoder you can see the counter in Serial Monitor
29
30
31
32 */
33 #include <SimpleRotary.h>
34
35 // Pin A, Pin B, Button Pin
36 SimpleRotary rotary(32,21,25);
```



```
38 int z = 0;
40
42 void setup() {
44     // Set the trigger to be either a HIGH or LOW pin (Default: HIGH)
44     // Note this sets all three pins to use the same state.
44     rotary.setTrigger(HIGH);
46
46     // Set the debounce delay in ms (Default: 2)
46     rotary.setDebounceDelay(5);
48
48     // Set the error correction delay in ms (Default: 200)
48     rotary.setErrorDelay(250);
50     Serial.begin(115200);
52 }
54
54 void loop() {
56     byte i,k;
56     i = rotary.rotate();
58     k = rotary.push();
58     // Only print CW / CCW output to prevent an endless stream of output.
60     if ( i == 1 ) {
60         Serial.println(z++);
62     }
62     if ( i == 2 ) {
64         Serial.println(z--);
64     }
66     if ( k == 1 ) {
66         Serial.println("Pushed - Counter Reset");
68         z = 0;
68         Serial.println(z);
70     }
70 }
```

Listing 8: RotEncoderCW-CCW-CounterSerMon.ino



```
RotEncoderCW-CCW-CounterSerMon | Arduino 1.8.16
File Edit Sketch Tools Help

COM3

RotEncoderCW-CCW-CounterSerMon
44 // Note this s
45 rotary.setTrig
46
47 // Set the deb
48 rotary.setDebo
49
50 // Set the err
51 rotary.setErro
52 Serial.begin(1
53 }
54
55 void loop() {
56   byte i,k;
57   i = rotary.rot
58   k = rotary.pus
59   // Only print
60   if ( i == 1 )
61     Serial.print
62 }
```

Pushed - Counter Reset Knob pressed

turn clockwise

turn counter clockwise

☒ Autoscroll ☐ Show timestamp

Abbildung 25: Counter beim Drehen des Rotary Encoders

8.6 NeoPixel-Ring mit Rotary Encoder steuern

Durch Drehung des Rotary-Encoders wird jedes Pixel des Ringes zum Leuchten gebracht. Die Drehrichtung wird auf den Ring abgebildet.

Durch Drücken des Knopfes werden nacheinander im Uhrzeigersinn alle LEDs zum Leuchten gebracht.

Laden Sie auch in der Arduino-IDE die Libraries ADAFRUIT_NEOPixel und SIMPLEROTARY.



Immer, wenn der Neopixel-Ring verwendet wird, muss ein externer DC-Adapter (9-20V) das Board mit Spannung versorgen. Ohne angeschlossenen DC-Adapter werden die LEDs nicht leuchten. Der Adapter sollte für mindestens 2-3A ausgelegt sein.

Das Step-Down-Modul auf dem Board versorgt die LEDs mit 5V Spannung. Der Strombedarf aller LEDs beträgt ca. 200mA.

Source-Code [PixelringCtrlRotEncSimple.ino](https://github.com/mprograms/SimpleRotary):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/PixelringCtrlRotEncSimple/PixelringCtrlRotEncSimple.ino>

```
1 /*
=====
3 SimpleRotary.h Library Options Example Sketch
Learn more at [https://github.com/mprograms/SimpleRotary]
5 This example shows how to set various options.
=====
```



```
7 PREPARATION
=====
9 Plug in DC Power Adapter - we need 5V from DC-DC-Step-Down-Module

11 activate Power: Place red Jumper over white Field on:
    PixelRing 5Vext
13    RotaryEncoder 3V3

15 Connections with KY-040:
    DT to GPIO32
17    CLK to GPIO21
    SW to GPIO25
19    NeoPixel Din to GPIO5

21 Connections with EC11:
    A to GPIO32
23    B to GPIO21
    C to GPIO25
25    NeoPixel Din to GPIO5

27
Function: By turning the rotaryEncoder the Pixels light up each after the other as
    long as you turn. Direction is considered.

29

31
*/
33
#include <SimpleRotary.h>
35 // Pin A (DT), Pin B (CLK), Button Pin(SW)
SimpleRotary rotary(32,21,25);
37

39 #include <Adafruit_NeoPixel.h>
#define PIXEL_PIN 5 // Digital IO pin connected to the NeoPixels.
41 #define PIXEL_COUNT 12 // Number of NeoPixels 12 or 16
// Declare our NeoPixel strip object:
43 Adafruit_NeoPixel strip(PIXEL_COUNT, PIXEL_PIN, NEO_GRB + NEO_KHZ800);
// Argument 1 = Number of pixels in NeoPixel strip
45 // Argument 2 = Arduino pin number (most are valid)
// Argument 3 = Pixel type flags, add together as needed:
47 // NEO_KHZ800 800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
// NEO_KHZ400 400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
49 // NEO_GRB Pixels are wired for GRB bitstream (most NeoPixel products)
// NEO_RGB Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
51 // NEO_RGBW Pixels are wired for RGBW bitstream (NeoPixel RGBW products)

53
int z = 0;
55

57 void setup() {
    strip.begin(); // Initialize NeoPixel strip object (REQUIRED)
59    strip.show(); // Initialize all pixels to 'off'

61    // Set the trigger to be either a HIGH or LOW pin (Default: HIGH)
    // Note this sets all three pins to use the same state.
63    rotary.setTrigger(HIGH);

65    // Set the debounce delay in ms (Default: 2)
    rotary.setDebounceDelay(5);
67

    // Set the error correction delay in ms (Default: 200)
69    rotary.setErrorDelay(250);
```



```
71 }  
72 Serial.begin(115200);  
73 void loop() {  
74     byte i,k;  
75     i = rotary.rotate();  
76     k = rotary.push();  
77     // Only print CW / CCW output to prevent an endless stream of output.  
78     if ( i == 1 ) {  
79         Serial.println(z++);  
80         if ( z > (PIXEL_COUNT-1) ) {z=0;}  
81         strip.setPixelColor(z, strip.Color(255, 0, 0)); strip.show();delay(50);  
82         strip.setPixelColor(z-1, strip.Color(0, 0, 0)); strip.show();delay(50);  
83         if ( z == 0 ) {strip.setPixelColor((PIXEL_COUNT-1), strip.Color(0, 0, 0));  
84         strip.show();delay(50); }  
85         if ( z > (PIXEL_COUNT-1) ) {strip.show();} // Initialize all pixels to 'off'  
86     }  
87     if ( i == 2) {  
88         Serial.println(z--);  
89         if ( z < 0 ) {z=PIXEL_COUNT-1;}  
90         strip.setPixelColor(z, strip.Color(255, 0, 0)); strip.show();delay(50);  
91         strip.setPixelColor(z+1, strip.Color(0, 0, 0)); strip.show();delay(50);  
92         if ( z == PIXEL_COUNT-1 ) {strip.setPixelColor((0), strip.Color(0, 0, 0));  
93         strip.show();delay(50); }  
94     }  
95  
96  
97     if ( k == 1 ) {  
98  
99         for(int j=0; j<strip.numPixels(); j++) { // For each pixel in strip...  
100         strip.setPixelColor(j, strip.Color(0, 0, 0)); // Set pixel's color  
101         (in RAM)  
102         strip.show(); // Update strip to match  
103         delay(50); // Pause for a moment  
104     }  
105  
106     Serial.println("Pushed - Counter Reset");  
107     z = 0;  
108     Serial.println(z);  
109     for(int j=0; j<strip.numPixels(); j++) { // For each pixel in strip...  
110     strip.setPixelColor(j, strip.Color(0, 255, 0)); // Set pixel's  
111     color (in RAM)  
112     strip.show(); // Update strip to match  
113     delay(50); // Pause for a moment  
114 }  
115 }
```

Listing 9: [PixelringCtrlRotEncSimple.ino](#)

8.7 NeoPixel-Choreografien

Mit dem Taster B1 können 8 Choreografien sequentiell abgerufen werden. Der Taster B1 *B1 out* wird mit GPIO25 und das *Neopixel Din* mit GPIO 5 verbunden.

Source-Code [PixelringButtonB1Cycler.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/PixelringButtonB1Cycler/PixelringButtonB1Cycler.ino>



```
=====
3 PREPARATION
=====
5
7 activate Power: Place red Jumper over white Field on:
   PixelRing 5Vext
9
11 Connections:
   B1 out      to GPIO 25
   NeoPixel Din to GPIO 5
13
15 *
   Load Sketch and press B1 in order to see 8 choreografies
   */
17
19
21 // Simple demonstration on using an input device to trigger changes on your
23 // NeoPixels. Wire a momentary push button to connect from ground to a
25 // digital IO pin. When the button is pressed it will change to a new pixel
27 // animation. Initial state has all pixels off -- press the button once to
29 // start the first animation. As written, the button does not interrupt an
31 // animation in-progress, it works only when idle.
33
35 #include <Adafruit_NeoPixel.h>
37 #ifndef __AVR__
39   #include <avr/power.h> // Required for 16 MHz Adafruit Trinket
41 #endif
43
45 // Digital IO pin connected to the button. This will be driven with a
47 // pull-up resistor so the switch pulls the pin to ground momentarily.
49 // On a high -> low transition the button press logic will execute.
51 #define BUTTON_PIN 25
53
55 #define PIXEL_PIN 5 // Digital IO pin connected to the NeoPixels.
57
59 #define PIXEL_COUNT 12 // Number of NeoPixels 12 or 16
61
63 // Declare our NeoPixel strip object:
65 Adafruit_NeoPixel strip(PIXEL_COUNT, PIXEL_PIN, NEO_GRB + NEO_KHZ800);
// Argument 1 = Number of pixels in NeoPixel strip
// Argument 2 = Arduino pin number (most are valid)
// Argument 3 = Pixel type flags, add together as needed:
// NEO_KHZ800 800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
// NEO_KHZ400 400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
// NEO_GRB Pixels are wired for GRB bitstream (most NeoPixel products)
// NEO_RGB Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
// NEO_RGBW Pixels are wired for RGBW bitstream (NeoPixel RGBW products)

67 boolean oldState = HIGH;
69 int mode = 0; // Currently-active animation mode, 0-9
71
73 void setup() {
75   pinMode(BUTTON_PIN, INPUT_PULLUP);
77   strip.begin(); // Initialize NeoPixel strip object (REQUIRED)
79   strip.show(); // Initialize all pixels to 'off'
81 }
83
85 void loop() {
87   // Get current button state.
89   boolean newState = digitalRead(BUTTON_PIN);
91 }
```



```
67 // Check if state changed from high to low (button press).
68 if((newState == LOW) && (oldState == HIGH)) {
69     // Short delay to debounce button.
70     delay(20);
71     // Check if button is still low after debounce.
72     newState = digitalRead(BUTTON_PIN);
73     if(newState == LOW) { // Yes, still low
74         if(++mode > 8) mode = 0; // Advance to next mode, wrap around after #8
75         switch(mode) { // Start the new animation...
76             case 0:
77                 colorWipe(strip.Color( 0, 0, 0), 50); // Black/off
78                 break;
79             case 1:
80                 colorWipe(strip.Color(255, 0, 0), 50); // Red
81                 break;
82             case 2:
83                 colorWipe(strip.Color( 0, 255, 0), 50); // Green
84                 break;
85             case 3:
86                 colorWipe(strip.Color( 0, 0, 255), 50); // Blue
87                 break;
88             case 4:
89                 theaterChase(strip.Color(127, 127, 127), 50); // White
90                 break;
91             case 5:
92                 theaterChase(strip.Color(127, 0, 0), 50); // Red
93                 break;
94             case 6:
95                 theaterChase(strip.Color( 0, 0, 127), 50); // Blue
96                 break;
97             case 7:
98                 rainbow(10);
99                 break;
100            case 8:
101                theaterChaseRainbow(50);
102                break;
103        }
104    }
105    // Set the last-read button state to the old state.
106    oldState = newState;
107 }
108
109 // Fill strip pixels one after another with a color. Strip is NOT cleared
110 // first; anything there will be covered pixel by pixel. Pass in color
111 // (as a single 'packed' 32-bit value, which you can get by calling
112 // strip.Color(red, green, blue) as shown in the loop() function above),
113 // and a delay time (in milliseconds) between pixels.
114 void colorWipe(uint32_t color, int wait) {
115     for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...
116         strip.setPixelColor(i, color); // Set pixel's color (in RAM)
117         strip.show(); // Update strip to match
118         delay(wait); // Pause for a moment
119     }
120 }
121
122 // Theater-marquee-style chasing lights. Pass in a color (32-bit value,
123 // a la strip.Color(r,g,b) as mentioned above), and a delay time (in ms)
124 // between frames.
125 void theaterChase(uint32_t color, int wait) {
126     for(int a=0; a<10; a++) { // Repeat 10 times...
127         for(int b=0; b<3; b++) { // 'b' counts from 0 to 2...
128             strip.clear(); // Set all pixels in RAM to 0 (off)
129             // ... (rest of the function code) ...
130         }
131         delay(wait);
132     }
133 }
```




```
131 // 'c' counts up from 'b' to end of strip in steps of 3...
132 for(int c=b; c<strip.numPixels(); c += 3) {
133     strip.setPixelColor(c, color); // Set pixel 'c' to value 'color'
134 }
135 strip.show(); // Update strip with new contents
136 delay(wait); // Pause for a moment
137 }
138 }
139
140 // Rainbow cycle along whole strip. Pass delay time (in ms) between frames.
141 void rainbow(int wait) {
142     // Hue of first pixel runs 3 complete loops through the color wheel.
143     // Color wheel has a range of 65536 but it's OK if we roll over, so
144     // just count from 0 to 3*65536. Adding 256 to firstPixelHue each time
145     // means we'll make 3*65536/256 = 768 passes through this outer loop:
146     for(long firstPixelHue = 0; firstPixelHue < 3*65536; firstPixelHue += 256) {
147         for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...
148             // Offset pixel hue by an amount to make one full revolution of the
149             // color wheel (range of 65536) along the length of the strip
150             // (strip.numPixels() steps):
151             int pixelHue = firstPixelHue + (i * 65536L / strip.numPixels());
152             // strip.ColorHSV() can take 1 or 3 arguments: a hue (0 to 65535) or
153             // optionally add saturation and value (brightness) (each 0 to 255).
154             // Here we're using just the single-argument hue variant. The result
155             // is passed through strip.gamma32() to provide 'truer' colors
156             // before assigning to each pixel:
157             strip.setPixelColor(i, strip.gamma32(strip.ColorHSV(pixelHue)));
158         }
159         strip.show(); // Update strip with new contents
160         delay(wait); // Pause for a moment
161     }
162 }
163
164 // Rainbow-enhanced theater marquee. Pass delay time (in ms) between frames.
165 void theaterChaseRainbow(int wait) {
166     int firstPixelHue = 0; // First pixel starts at red (hue 0)
167     for(int a=0; a<30; a++) { // Repeat 30 times...
168         for(int b=0; b<3; b++) { // 'b' counts from 0 to 2...
169             strip.clear(); // Set all pixels in RAM to 0 (off)
170             // 'c' counts up from 'b' to end of strip in increments of 3...
171             for(int c=b; c<strip.numPixels(); c += 3) {
172                 // hue of pixel 'c' is offset by an amount to make one full
173                 // revolution of the color wheel (range 65536) along the length
174                 // of the strip (strip.numPixels() steps):
175                 int hue = firstPixelHue + c * 65536L / strip.numPixels();
176                 uint32_t color = strip.gamma32(strip.ColorHSV(hue)); // hue -> RGB
177                 strip.setPixelColor(c, color); // Set pixel 'c' to value 'color'
178             }
179             strip.show(); // Update strip with new contents
180             delay(wait); // Pause for a moment
181             firstPixelHue += 65536 / 90; // One cycle of color wheel over 90 frames
182         }
183     }
184 }
```

Listing 10: PixelringButtonB1Cycler.ino

8.8 Buttons UP, DOWN, LEFT, RIGHT und SELECT auswerten

Der beiliegende Sketch wertet die fünf Taster UP, DOWN, LEFT, RIGHT und SELECT aus. Je nach gedrückter Taste, wird eine Spannung am Pin *Buttons_Vout* angelegt. Der Sketch wertet



diese Spannung aus und leitet daraus die gedrückte Taste ab. Dazu wird der analog-digital-Converter des ESP32 verwendet.



Der Analog-Digital-Converter (ADC) des ESP32 ist relativ unpräzise. Es kann daher sein, dass die ADC-Werte für Ihren ESP32 angepasst werden müssen. Die Spannungsangaben auf dem Board entsprechen den gemessenen Werten mit einem Voltmeter. Die Angaben im Skript entsprechen den gemessenen Werten mit dem ESP32. Die individuellen Spannungswerte pro Taster wurden auf dem Board mit einem Spannungsteiler realisiert.

Source-Code [Read5Buttons.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/Read5Buttons/Read5Buttons.ino>

```
/*
2 Title:      read analog Signal from Buttons_Vout and derive pressed Button
  Sketchfile: Read5Buttons.ino
4
  Function:    Read analog Signal from Buttons_Vout and convert it into Voltage,
              using ADC and derive pressed Button
6
  =====
8 Button related Values:
  =====
10 Button      ADC-Value      Volt
  =====
12 no Button   4095           3.3V
13 UP          340-360        0.3V
14 DOWN        1080..1090     0.9V
15 LEFT        1730..1750     1.4V
16 RIGHT       0              0V
17 SELECT      2640..2670     2.2V
18 =====

20 Connections: Buttons_Vout is connected to GPIO 34 (Analog ADC1_CH6)

22 Date:        tja/21.11.2021

24 */

26
27 const int BtnPin = 34;
28
29 // variable for storing the potentiometer value
30 int BtnValue = 0;

31 int btn      = 0;
32 int adc_key_in = 0;
33 #define btnRIGHT  0
34 #define btnUP     1
35 #define btnDOWN   2
36 #define btnLEFT   3
37 #define btnSELECT 4
38 #define btnNONE   5
39
40
41
42 void setup() {
43     Serial.begin(115200);
44     delay(1000);
45     analogSetPinAttenuation(BtnPin, ADC_11db);
46     analogReadResolution(12);
```



```
48     analogSetWidth(12);
49 }
50 // read the buttons
52 int read_buttons()
53 {
54     adc_key_in = analogRead(BtnPin);    // read the value from the sensor
55     // float voltage = adc_key_in * (3.3 / 4095.0);
56     // Serial.print(adc_key_in); Serial.print(" relates to ");Serial.print(voltage);
57     Serial.println(" Volt");
58     // my buttons when read are centered at these values: 0, 144, 329, 504, 741
59     // we add approx 50 to those values and check to see if we are close
60     if (adc_key_in > 3500) return btnNONE; // We make this the 1st option for speed
61     // reasons since it will be the most likely result
62     // For V1.1 us this threshold
63     if (adc_key_in < 50)    return btnRIGHT;
64     if (adc_key_in < 500)   return btnUP;
65     if (adc_key_in < 1500)  return btnDOWN;
66     if (adc_key_in < 1800)  return btnLEFT;
67     if (adc_key_in < 3000)  return btnSELECT;
68
69     return btnNONE; // when all others fail, return this...
70 }
71
72 void loop() {
73     btn = read_buttons(); // read the buttons
74     // Reading Btn value
75     switch (btn)          // depending on which button was pushed, we perform an
76         action
77     {
78         case btnRIGHT:
79         {
80             Serial.println("RIGHT ");
81             break;
82         }
83         case btnLEFT:
84         {
85             Serial.println("LEFT  ");
86             break;
87         }
88         case btnUP:
89         {
90             Serial.println("UP    ");
91             break;
92         }
93         case btnDOWN:
94         {
95             Serial.println("DOWN  ");
96             break;
97         }
98         case btnSELECT:
99         {
100             Serial.println("SELECT");
101             break;
102         }
103         case btnNONE:
104         {
105             Serial.println("NO BUTTON PRESSED ");
106             break;
107         }
108     }
```



```
108 | }  
    | }  
110 | }  
    | }
```

Listing 11: [Read5Buttons.ino](#)

Seite:
38/115

8.9 Motor 1: Typ JGA25-370 oder N20

Auf dem Board ist entweder ein DC-Motor vom Typ 370 oder N20 verbaut. Beide Motortypen werden mit dem Motortreiber [TB6612FNG](#) angesteuert und liefern ein Feedback Signal, welches über die Pins PhA und PhB abgegriffen werden können. Diese Signale entstehen durch rotierende Magnete, welche der Hallsensor des Motors verarbeitet.

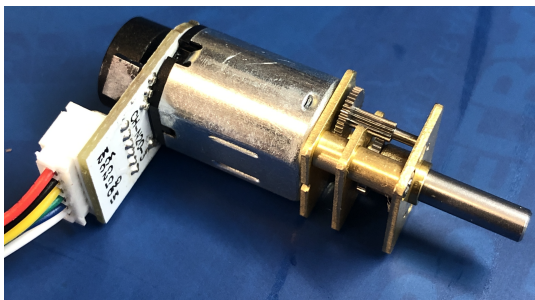


Abbildung 26: DC-Motor Typ N20

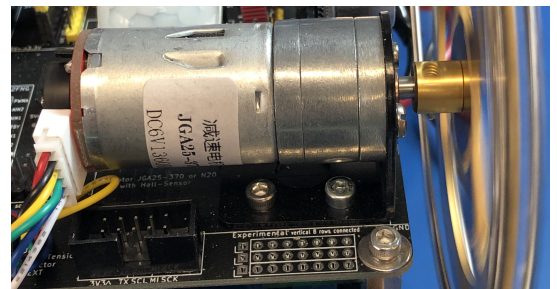


Abbildung 27: DC-Motor Typ 370

Damit der Motor läuft müssen folgende Vorbereitungen getroffen werden:

1. Externe Spannungsquelle muss angeschlossen sein, damit der Motor genügend Strom erhält. Die Spannung für den Motor (5V) wird mit dem Step-Down-Modul aufbereitet. Die Eingangsspannung liefert Ihr DC-Adapter (9-25V).
2. Die Jumper *Motor 5V ext* und *TB6612 3V3* müssen über dem weissen Feld platziert werden. Damit wird die Spannung für den Motor und den Motortreiber eingeschaltet.
3. 4 Kabelverbindung vorbereiten:

Motortreiber-Pin	ESP32-Pin
PWMA	4
AIN1	16
AIN2	17
STDBY	5

Bild zu 1.

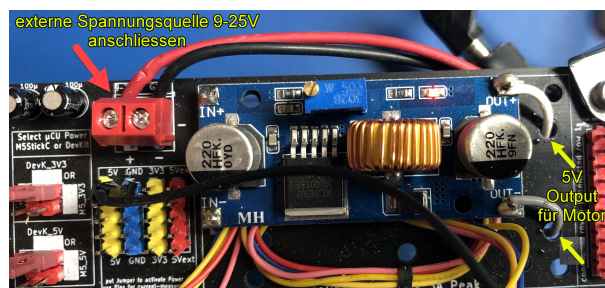


Abbildung 28: ext. Spannungsquelle anschliessen



Bild zu 2.

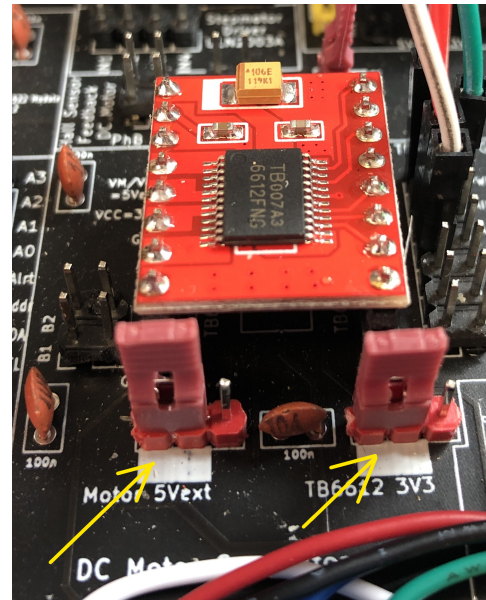


Abbildung 29: Jumper schalten Spannung für Motor und Motortreiber TB6612FNG

Source-Code [MotorA370TB6612FNG.ino](https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/MotorA370TB6612FNG/MotorA370TB6612FNG.ino):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/MotorA370TB6612FNG/MotorA370TB6612FNG.ino>

```
1  /*
   * Function: Motor Control with TB6612FNG-Motor-Driver. Turn Clockwise-STOP-Turn
   *           CounterClock-BREAK-STANDBY- repeat
3  * MCU:      Olimex ESP32-DevKit-LiPo
   * File:      13-TB6612FNG_MotorTests.ino
5  * Connections: AIN1=GPI016; AIN2=GPI017; STDBY=GPI05; PWMA=GPI04
   * Author:    Thomas Jäggi
7  * Date:      30.07.2021
   */
9  #define PWMA_KANAL_0 0
   #define PWMA_AUFLOESUNG 8
11 #define PWMA_FREQ 5000
   #define PWMA_PIN 4 // PWM-Pin
13 #define AIN1 16
   #define AIN2 17
15 #define STDBY 5 // STDBY muss high sein, damit motor läuft
   #define SPEED 255 //0..255; 255 is full-speed
17
   #define CW      (digitalWrite(STDBY,HIGH),digitalWrite(AIN1,HIGH),digitalWrite(AIN2,
   ,LOW),ledcWrite(PWMA_KANAL_0,SPEED))
19 #define CCW     (digitalWrite(STDBY,HIGH),digitalWrite(AIN1,LOW),digitalWrite(AIN2,
   HIGH),ledcWrite(PWMA_KANAL_0,SPEED))
   #define STOP    (digitalWrite(STDBY,HIGH),digitalWrite(AIN1,LOW),digitalWrite(AIN2,
   LOW), ledcWrite(PWMA_KANAL_0,0))
21 #define BRAKE   (digitalWrite(STDBY,HIGH),digitalWrite(AIN1,HIGH),digitalWrite(AIN2,
   LOW), ledcWrite(PWMA_KANAL_0,0))
   #define STANDBY (digitalWrite(STDBY,LOW),digitalWrite(AIN1,HIGH),digitalWrite(AIN2,
   LOW), ledcWrite(PWMA_KANAL_0,0))
23
   // int maxTV = pow(2,PWMA_AUFLOESUNG)-1; // Bei 8 Bit = 0-255
25
27 /* TB6612FNG Control Function
   *      INPUT      OUTPUT
   *      INA1  INA2  PWM Mode
   *      =====
31 *      H      H      H/L Short Brake
```




```
33 * L H H CCW
34 * L H L Short Brake
35 * H L H CW
36 * H L L Short Brake
37 * L L H Stop
38 *
39 * https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/
40 * DatasheetsComponents/TB6612FNG_Datasheet.pdf
41 */
42
43 void setup() {
44   pinMode(AIN1, OUTPUT);
45   pinMode(AIN2, OUTPUT);
46   pinMode(STDBY, OUTPUT);
47   pinMode(PWMA_PIN, OUTPUT);
48   ledcSetup(PWMA_KANAL_0, PWMA_FREQ, PWMA_AUFLOESUNG);
49   ledcAttachPin(PWMA_PIN, PWMA_KANAL_0);
50 }
51
52
53
54 void loop() {
55   CW;
56   delay(5000);
57   STOP;
58   delay(1000);
59   CCW;
60   delay(2000);
61   BRAKE;
62   delay(1000);
63   STANDBY;
64   delay(5000);
65 }
```

Listing 12: [MotorA370TB6612FNG.ino](#)

8.10 Motor 1: Hall-Phasensignale *PhA* und *PhB* auswerten

Je nachdem, ob der Motor vorwärts oder rückwärts dreht, kann eine entsprechende Phasenverschiebung an den Ausgängen *PhA* und *PhB* beobachtet werden:

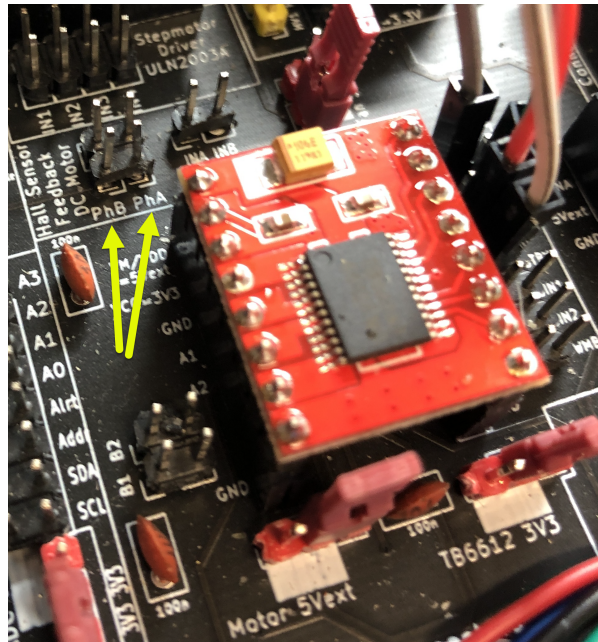


Abbildung 30: Phasensignale *PhA* und *PhB* vom Motor-Hall-Sensor

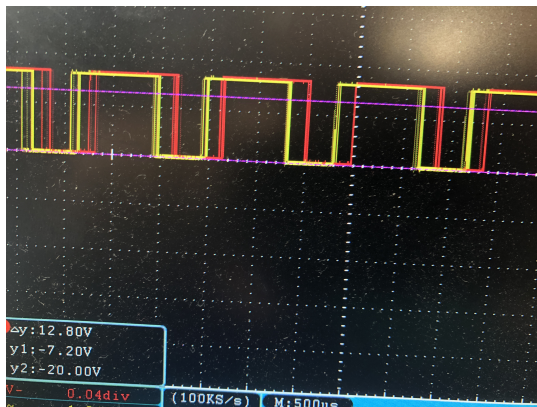


Abbildung 31: Vorwärts: Phase *PhA*-gelb ist vor Phase *PhB*-rot

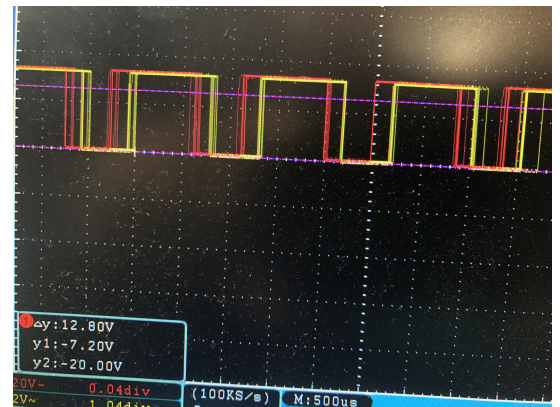


Abbildung 32: Rückwärts: Phase *PhB*-rot ist vor Phase *PhA*-gelb

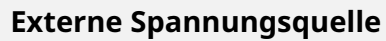
Diese Eigenschaften können zusammen mit der Signalfrequenz für die Ermittlung der Geschwindigkeit und der Drehrichtung verwendet werden.

8.11 Motor 2: Lüfter mit Propellermodul L9110

Das Lüfter-Modul besteht aus einem Propeller mit Gleichstrommotor. Als Motortreiber wird der [L9110H](#) eingesetzt. Dieser ist auf dem Modul verbaut.



Abbildung 33: Fan Modul mit integriertem Motor-Treiber L9110H



INB to GPIO17

Fan Module Power on

INA to GPIO16

Abbildung 34: Fan Modul: Power aktivieren und Verbindungen stecken

Source-Code [FanModuleWithL9110HDriver.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/FanModuleWithL9110HDriver/FanModuleWithL9110HDriver.ino>



```
1  /*
   * Function:  Fan Module with L9110H-Motor-Driver
3  * MCU:      Olimex ESP32-DevKit-LiPo
   * Connections: Fan Ctrl INA with GPIO 16, Fan Ctrl INB with GPIO17
5  * Power On:  Jumper over white Marker Pins "Fan 5Vext"
   * File:      FanModuleWithL9110HDriver.ino
7  *
   * Author:    Thomas Jäggi
9  * Date:      30.07.2021
   */
11 #define PWM_KANAL_0 0
   #define PWM_KANAL_1 1
13 #define PWM_AUFLOESUNG 8
   #define PWM_FREQ 5000
15 #define INA_PIN 16 // PWM-Pin
   #define INB_PIN 17 // PWM-Pin
17
19
   #define CW      (digitalWrite(INA_PIN,LOW),ledcWrite(INB_KANAL_0,255))
21 #define CCW      (digitalWrite(INA_PIN,HIGH),ledcWrite(INB_KANAL_0,0))
   #define STOP     (digitalWrite(INA_PIN,LOW),digitalWrite(INB_PIN,LOW))
23
   // int maxTV = pow(2,PWMA_AUFLOESUNG)-1; // Bei 8 Bit = 0-255
25
27 /*      L9110H Control
   *      INPUT
29  *  INA  INA  Function
   *  =====
31  *  L    L    OFF
   *  H    L    FORWARD CW
33  *  L    H    REVERSE CCW
   *  H    H    OFF
35  */
37 int maxspeed = 255;
   int minspeed = 0;
39 int speed=0;
41 void setup() {
   pinMode(INA_PIN,OUTPUT);
43 pinMode(INB_PIN,OUTPUT);
   ledcSetup(PWM_KANAL_0,PWM_FREQ,PWM_AUFLOESUNG);
45 ledcSetup(PWM_KANAL_1,PWM_FREQ,PWM_AUFLOESUNG);
   ledcAttachPin(INA_PIN, PWM_KANAL_0);
47 ledcAttachPin(INB_PIN, PWM_KANAL_1);
   }
49
51
   void loop() {
53 // Turn Fullspeed Clockwise CW
   ledcWrite(PWM_KANAL_0,maxspeed); // INA = 255
55 ledcWrite(PWM_KANAL_1,(0)); // INB = 0
   delay(2000);
57 // Motor OFF
   ledcWrite(PWM_KANAL_0,0); // INA = 0
59 ledcWrite(PWM_KANAL_1,0); // INB = 0
   delay(2000);
61 // Turn Fullspeed Counter-Clockwise CCW
   ledcWrite(PWM_KANAL_0,0); // INA = 0
63 ledcWrite(PWM_KANAL_1,maxspeed); // INB = 255
```




```
        delay(2000);
65 // Motor OFF
    ledcWrite(PWM_KANAL_0,0);           // INA = 0
67 ledcWrite(PWM_KANAL_1,0);           // INB = 0
    delay(2000);
69 // Increase Fan from minspeed to maxspeed - Clockwise
    for (int i = 100; i <= 255; i++) {
71     ledcWrite(PWM_KANAL_0,i);         // INA = 0..255
        ledcWrite(PWM_KANAL_1,(0));     // INB = 0
73     delay(100);
    }
75
    // Decrease Fan from Fullspeed to minspeed - Clockwise
77 for (int i = 255; i >= 0; i--) {
    ledcWrite(PWM_KANAL_0,i);           // INA = 255..0
79     ledcWrite(PWM_KANAL_1,0);         // INB = 0
        delay(100);
81 }

83 // Increase Fan from minspeed to maxspeed - Counter-Clockwise
    for (int i = 100; i <= 255; i++) {
85     ledcWrite(PWM_KANAL_0,0);         // INA = 0
        ledcWrite(PWM_KANAL_1,i);       // INB = 0..255
87     delay(100);
    }
89
    // Decrease Fan from Fullspeed to minspeed - Counter-Clockwise
91 for (int i = 255; i >= 0; i--) {
    ledcWrite(PWM_KANAL_0,0);           // INA = 0
93     ledcWrite(PWM_KANAL_1,i);         // INB = 255..0
        delay(100);
95 }
97
99 delay(1000);
101 }
```

Listing 13: [FanModuleWithL9110HDriver.ino](#)

8.12 Motor 3: Schrittmotor 28BYJ-48 und ULN2003 Motor Driver

In diesem Demo-Sketch verwenden wir den unipolaren Schrittmotor [28BYJ-48](#). Der Motor Driver ist ein IC mit dem Namen [ULN2003](#).

Der Schrittmotor hat folgende Eigenschaften:

- › Total steps pro Umdrehung = 2048 steps
- › Step angle = 0.18° /step

Folgende Vorbereitungen sind notwendig:

Power-Jumper aktivieren (Jumper über das weisse Feld platzieren) für:

- › *Stepper 5Vext*

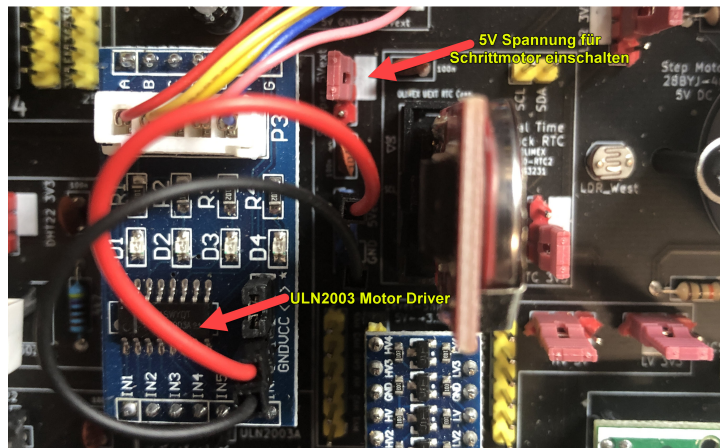


Abbildung 35: Spannung für Schrittmotor wird hier eingeschaltet
Verbindungen mit Jumper-Kabel stecken:

- › IN1 mit GPIO19
- › IN2 mit GPIO18
- › IN3 mit GPIO5
- › IN4 mit GPIO17

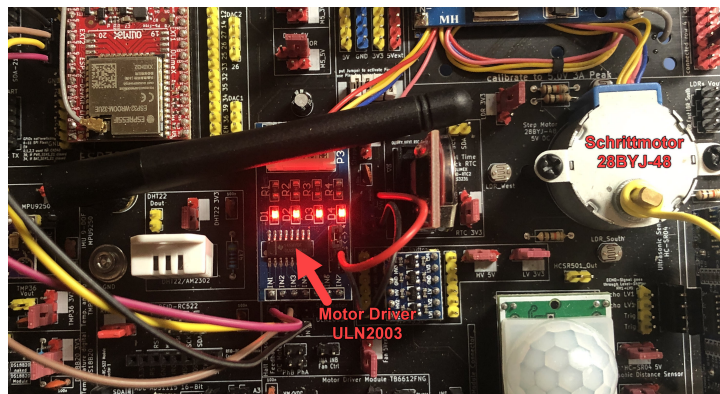


Abbildung 36: Aufbau der Schaltung
Source-Code [StepMotor28BYJ-48.ino](https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/StepMotor28BYJ-48/StepMotor28BYJ-48.ino):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/StepMotor28BYJ-48/StepMotor28BYJ-48.ino>

```
/*
2 Title:      28BYJ-48 Stepper Motor
4 Sketchfile: StepMotor28BYJ-48.ino
6 Libraries used: Stepper built-in by Arduino
8 Function:   In this sketch, you will learn how to control the 28BYJ-48 unipolar
             stepper motor with the ULN2003 motor driver.
10 Activate Power on Stepper 5Vext
12 Connections: Motor Driver      ESP32
               =====
14               IN1                GPIO19
16               IN2                GPIO18
18               IN3                GPIO5
20               IN4                GPIO17

             siehe auch: https://randomnerdtutorials.com/esp32-stepper-motor-28byj-48-uln2003/
```



```

22  Date:          tja/05.12.2021
23  */
24  #include <Stepper.h>
25
26  const int stepsPerRevolution = 2048;  // per step: 0.18 Degrees
27
28  // ULN2003 Motor Driver Pins
29  #define IN1 19
30  #define IN2 18
31  #define IN3 5
32  #define IN4 17
33
34  // initialize the stepper library
35  Stepper myStepper(stepsPerRevolution, IN1, IN3, IN2, IN4);
36
37  void setup() {
38    // set the speed at 5 rpm
39    myStepper.setSpeed(5);
40    // initialize the serial port
41    Serial.begin(115200);
42  }
43
44  void loop() {
45    // step one revolution in one direction:
46    Serial.println("counterclockwise");
47    myStepper.step(stepsPerRevolution);
48    delay(1000);
49
50    // step one revolution in the other direction:
51    Serial.println("clockwise");
52    myStepper.step(-stepsPerRevolution);
53    delay(1000);
54  }
```

Listing 14: [StepMotor28BYJ-48.ino](#)

8.13 Analoger Temperatursensor TMP36

Der Temperatursensor TMP36 ist ein analoger Sensor. Er liefert eine Spannung, welche proportional zur Umgebungstemperatur ist. Der lineare Zusammenhang zwischen Spannung und Temperatur ist in folgender Grafik ersichtlich.

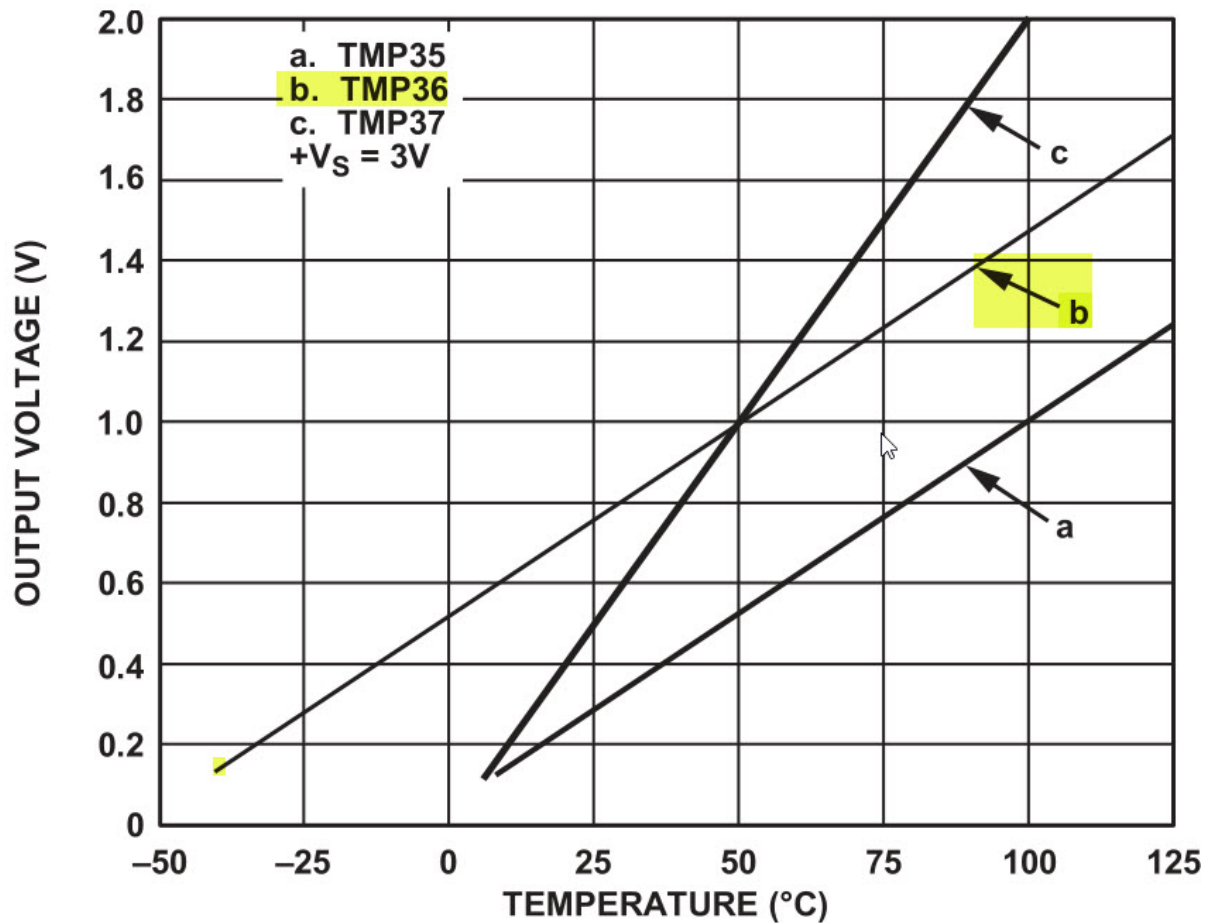


Abbildung 37: Kennlinie des Sensors TMP36

Um die Temperatur zu ermitteln, muss die Spannung gelesen und daraus die Temperatur berechnet werden. Da es sich um einen linearen Zusammenhang handelt, ist die Formel einfach:

$$\text{Temperatur [Grad Celsius]} = (\text{Spannung[V]} - 0.5) * 100$$

Um eine Spannung zu messen, benötigen wir ein Analog-Digital-Converter (ADC). Der ESP32 hat einen ADC eingebaut. Leider ist der nicht präzise genug, um eine genügend genaue Temperatur zu berechnen. Als Beispiel hier die gemessenen Werte des eingebauten ADC und der effektiven Spannung, gemessen mit einem Voltmeter.

Spannung [V] mit ADC ESP32	effektive Spannung [V]
0.2	0.327
0.4	0.550
0.5	0.632
0.7	0.835
1	1.138
1.3	1.442
1.6	1.759
2.0	2.155
3.0	2.963
3.2	3.082

Wenn wir die Kennlinie des TMP36 betrachten, sehen wir, dass eine Steigung besteht:



kleine Voltänderungen haben eine grosse Auswirkung auf die daraus berechnete Temperatur: 0.7V entspricht ca 20 Grad Celsius und 0.8V entspricht bereits ca 28 Grad Celsius. Der eingebaute ADC des ESP32 eignet sich also für diese Anwendung nicht! Wir benötigen einen viel präziseren Analog-Digital-Converter. Dies ist der ADS1115. Komponente Nr. 19, siehe Abbildung 4.

Hier Vergleichsmessungen mit dem ADS1115. Die Abweichungen sind jetzt viel geringer. Damit ist es möglich, Temperaturen zu messen, die nahe an der Realität sind.

Spannung [V] mit ADS1115	effektive Spannung [V]
0.203	0.200
0.400	0.392
0.443	0.434
0.506	0.496
0.599	0.586
0.699	0.684
0.711	0.696
0.718	0.703
0.739	0.723
0.756	0.740
1.009	0.987
1.300	1.272
1.600	1.565
2.034	1.989
2.996	2.928
3.3	3.225

8.13.1 TMP36 mit ADC ADS1115 auslesen

Hier wird die analoge Spannung des TMP36 mit einem präzisen Analog-Digital-Converter ADS1115 gemessen. Der ADS1115 ist ein 16-Bit ADC. Der ADC kann so konfiguriert werden, dass die 16-Bit auf den 1 Volt Spannungsbereich des TMP36 abgebildet werden können. Siehe dazu das zweite Listing in diesem Abschnitt.

Damit der ADS1115 den Spannungswert des TMP36 verarbeiten kann, müssen folgende 3 Verbindungen gemacht werden:

Pin *TMP36 Vout* → ADS1115 *A0*

ADS1115 *SCL* → ESP32 *SCL*

ADS1115 *SDA* → ESP32 *SDA*

Details zum ADS1115 findet sich hier:

<https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/>

<https://microcontrollerslab.com/ads1115-external-adc-with-esp32/>

Hier der Beispiel-Sketch zum Lesen des analogen Signals des TMP36 mit dem ADC ADS1115 und der Umrechnung in Grad Celsius gemäss der Kennlinie.

Source-Code [TMP36ADC1115SingleEnded.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/TMP36ADC1115SingleEnded/TMP36ADC1115SingleEnded.ino>



```
1  /*
   Title:      read analog Signal from TMP36 with ADS1115 - Analog Digital Converter
3
   Sketchfile: TMP36ADC1115SingleEnded.ino
5
   Libraries used: Wire for I2C, Adafruit_ADS1X15 for ADS1115
7
   Function:   Read analog Signal from TMP36 and convert it into Voltage, using
               ADS1115.
9               Temperature is derived from Voltage with Formula: temp = (voltage -
               0.5) * 100 ;
11
   Connections: TMP36 Vout is connected to GPIO34 (Analog ADC1_CH6)
13
   Date:      tja/04.12.2021
15
   */
17
19
21 #include <Wire.h>
   #include <Adafruit_ADS1X15.h>
23
   Adafruit_ADS1115 ads1115;
25
   float volts0=0.000;
   float volts1=0.000;
27   float temp=0.000;
29
   void setup(void)
31 {
   Serial.begin(115200);
33   Serial.println("Hello!");
35
   Serial.println("Getting single-ended readings from AIN0..3");
   Serial.println("ADC Range: +/- 6.144V (1 bit = 3mV)");
37   ads1115.begin();
   }
39
   void loop(void)
41 {
   int16_t adc0;
43
   adc0 = ads1115.readADC_SingleEnded(0);
45   //adc1 = ads1115.readADC_SingleEnded(1);
   //adc2 = ads1115.readADC_SingleEnded(2);
47   //adc3 = ads1115.readADC_SingleEnded(3);
49
   //volts1 = ads1115.computeVolts(adc0);
   volts0 = (ads1115.computeVolts(adc0)) * 1.028206821282401; //Korrekturwert für
   Raumtemperatur. experimentell berechnet.
51   //volts2 = ads1115.computeVolts(adc2);
   //volts3 = ads1115.computeVolts(adc3);
53
   temp = (volts0 - 0.5) * 100 ;
55
   //Serial.print("AIN0: "); Serial.print(adc0);Serial.print(" ");Serial.println(
   volts0,3);
57   //Serial.print("AIN1: "); Serial.print(adc1);Serial.print(" relates to ");
   Serial.print(volts0,3);Serial.print(" ->");Serial.print(temp,3);Serial.println("
   *C");
59   //Serial.print("AIN2: "); Serial.println(adc2);
```




```
61 //Serial.print("AIN3: "); Serial.println(adc3);  
Serial.println(" ");  
63 delay(1000);  
}
```

Listing 15: [TMP36ADC1115SingleEnded.ino](#)

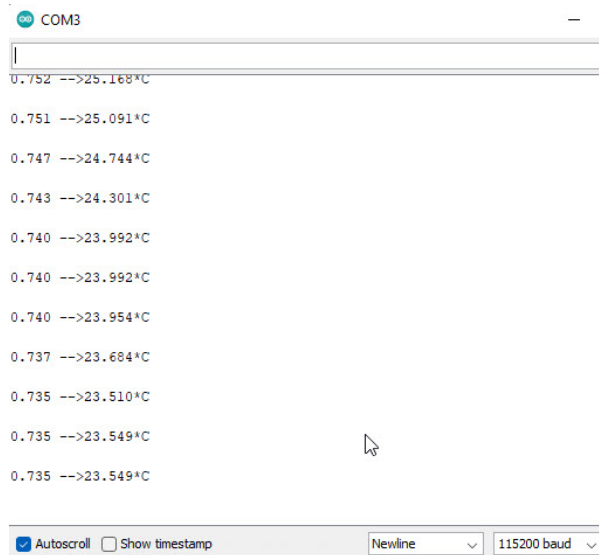


Abbildung 38: Temperaturausgabe TMP36 im Serial Monitor

Mit dem folgenden Sketch wird die 16Bit-Auflösung des ADS1115 auf den 1V Spannungsbereich des TMP36 angewendet. 1 Bit entspricht einer Auflösung von 0.03125mV. Mit vorhergehendem Beispiel war die Auflösung von 1 Bit 0.1875mV. Die Auflösung ist also 6x grösser. Siehe Zeile 42 im Source-Code.

Source-Code [TMP36ADC1115SingleEnded_with_GAIN4x.ino](#):

https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/TMP36ADC1115SingleEnded_with_GAIN4x/TMP36ADC1115SingleEnded_with_GAIN4x.ino

```
/*  
2 Title:      read analog Signal from TMP36 with ADS1115 - Analog Digital Converter  
4 Sketchfile: TMP36ADC1115SingleEnded_with_GAIN4x.ino  
6 Libraries used: Wire for I2C, Adafruit_ADS1X15 for ADS1115  
8 Function:   Read analog Signal from TMP36 and convert it into Voltage, using  
              ADS1115.  
              Temperature is derived from Voltage with Formula: temp = (voltage -  
10              0.5) * 100 ;  
12 Connections: TMP36 Vout is connected to A0 of ADS1115  
              SDA ADS1115 to SDA of ESP32  
              SCL ADS1115 to SCL of ESP32  
14  
16 Date:      tja/04.12.2021  
18 */  
20  
22 #include <Wire.h>  
#include <Adafruit_ADS1X15.h>
```



```
24 Adafruit_ADS1115 ads1115;
26
28 float volts0=0.000;
   float volts1=0.000;
   float temp=0.000;
30
32 void setup(void)
33 {
34   Serial.begin(115200);
   Serial.println("Hello!");
36
   Serial.println("Getting single-ended readings from AIN0..3");
38   Serial.println("ADC Range: +/- 6.144V (1 bit = 3mV)");
   // Set GAIN:
40   // TMP36 Input is about 1V. With GAIN_FOUR we push this range over 16Bits and get
   // much precise measurements than with
   // 16Bits over +/- 6.144 Volt (=default)
42   ads1115.setGain(GAIN_FOUR);           // 4x gain   +/- 1.024V  1 bit = 0.03125mV
                                           // push the 1V input signal over the entire 16-
                                           // bit range of the ADC
44   // other GAINS - For TMP36: GAIN_FOUR is perfect!
   // ads1115.setGain(GAIN_TWOTHIRDS); // 2/3x gain +/- 6.144V  1 bit = 0.1875mV (
   // default, no gain specified)
46   // ads1115.setGain(GAIN_ONE);         // 1x gain   +/- 4.096V  1 bit = 0.125mV
   // ads1115.setGain(GAIN_TWO);         // 2x gain   +/- 2.048V  1 bit = 0.0625mV
48   // ads1115.setGain(GAIN_FOUR);        // 4x gain   +/- 1.024V  1 bit = 0.03125mV
   // ads1115.setGain(GAIN_EIGHT);       // 8x gain   +/- 0.512V  1 bit = 0.015625mV
50   // ads1115.setGain(GAIN_SIXTEEN);     // 16x gain  +/- 0.256V  1 bit = 0.0078125mV

52   if (!ads1115.begin()) {
       Serial.println("Failed to initialize ADS1115.");
54       while (1);
   }
56 }

58 void loop(void)
59 {
60   int16_t adc0;

62   adc0 = ads1115.readADC_SingleEnded(0);
   //adc1 = ads1115.readADC_SingleEnded(1);
64   //adc2 = ads1115.readADC_SingleEnded(2);
   //adc3 = ads1115.readADC_SingleEnded(3);
66   volts0 = (ads1115.computeVolts(adc0))*1.0088888888888888; // Korrekturwert für
   Raumtemperatur. experimentell bestimmt.
                                           // Mit diesem
   Korrekturwert erhalten wir absolut
68   ~Millivoltgenauigkeit!
                                           // präzise Messungen.
                                           // Muss für jeden
   einzelnen ADC individuell bestimmt werden.
70   //volts1 = ads1115.computeVolts(adc1);
   //volts2 = ads1115.computeVolts(adc2);
72   //volts3 = ads1115.computeVolts(adc3);

74   temp = (volts0 - 0.5) * 100 ;

76   //Serial.print("AIN0: "); Serial.print(adc0);Serial.print(" ");Serial.println(
   volts0,3);
   //Serial.print("AIN1: "); Serial.print(adc1);Serial.print(" relates to ");
78   Serial.print(volts0,3);Serial.print(" -->");Serial.print(temp,3);Serial.println("
   *C");
```



```
80 //Serial.print("AIN2: "); Serial.println(adc2);  
//Serial.print("AIN3: "); Serial.println(adc3);  
Serial.println(" ");  
82 delay(1000);  
84 }
```

Listing 16: TMP36ADC1115SingleEnded_with_GAIN4x.ino

8.14 Digitaler Feuchtesensor DHT22-AM2302

Der DHT22-Sensor ist ein Feuchte- und Temperatursensor. Komponente Nr. 10, siehe Abbildung 4.

Das digitale Signal kann am Pin *DHT22 Dout* abgegriffen werden. Im Beispiel wird das Signal mit dem GPIO16 des ESP32 ausgewertet. Im Beispiel-Sketch wird die Library von *Adafruit* verwendet.

Source-Code [dht22.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/dht22/dht22.ino>

```
/*  
2 * Function: Auslesen der Feuchtigkeit und Temperatur mit dem Sensor DHT22  
* MCU: Olimex ESP32-DevKit-LiPo  
4 * File: dht22.ini  
* Library: DHT sensor library by Adafruit  
6 * Connections: DHT22 Dout mit GPIO 16 verbinden  
* Output: Serial Monitor with 115200baud  
8 * Author: Thomas Jäggi  
* Date: 30.07.2021  
10 */  
  
12 /***** ( Importieren der genutzten Bibliotheken )  
*****/  
#include "DHT.h"  
14 #define DHTPIN 16 // Hier die Pin Nummer eintragen wo der Sensor  
// angeschlossen ist  
#define DHTTYPE DHT22 // Hier wird definiert was für ein Sensor ausgelesen wird.  
// In  
16 // unserem Beispiel möchten wir einen DHT22 auslesen,  
// falls du  
// ein DHT22 hast einfach DHT11 eintragen  
18  
/***** ( Definieren der Objekte )  
*****/  
20 DHT dht(DHTPIN, DHTTYPE);  
  
22 void setup() {  
Serial.begin(115200);  
24 Serial.println("DHT22 Testprogramm");  
dht.begin();  
26 }  
void loop() {  
28 // Wait a few seconds between measurements.  
delay(2000); // Hier definieren wir die Verweilzeit die  
// gewartet wird  
30 // bis der Sensor wieder ausgelesen wird. Da der  
DHT22 // auch ca. 2 Sekunden hat um seine Werte zu  
aktualisieren  
32 // macht es keinen Sinn ihn schneller auszulesen  
!}
```



```
34 float h = dht.readHumidity();    // Lesen der Luftfeuchtigkeit und speichern in
    die Variable h
36 float t = dht.readTemperature(); // Lesen der Temperatur in C und speichern in
    die Variable t

38 /***** ( Ueberpruefen ob alles richtig Ausgelesen wurde )
    *****/
38 if (isnan(h) || isnan(t)) {
    Serial.println("Fehler beim auslesen des Sensors!");
40     return;
    }
42
    // Nun senden wir die gemessenen Werte an den PC diese werden wir im Seriellen
    Monitor sehen
44 Serial.print("Luftfeuchtigkeit: ");
    Serial.print(h);                // Ausgeben der Luftfeuchtigkeit
46 Serial.print("%t");              // Tabulator
    Serial.print("Temperatur: ");
48 Serial.print(t);                // Ausgeben der Temperatur
    Serial.write('*');              // Schreiben des Grad-Zeichen
50 Serial.println("C");
    }
```

Listing 17: [dht22.ino](#)

Die Ausgabe erfolgt hier im Beispiel mit dem Serial Monitor:

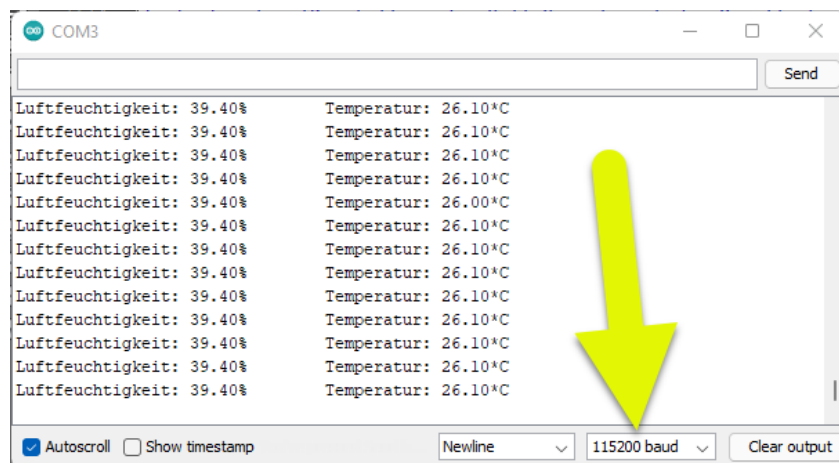


Abbildung 39: Ausgabe der Sensorwerte des Feuchtesensors DHT22/AM2302



8.14.1 Digitaler Feuchtesensor DHT22-AM2302 - Ausgabe auf OLED-Display

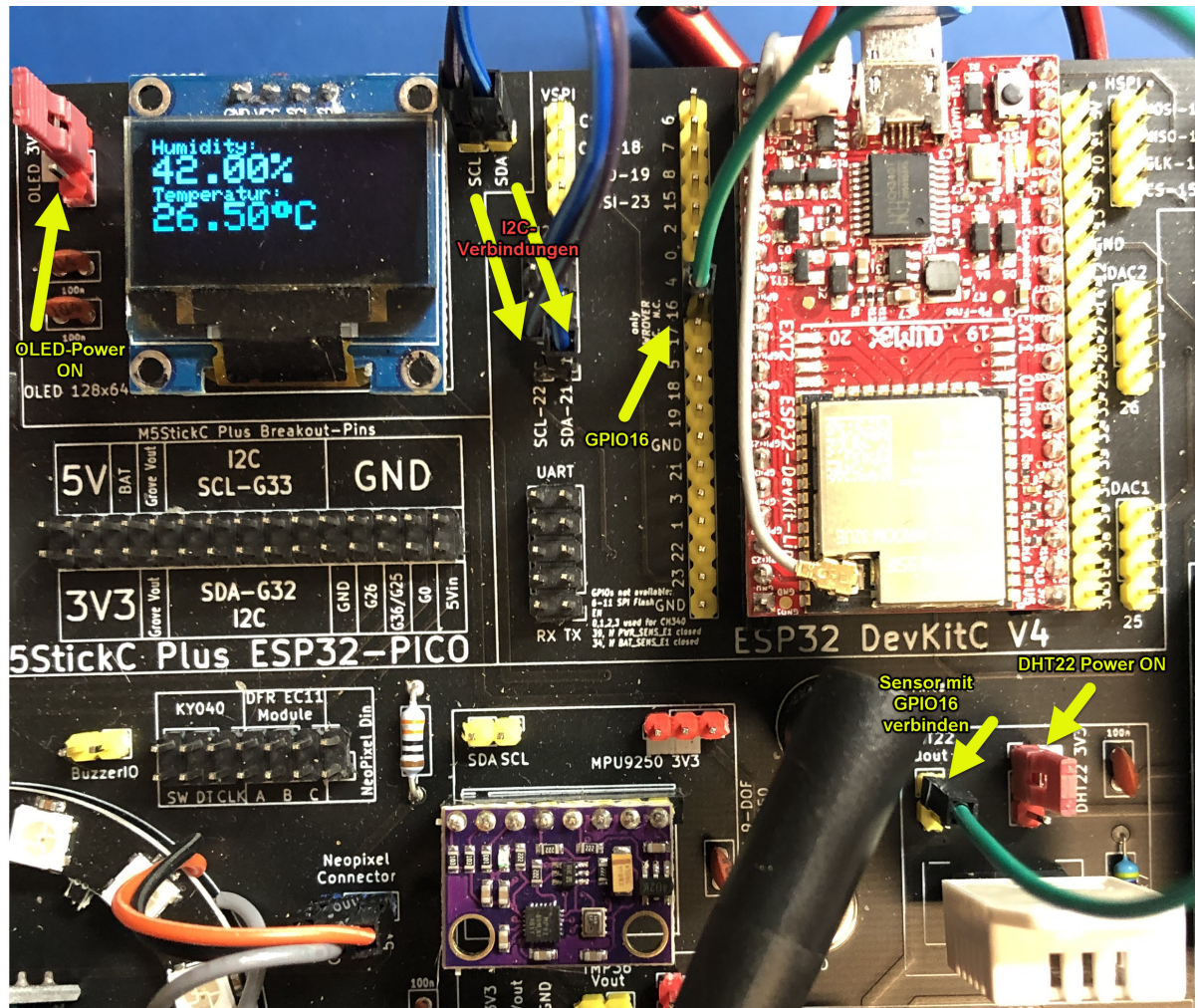


Abbildung 40: Ausgabe Sensorwerte auf OLED-Display
Source-Code [dht22OLED.ino](https://github.com/tjaeggli/iot-education-board/blob/master/ArduinoDemoSketches/dht22OLED/dht22OLED.ino):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/dht22OLED/dht22OLED.ino>

```
1 /*
2  * Function: Auslesen der Feuchtigkeit und Temperatur mit dem Sensor DHT22
3  * MCU:      Olimex ESP32-DevKit-LiPo
4  * File:     dht22OLED.ini
5  * Library:  DHT sensor, Adafruit GFX und Adafruit SSD106 Library wird verwendet.
6  * Connections: "DHT22 Dout" mit GPIO 16 verbinden. SCL und SDA from OLED to
7  *             Microcontroller.
8  *             Activate Power with Jumper "OLED 3V3" and "DHT22 3V3". Place Jumper
9  *             over white field.
10  * Output:   Serial Monitor with 115200baud and OLED-Display
11  * Author:   Thomas Jäggi
12  * Date:     30.07.2021
13  */
14
15 /***** ( Importieren der genutzten Bibliotheken )
16  *****/
17 #include "DHT.h"
18 #define DHTPIN 16 // Hier die Pin Nummer eintragen wo der Sensor
19 //                // angeschlossen ist
20 #define DHTTYPE DHT22 // Hier wird definiert was für ein Sensor ausgelesen wird.
21 //                // In
22 //                // unserem Beispiel möchten wir einen DHT22 auslesen,
```




```
falls du
// ein DHT22 hast einfach DHT11 eintragen
19 //***** OLED Settings *****
#include <Wire.h>
21 #include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
23
#define SCREEN_WIDTH 128 // OLED display width, in pixels
25 #define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define OLED_RESET 4 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64, 0x3C
for 128x32
29
//***** ( Definieren der Objekte )
*****/
31 DHT dht(DHTPIN, DHTTYPE);
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
33
void setup() {
35   Serial.begin(115200);
   Serial.println("DHT22 Testprogramm");
37   if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
       Serial.println(F("SSD1306 allocation failed"));
39       for(;;); // Don't proceed, loop forever
   }
41   display.clearDisplay();
   dht.begin();
43 }
void loop() {
45   // Wait a few seconds between measurements.
   delay(2000); // Hier definieren wir die Verweilzeit die
               gewartet wird
47               // bis der Sensor wieder ausgelesen wird. Da der
               DHT22
               // auch ca. 2 Sekunden hat um seine Werte zu
               aktualisieren
49               // macht es keinen Sinn ihn schneller auszulesen
               !

   float h = dht.readHumidity(); // Lesen der Luftfeuchtigkeit und speichern in
die Variable h
   float t = dht.readTemperature(); // Lesen der Temperatur in Grad-C und speichern
in die Variable t
53
//***** ( Ueberprüfen ob alles richtig Ausgelesen wurde )
*****/
55 if (isnan(h) || isnan(t)) {
   Serial.println("Fehler beim Auslesen des Sensors!");
57   display.clearDisplay();
   display.setTextSize(2); // Normal 1:1 pixel scale
59   display.setTextColor(SSD1306_WHITE); // Draw white text
   display.setCursor(0, 0); // Start at top-left corner
61   display.cp437(true); // Use full 256 char 'Code Page 437' font
   display.print("Fehler beim Auslesen des Sensors!");
63   display.display();
   return;
65 }

67 // Nun senden wir die gemessenen Werte an den PC diese werden wir im Seriellen
Monitor sehen
   display.clearDisplay();
69   display.setTextSize(1); // Normal 1:1 pixel scale
   display.setTextColor(SSD1306_WHITE); // Draw white text
```



```
71   display.setCursor(0, 0);           // Start at top-left corner
    //display.cp437(true);             // Use full 256 char 'Code Page 437' font
73   Serial.print("Luftfeuchtigkeit: ");
    Serial.print(h);                   // Ausgeben der Luftfeuchtigkeit
75   Serial.print("%\t");               // Tabulator
    Serial.print("Temperatur: ");
77   Serial.print(t);                   // Ausgeben der Temperatur
    Serial.print("*");                 // Schreiben des Grad Zeichen
79   Serial.println("C");
    // OLED Ausgabe:
81   display.println("Humidity:");
    display.setTextSize(2);
83   display.print(h); display.println("%"); // Ausgeben der Luftfeuchtigkeit
    display.setTextSize(1);
85   display.println("Temperatur:");
    display.setTextSize(2);
87   display.print(t);                   // Ausgeben der Temperatur
    display.print((char)247); // degree symbol
89   display.println("C");
    display.display();
91 }
```

Listing 18: [dht22OLED.ino](#)

8.15 Digitaler Temperatursensor DS18B20

Der DS18B20 ist ein digitaler Temperatursensor, welcher seine Daten mit dem 1-Wire Protokoll an den Microcontroller sendet. Der Sensor kann in zwei Versionen auf dem Board verbaut werden: Als Modul oder als nackter Sensor. Je nachdem, welche Version auf Ihrem Board verbaut ist, müssen Sie die Verbindung vom Pin *outnaked* oder *outModule* mit GPI15 verbinden. Entnehmen Sie den Bildern Ihre verbaute Version.

Source-Code [DS18B20.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/DS18B20/DS18B20.ino>

```
1  /*
   * Function: Auslesen der Temperatur mit dem Sensor DS18B20
3  * MCU:      Olimex ESP32-DevKit-LiPo
   * File:      DS18B20.ino
5  * Library:  OneWire und DallasTemperature by Miles Burton
   * Connections: "outnaked" oder "outModul" mit GPI04 verbinden, depending on
   *              Sensortype
7  *          Activate Power with Jumper "DS18B20 3V3". Place Jumper over white
   *          field.
   * Output:    Serial Monitor
9  * Author:    Thomas Jäggi
   * Date:      04.12.2021
11  */

13 #include <OneWire.h>
   #include <DallasTemperature.h>
15
   // GPIO where the DS18B20 is connected to
17 const int oneWireBus = 15;

19 // Setup a oneWire instance to communicate with any OneWire devices
   OneWire oneWire(oneWireBus);
21
   // Pass our oneWire reference to Dallas Temperature sensor
23 DallasTemperature sensors(&oneWire);
```

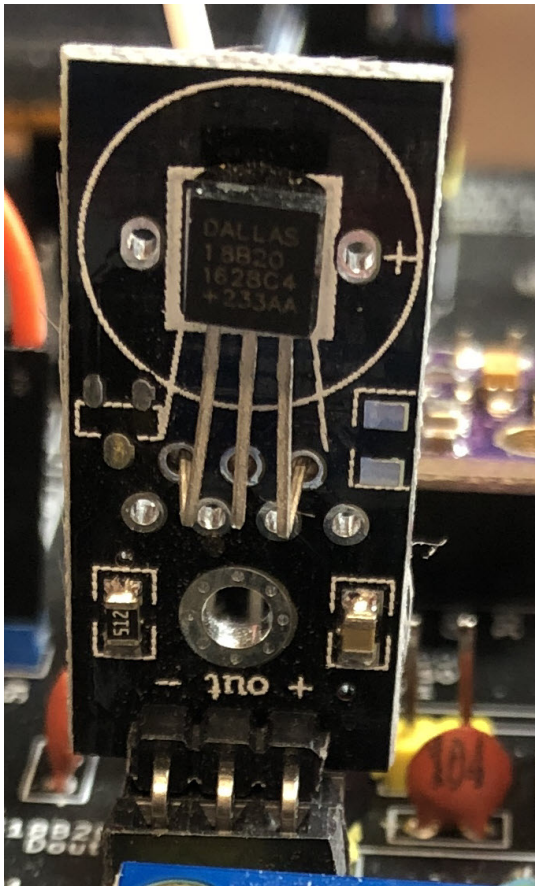


Abbildung 41: DS18B20 als Modul



Abbildung 42: nackter DS18B20-Sensor

```
25 void setup() {  
    // Start the Serial Monitor  
27    Serial.begin(115200);  
    // Start the DS18B20 sensor  
29    sensors.begin();  
    }  
31  
32 void loop() {  
33    sensors.requestTemperatures();  
    float temperatureC = sensors.getTempCByIndex(0);  
35    //float temperatureF = sensors.getTempFByIndex(0); // if you also want  
        Fahrenheit  
    Serial.print(temperatureC);  
37    Serial.println("*C");  
    //Serial.print(temperatureF); // if you also want Fahrenheit  
39    //Serial.println("*F"); // if you also want Fahrenheit  
    delay(1000);  
41 }
```

Listing 19: [DS18B20.ino](#)



The screenshot shows a serial monitor window titled "COM3". The data stream contains the following temperature readings in Celsius:

```

24.94°C
24.94°C
24.94°C
25.00°C
26.56°C
27.37°C
27.94°C
28.31°C

```

A large text overlay in the center of the window reads "Output im Serial Monitor". At the bottom of the window, there are control options: "Autoscroll" (checked), "Show timestamp" (unchecked), a "Newline" dropdown menu, and a baud rate setting of "115200 baud".

8.16 Distanzsensor HC-SR04

Distanzanzeige

OLED-Display

GPIO5

GPIO18

ECHO Pin auf GPIO18

Trig-Pin auf GPIO5

Objekt 10cm vor Sensor

Power-Jumper aktivieren (Jumper über das weisse Feld platzieren) für:

- ### Verbindungen mit Jumper-Kabel stecken:

- Bemerkung: Der HC-SR04 liefert am ECHO-Pin ein Signal mit einem Pegel von 5V. Der ESP32 kann maximal 3.3V verarbeiten. Aus diesem Grunde wird das Signal über den Level-Shifter (siehe Bauteil Nr. 30 in Abbildung 4) auf ein GPIO des ESP32 geführt.

<https://gitlab.com/tiaegqi/iot-education-board/-/blob/master/ArduinoDemoSketches/HC-SR04-OLED/HC-SR04-OLED.ino>



```
1  /*
   Title:          Read Distance with HC-SR04-Ultrasonic-Sensor
3
   Sketchfile:    HC-SR04-OLED.ino
5
   Libraries used: Wire, Adafruit SSD1306, Adafruit GFX Library
7
   Function:      The HC-SR04 ultrasonic sensor uses sonar to determine the distance to
                   an object.
9                   This sensor reads from 2cm to 400cm (0.8inch to 157inch) with an
                   accuracy of 0.3cm (0.1inches),
                   which is good for most hobbyist projects. In addition, this
                   particular module comes with ultrasonic transmitter
11                  and receiver modules.
                   The ultrasonic sensor uses sonar to determine the distance to an
                   object. Here is how it works:
13                  1. The ultrasound transmitter (trig pin) emits a high-frequency sound
                   (40 kHz).
                   2. The sound travels through the air. If it finds an object, it
                   bounces back to the module.
15                  3. The ultrasound receiver (echo pin) receives the reflected sound (
                   echo).
                   see also : https://randomnerdtutorials.com/esp32-hc-sr04-ultrasonic-arduino/
17
   Connections:  Activate Power on HC-SR04 5V,
19                  HV 5V,
                  LV 3V3
21                  OLED 3V3
                  Jumper Cable Connections:
23                  OLED SCL to ESP32 SCL
                  OLED SDA to ESP32 SDA
25                  ECHO LV1 to GPIO18
                  Trig to GPIO5
27
   Date:         tja/05.12.2021
29 */

31 #include <Wire.h>
   #include <Adafruit_GFX.h>
33 #include <Adafruit_SSD1306.h>

35 #define SCREEN_WIDTH 128 // OLED display width, in pixels
   #define SCREEN_HEIGHT 64 // OLED display height, in pixels
37
   // Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
39 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

41 const int trigPin = 5;
   const int echoPin = 18;
43
   //define sound speed in cm/uS
45 #define SOUND_SPEED 0.034
   #define CM_TO_INCH 0.393701
47
   long duration;
49 int distanceCm;
   int distanceInch;
51
   void setup() {
53     Serial.begin(115200);
       pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
55     pinMode(echoPin, INPUT); // Sets the echoPin as an Input
```




```
57 if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {  
    Serial.println(F("SSD1306 allocation failed"));  
59     for(;;);  
    }  
61     delay(500);  
    display.clearDisplay();  
63  
    display.setTextSize(2);  
65    display.setTextColor(WHITE);  
    }  
67  
void loop() {  
69    // Clears the trigPin  
    digitalWrite(trigPin, LOW);  
71    delayMicroseconds(2);  
    // Sets the trigPin on HIGH state for 10 micro seconds  
73    digitalWrite(trigPin, HIGH);  
    delayMicroseconds(10);  
75    digitalWrite(trigPin, LOW);  
  
77    // Reads the echoPin, returns the sound wave travel time in microseconds  
    duration = pulseIn(echoPin, HIGH);  
79  
    // Calculate the distance  
81    distanceCm = duration * SOUND_SPEED/2;  
  
83    // Convert to inches  
    distanceInch = distanceCm * CM_TO_INCH;  
85  
    // Prints the distance in the Serial Monitor  
87    Serial.print("Distance (cm): ");  
    Serial.println(distanceCm);  
89    Serial.print("Distance (inch): ");  
    Serial.println(distanceInch);  
91  
    display.clearDisplay();  
93    display.setCursor(0, 0);  
    display.setTextSize(1);  
95    display.println("Move your hand in");  
    display.println("front of HC-SR04!");  
97    display.setCursor(0, 35);  
    display.setTextSize(2);  
99    //Display distance in cm  
    display.print(distanceCm);  
101    display.println(" cm");  
    //display.print(distanceInch);  
103    //display.println(" inch");  
  
105    // Display distance in inches  
    /* display.print(distanceInch);  
107    display.print(" in");*/  
    display.display();  
109  
    delay(500);  
111 }
```

Listing 20: HC-SR04-OLED.ino

8.17 Bewegungssensor HC-SR501

Dieser Demo-Sketch demonstriert das Detektieren einer Bewegung mit dem Bewegungssensor HC-SR501. Wenn eine Bewegung vom Sensor detektiert wird, wird ein Interrupt



ausgelöst. In diesem Moment startet ein Timer und schaltet eine LED ein, welche für eine bestimmte Dauer leuchtet. Das Ereignis wird auch im Serial Monitor ausgegeben. Wenn der Timer abgelaufen ist, wird das LED automatisch ausgeschaltet. Dies ist eine typische Situation einer Lichtsteuerung in einem Hauseingang.

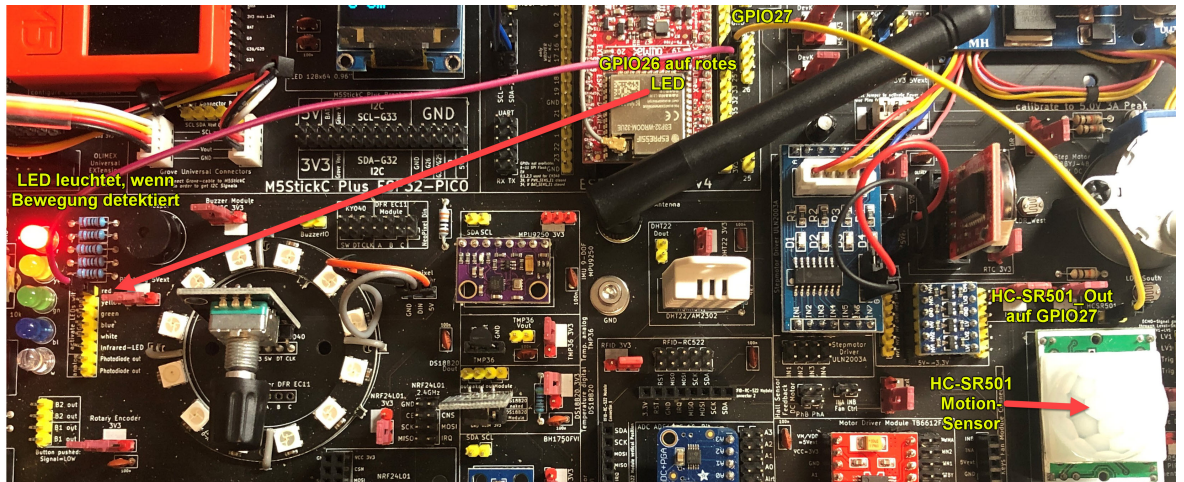


Abbildung 45: Bewegungsdetektor mit HC-SR501

Folgende Vorbereitungen sind notwendig:

Power-Jumper aktivieren (Jumper über das weiße Feld platzieren) für:

- › HC-SR501 5V

Verbindungen mit Jumper-Kabel stecken:

- › HC-SR501_Out mit GPIO27
- › LED red mit GPIO26

Source-Code [HC-SR501-LED.ino](https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/HC-SR501-LED/HC-SR501-LED.ino):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/HC-SR501-LED/HC-SR501-LED.ino>

```
/*
2  Title:      Detect Motion with HC-SR501
4  Sketchfile: HC-SR501-LED.ino
6  Libraries used: none
8  Function:   This tutorial shows how to detect motion with the ESP32 using a PIR
               motion sensor.
               In this example, when motion is detected (an interrupt is triggered),
               the ESP32 starts
10             a timer and turns an LED on for a predefined number of seconds. When
               the timer finishes
12             counting down, the LED is automatically turned off.
14  Connections: Activate Power on HC-SR501 5V PIR
               Jumper Cable Connections:
16                     HC-SR501_Out mit GPIO27
                     LED red mit      GPIO26
18
20  siehe auch: https://randomnerdtutorials.com/esp32-pir-motion-sensor-interrupts-
               timers/
```



```

22  Date:          tja/05.12.2021
23  */
24  #define timeSeconds 10
25
26  // Set GPIOs for LED and PIR Motion Sensor
27  const int led = 26;
28  const int motionSensor = 27;
29
30  // Timer: Auxiliary variables
31  unsigned long now = millis();
32  unsigned long lastTrigger = 0;
33  boolean startTimer = false;
34
35  // Checks if motion was detected, sets LED HIGH and starts a timer
36  void IRAM_ATTR detectsMovement() {
37      Serial.println("MOTION DETECTED!!!");
38      digitalWrite(led, HIGH);
39      startTimer = true;
40      lastTrigger = millis();
41  }
42
43  void setup() {
44      // Serial port for debugging purposes
45      Serial.begin(115200);
46
47      // PIR Motion Sensor mode INPUT_PULLUP
48      pinMode(motionSensor, INPUT_PULLUP);
49      // Set motionSensor pin as interrupt, assign interrupt function and set RISING
50      // mode
51      attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);
52
53      // Set LED to LOW
54      pinMode(led, OUTPUT);
55      digitalWrite(led, LOW);
56  }
57
58  void loop() {
59      // Current time
60      now = millis();
61      // Turn off the LED after the number of seconds defined in the timeSeconds
62      // variable
63      if(startTimer && (now - lastTrigger > (timeSeconds*1000))) {
64          Serial.println("Motion stopped...");
65          digitalWrite(led, LOW);
66          startTimer = false;
67      }
68  }
```

Listing 21: [HC-SR501-LED.ino](#)

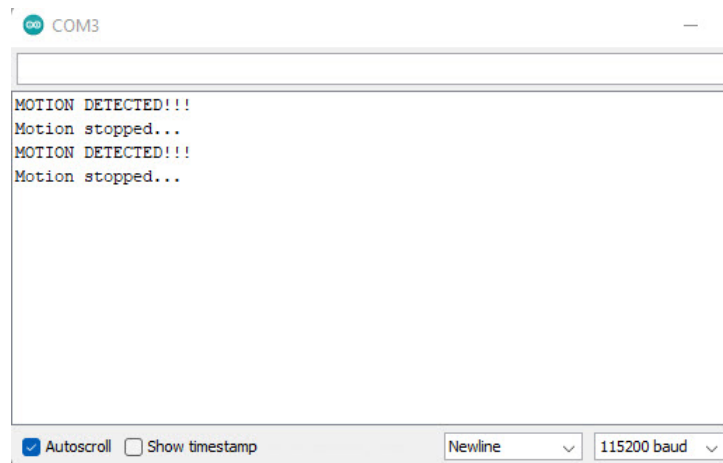


Abbildung 46: Das Detektieren der Bewegung und das Ausschalten der LED wird auch im Serial Monitor ausgegeben

8.18 LDR - Fotowiderstand

Ein Fotowiderstand ist ein Halbleiter, dessen Widerstandswert lichtabhängig ist. Er wird auch *LDR* (Light Dependent Resistor) genannt.

Alle Halbleitermaterialien sind lichtempfindlich und würden sich deshalb gut für eine Fotowiderstand eignen. Da dieser Effekt nicht in jedem Halbleiter gleich stark in Erscheinung tritt, gibt es spezielle Halbleitermischungen, bei denen dieser Effekt besonders stark auftritt.

Neben Cadmiumsulfid (CdS) gibt es für Fotowiderstände auch Bleisulfid (PbS), Bleisele-
nid (PbSe), Indiumarsenid (InAs), Germanium (Ge) oder Silizium (Si). Diese Halbleitermi-
schungen haben einen besonders starken inneren fotoelektrischen Effekt. Je nach elek-
trischen Eigenschaften und Hersteller gibt es noch viele weitere Halbleitermischungen.

Auf dem Board sind 4 LDRs vorhanden. Jeder LDR steht für eine Himmelsrichtung: LDR_North, LDR_South, LDR_West und LDR_East. Damit können wir den Lichteinfall aus einer Himmelsrichtung simulieren, indem wir z. Bsp. die Taschenlampe unseres Smart-
phones darauf halten oder das LDR mit dem Finger gleich abdecken.

Die LDRs sind jeweils Bestandteil eines Spannungsteilers. Der Spannungsteiler ist auf dem Board derart aufgebaut, dass eine hohe Spannung eine helle Umgebung und eine tiefe Spannung eine dunkle Umgebung simuliert.

Die Spannung der 4 LDRs, bzw. Spannungsteiler, kann an den Pins *LDR_North*, *LDR_South*, *LDR_West* und *LDR_East* abgegriffen werden.

Die Spannungsauswertung werden wir mit dem Analog-Digital-Converter ADS1115 vor-
nehmen.

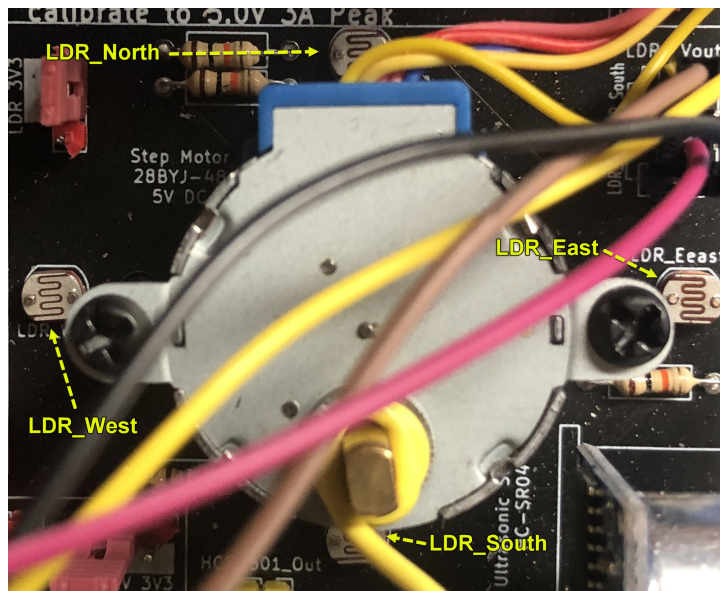


Abbildung 47: Positionen der 4 LDRs

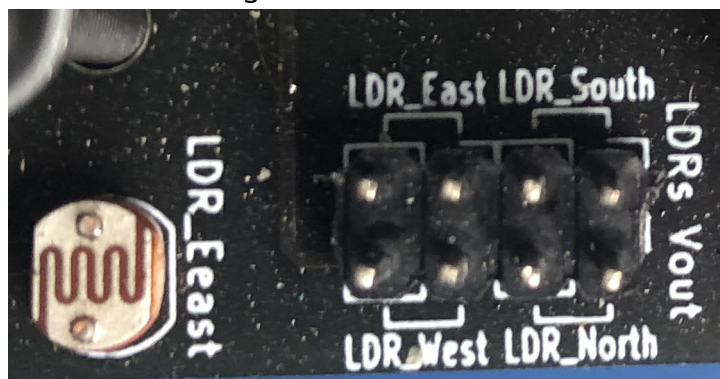


Abbildung 48: Spannungsabgriff *LDRs Vout* für die vier LDR_North, -South, -West und -East

Folgende Vorbereitungen sind notwendig:

Power-Jumper aktivieren (Jumper über das weisse Feld platzieren) für:

- › LDR 3V3
- › ADC 3V3

Verbindungen mit Jumper-Kabel stecken:

- › ADS1115 SDA mit ESP32 SDA
- › ADS1115 SCL mit ESP32 SCL
- › LDR_North mit ADS1115 A0
- › LDR_South mit ADS1115 A1
- › LDR_West mit ADS1115 A2
- › LDR_East mit ADS1115 A3

Source-Code [LDRADC1115SingleEnded.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/LDRADC1115SingleEnded/LDRADC1115SingleEnded.ino>

```
1  /*  
   Title:      read analog Signal from LDR_N, LDR_S, LDR_W, LDR_E with ADS1115 -  
               Analog Digital Converter  
3
```




```
Sketchfile:  LDRADC1115SingleEnded.ino
5
Libraries used: Wire for I2C, Adafruit_ADS1X15 for ADS1115
7
Function:      Read analog Signal from 4 LDRs and convert it into Voltage, using
                ADS1115.
9
11 Connections: LDR_North connected to A0 of ADS1115
                LDR_South connected to A1 of ADS1115
13                LDR_West connected to A2 of ADS1115
                LDR_East connected to A3 of ADS1115
15
Date:          tja/05.12.2021
17
*/
19
#include <Adafruit_ADS1X15.h>
21
Adafruit_ADS1115 ads; /* Use this for the 16-bit version */
23 //Adafruit_ADS1015 ads; /* Use this for the 12-bit version */

25 void setup(void)
{
27   Serial.begin(115200);
   Serial.println("Hello!");
29
   Serial.println("Getting single-ended readings from AIN0..3");
31   Serial.println("ADC Range: +/- 6.144V (1 bit = 3mV/ADS1015, 0.1875mV/ADS1115)");

33   // The ADC input range (or gain) can be changed via the following
   // functions, but be careful never to exceed VDD +0.3V max, or to
35   // exceed the upper and lower limits if you adjust the input range!
   // Setting these values incorrectly may destroy your ADC!
37   //
   //                                     ADS1015
   //   ADS1115
   //   -----
   //   -----
39   // ads.setGain(GAIN_TWOTHIRDS); // 2/3x gain +/- 6.144V  1 bit = 3mV      0.1875
   //   mV (default)
   // ads.setGain(GAIN_ONE);        // 1x gain   +/- 4.096V  1 bit = 2mV      0.125
   //   mV
41   // ads.setGain(GAIN_TWO);        // 2x gain   +/- 2.048V  1 bit = 1mV      0.0625
   //   mV
   // ads.setGain(GAIN_FOUR);        // 4x gain   +/- 1.024V  1 bit = 0.5mV
   //   0.03125mV
43   // ads.setGain(GAIN_EIGHT);      // 8x gain   +/- 0.512V  1 bit = 0.25mV
   //   0.015625mV
   // ads.setGain(GAIN_SIXTEEN);     // 16x gain  +/- 0.256V  1 bit = 0.125mV
   //   0.0078125mV
45
   if (!ads.begin()) {
47     Serial.println("Failed to initialize ADS.");
     while (1);
49   }
}

51
void loop(void)
53 {
   int16_t adc0, adc1, adc2, adc3;
55   float voltsN, voltsS, voltsW, voltsE;

57   adc0 = ads.readADC_SingleEnded(0);
   adc1 = ads.readADC_SingleEnded(1);
```



```
59 adc2 = ads.readADC_SingleEnded(2);  
   adc3 = ads.readADC_SingleEnded(3);  
61  
   voltsN = ads.computeVolts(adc0);  
63   voltsS = ads.computeVolts(adc1);  
   voltsW = ads.computeVolts(adc2);  
65   voltsE = ads.computeVolts(adc3);  
67   Serial.println("----Je höher die Spannung - je heller ist die Umgebung des LDR  
   -----");  
   Serial.print("Nord: "); Serial.print(adc0); Serial.print(" "); Serial.print(  
   voltsN,2); Serial.println("V");  
69   Serial.print("Süd: "); Serial.print(adc1); Serial.print(" "); Serial.print(  
   voltsS,2); Serial.println("V");  
   Serial.print("West: "); Serial.print(adc2); Serial.print(" "); Serial.print(  
   voltsW,2); Serial.println("V");  
71   Serial.print("Ost: "); Serial.print(adc3); Serial.print(" "); Serial.print(  
   voltsE,2); Serial.println("V");  
73   delay(1000);  
}
```

Listing 22: LDRADC1115SingleEnded.ino

Als Resultat erhalten wir im Serial Monitor die Spannungen der 4 LDRs.

Es gilt:

Je höher die Spannung, desto heller ist die
Umgebung; je tiefer die Spannung, desto dunkler
ist die Umgebung.

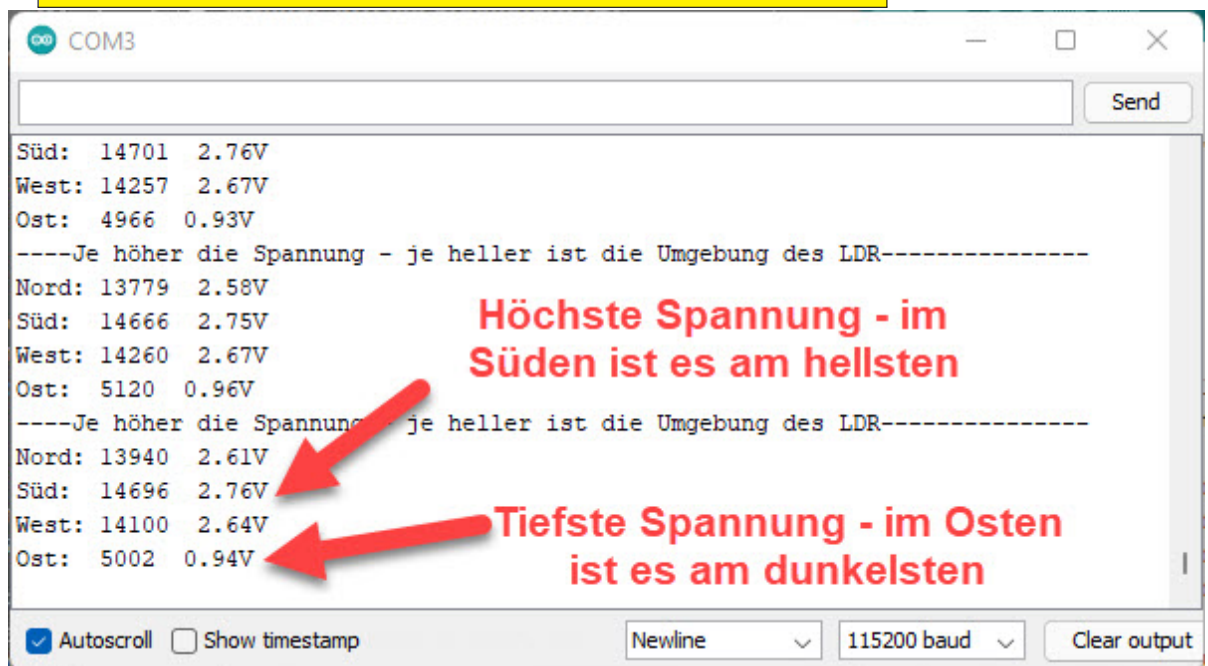


Abbildung 49: Ausgabe der LDR-Spannungswerte im Serial Monitor

Um eine Solarpanel-Anlage zu simulieren, die sich am Lichteinfall ausrichtet, müssen wir nur diesen Sketch mit dem Sketch von Listing 14 kombinieren.



8.19 Luftdrucksensor BMP280

Der BMP280 ist ein Barometer, mit dem der atmosphärische Druck gemessen werden kann. Zusätzlich kann der Sensor auch als Temperatursensor eingesetzt werden. Die Genauigkeit des Sensors beträgt ca. 1 hPa. Am gleichen Platz wie der BMP280 kann auch der BMP388 verwendet werden. Dieses neue Modell (2021) von Bosch hat eine Genauigkeit von ca. 1Pa. Standardmässig ist aber der BMP280 verbaut: Nr 16 in Abbildung 4.

Der Demo-Sketch gibt die Sensorwerte im Serial Monitor und auf dem OLED aus. Neben den üblichen SSD1306-Libraries von Adafruit für das OLED Display wurde für den BMP280 die Library *BMx280MI* verwendet.

Der BMP280 und auch das OLED-Display kommunizieren mit dem Microcontroller über I2C. Es müssen also nur die Pins SCL und SDA mit dem Microcontroller verbunden werden. Vergessen Sie nicht, die Komponenten mit dem roten Jumper mit Spannung zu versorgen. Jumperpositionen über weissem Feld bei *OLED 3V3* und *BMP280/388 3V3*.

Source-Code [BMP280withOLED.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/BMP280withOLED/BMP280withOLED.ino>

```
2  /*
   * Function:  Read Pressure and Temperatur from BMP280
   * MCU:      Olimex ESP32-DevKit-LiPo
   * File:      BMP280withOLED.ino
   * Libraries: BMx280MI, Adafruit GFX und Adafruit SSD106 Library wird verwendet.
   * Connections: SCL and SDA from BMP280 to SCL und SDA from ESP32. SCL und SDA from
   *              OLED to Microcontroller.
   *              Activate Power with Jumper "OLED 3V3" and "BMP280/388 3V3". Place
   *              Jumper over white field.
   * Output:    Serial Monitor with 115200baud and OLED-Display
   * Author:    Thomas Jäggi
   * Date:      07.12.2021
   * Modified from Example-Sketch
   */
14 #include <BMx280I2C.h>
   #include <Wire.h>
16 //*****OLED Settings
   *****
   #include <Adafruit_GFX.h>
18 #include <Adafruit_SSD1306.h>

20 #define SCREEN_WIDTH 128 // OLED display width, in pixels
   #define SCREEN_HEIGHT 64 // OLED display height, in pixels
22
   #define OLED_RESET      -1 // Reset pin # (or -1 if sharing Arduino reset pin)
24 #define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3C for 128x64
   //
   *****

26 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

28 #define I2C_ADDRESS_BMP280 0x76

30 //create a BMx280I2C object using the I2C interface with I2C Address 0x76
   BMx280I2C bmx280(I2C_ADDRESS_BMP280);
32
   void setup() {
34   // put your setup code here, to run once:
   Serial.begin(115200);
```



```
36  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
37      Serial.println(F("SSD1306 allocation failed"));
38      for(;;); // Don't proceed, loop forever
39  }
40  display.clearDisplay();
41
42  Wire.begin();
43
44  //begin() checks the Interface, reads the sensor ID (to differentiate between
    BMP280 and BME280)
    //and reads compensation parameters.
46  if (!bmx280.begin())
47  {
48      Serial.println("begin() failed. check your BMx280 Interface and I2C Address.");
49      while (1);
50  }
51
52  if (bmx280.isBME280())
53      Serial.println("sensor is a BME280");
54  else
55      Serial.println("sensor is a BMP280");
56
57  //reset sensor to default parameters.
58  bmx280.resetToDefaults();
59
60  //by default sensing is disabled and must be enabled by setting a non-zero
    //oversampling setting.
62  //set an oversampling setting for pressure and temperature measurements.
    bmx280.writeOversamplingPressure(BMx280MI::OSRS_P_x16);
64  bmx280.writeOversamplingTemperature(BMx280MI::OSRS_T_x16);
65
66  //if sensor is a BME280, set an oversampling setting for humidity measurements.
    if (bmx280.isBME280())
68      bmx280.writeOversamplingHumidity(BMx280MI::OSRS_H_x16);
69  }
70
71  void loop() {
72      // put your main code here, to run repeatedly:
73
74      delay(1000);
75
76      //start a measurement
77      if (!bmx280.measure())
78      {
79          Serial.println("could not start measurement, is a measurement already running?"
80          );
81          return;
82      }
83
84      //wait for the measurement to finish
85      do
86      {
87          delay(100);
88      } while (!bmx280.hasValue());
89
90  // Serial Monitor Ausgabe
    Serial.print("Pressure: "); Serial.print(bmx280.getPressure()/100); Serial.println
    (" hPa");
91  Serial.print("Pressure (64 bit): "); Serial.print(bmx280.getPressure64()/100);
    Serial.println(" hPa");
92  Serial.print("Temperature: "); Serial.print(bmx280.getTemperature()); Serial.
    println(" *C");
93  Serial.println(" ");
94  // OLED Ausgabe:
    display.clearDisplay();
```



```
96 display.setTextSize(1);          // Normal 1:1 pixel scale
display.setTextColor(SSD1306_WHITE); // Draw white text
display.setCursor(0, 0);           // Start at top-left corner
98 display.setTextSize(2);
display.println("Luftdruck:");
100 display.print(bmx280.getPressure()/100); display.println(" hPa"); // Ausgeben
    hPa
//display.println("Pressure 64Bit:");
102 //display.print(bmx280.getPressure64()/100);display.println("hPa"); // Ausgeben
    hPa 64 Bit
display.println("Temp:");
104 display.print(bmx280.getTemperature());display.print((char)247); display.println(
    "C");
display.display();
106 //important: measurement data is read from the sensor in function hasValue() only
    .
//make sure to call get*() functions only after hasValue() has returned true.
108 if (bmx280.isBME280())
{
110     Serial.print("Humidity: ");
    Serial.println(bmx280.getHumidity());
112 }
}
```

Listing 23: BMP280withOLED.ino

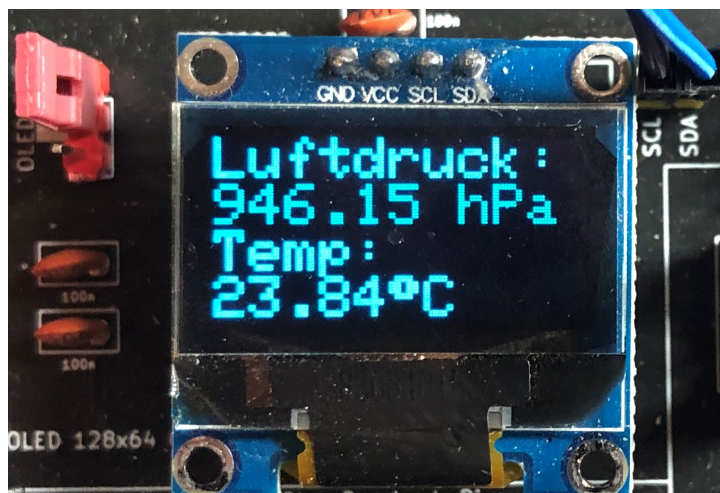


Abbildung 50: Ausgabe der Sensorwerte auf dem OLED-Display

8.20 Luftdrucksensor BMP388

Der BMP388 ist auf dem Board nicht standardmässig verbaut. Ein Stecksocket ist jedoch vorhanden, so dass man den BMP280 damit ersetzen kann.

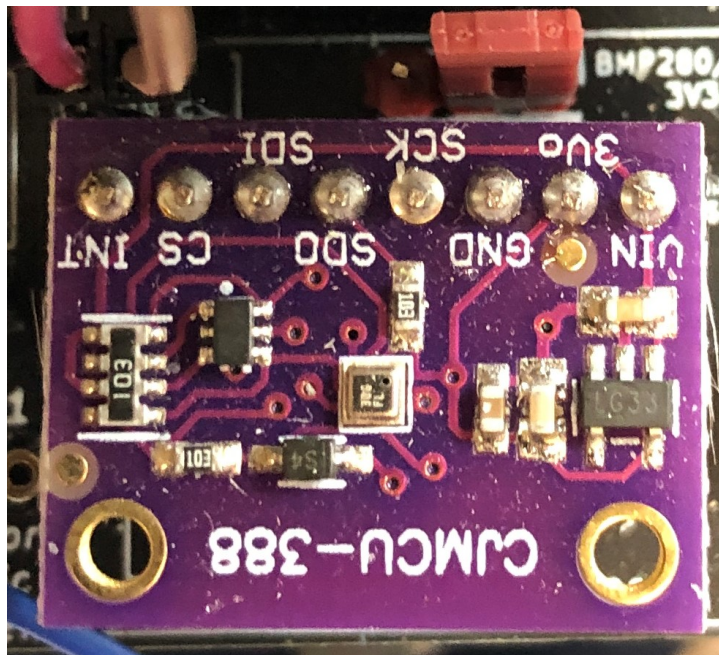


Abbildung 51: Der "neue" BMP388 mit 1Pa Genauigkeit
Source-Code [BMP388withOLED.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/BMP388withOLED/BMP388withOLED.ino>

```
1  /*  
2   * Function:  Read Pressure and Temperatur from BMP388  
3   * MCU:      Olimex ESP32-DevKit-LiPo  
4   * File:      BMP388withOLED.ino  
5   * Library:   Adafruit_BMP3XX, Adafruit GFX, Adafruit SSD106, Adafruit Unified  
6   *             Sensor Library wird verwendet.  
7   * Connections: SCL and SDA from BMP388 to SCL und SDA from ESP32. SCL und SDA from  
8   *             OLED to Microcontroller.  
9   *             Activate Power with Jumper "OLED 3V3" and "BMP280/388 3V3". Place  
10  *            Jumper over white field.  
11  * Output:    Serial Monitor with 115200baud and OLED-Display  
12  * Author:    Thomas Jäggi  
13  * Date:      07.12.2021  
14  * Modified from Example-Sketch. See Complete project details: https://  
15  *            RandomNerdTutorials.com/esp32-bmp388-arduino/  
16  */  
17  
18 #include <Wire.h>  
19 #include <Adafruit_Sensor.h>  
20 #include "Adafruit_BMP3XX.h"  
21  
22 #define SEALEVELPRESSURE_HPA (1008) // Wert nachschauen für aktuellen Standort:  
23 // https://meteologix.com/observations/pressure-qnh.html  
24 //*****OLED Settings  
25 //*****  
26 #include <Adafruit_GFX.h>  
27 #include <Adafruit_SSD1306.h>  
28  
29 #define SCREEN_WIDTH 128 // OLED display width, in pixels  
30 #define SCREEN_HEIGHT 64 // OLED display height, in pixels  
31  
32 #define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)  
33 #define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3C for 128x64  
34 //  
35 *****
```



```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
31
Adafruit_BMP3XX bmp;
33
void setup() {
35   Serial.begin(115200);
       if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
37       Serial.println(F("SSD1306 allocation failed"));
       for(;;); // Don't proceed, loop forever
39   }
   display.clearDisplay();
41   //while (!Serial);
   //Serial.println("Adafruit BMP388 / BMP390 test");
43
   if (!bmp.begin_I2C()) { // hardware I2C mode, can pass in address & alt Wire
45   //if (! bmp.begin_SPI(BMP_CS)) { // hardware SPI mode
   //if (! bmp.begin_SPI(BMP_CS, BMP_SCK, BMP_MISO, BMP_MOSI)) { // software SPI
       mode
47       Serial.println("Could not find a valid BMP3 sensor, check wiring!");
       while (1);
49   }

51   // Set up oversampling and filter initialization
   bmp.setTemperatureOversampling(BMP3_OVERSAMPLING_8X);
53   bmp.setPressureOversampling(BMP3_OVERSAMPLING_4X);
   bmp.setIIRFilterCoeff(BMP3_IIR_FILTER_COEFF_3);
55   bmp.setOutputDataRate(BMP3_ODR_50_HZ);
   }
57
void loop() {
59   if (! bmp.performReading()) {
       Serial.println("Failed to perform reading :(");
61       return;
   }
63   // Serial Monitor Ausgabe
   Serial.print("Temperature = ");
65   Serial.print(bmp.temperature);
   Serial.println(" *C");
67

   Serial.print("Pressure = ");
69   Serial.print(bmp.pressure / 100.0);
   Serial.println(" hPa");
71

   Serial.print("Approx. Altitude = ");
73   Serial.print(bmp.readAltitude(SEALEVELPRESSURE_HPA));
   Serial.println(" m");
75

   Serial.println();
77   // OLED Ausgabe:
   display.clearDisplay();
79   display.setTextSize(1); // Normal 1:1 pixel scale
   display.setTextColor(SSD1306_WHITE); // Draw white text
81   display.setCursor(0, 0); // Start at top-left corner
   display.setTextSize(1);
83   display.println("Luftdruck:");
   display.print(bmp.pressure / 100.0); display.println(" hPa"); // Ausgeben hPa
85   display.println("");
   display.println("Hoehe ueber Meer:");
87   display.print(bmp.readAltitude(SEALEVELPRESSURE_HPA));display.println(" m"); //
       Ausgeben H m
   display.println("");
89   display.println("Temperatur:");
   display.print(bmp.temperature);display.print((char)247); display.println("C");
91   display.display();
```



```
93 } delay(2000);
```

Listing 24: [BMP388withOLED.ino](#)



Abbildung 52: Ausgabe der Sensorwerte des BMP388 auf dem OLED-Display. Hier ebenfalls mit der berechneten Höhe über Meer.

Die folgenden Bilder geben einen Eindruck der Präzision des Sensors: Der barometrische Druck wurde auf verschiedenen Etagen eines Hauses gemessen und daraus die Höhe ü. M. berechnet.

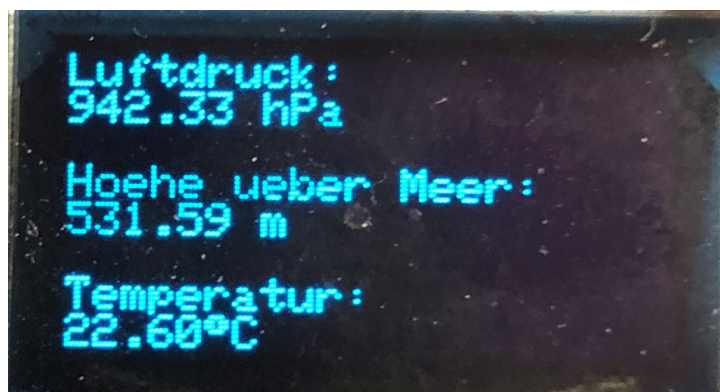


Abbildung 53: Höhenangabe in Etage -1

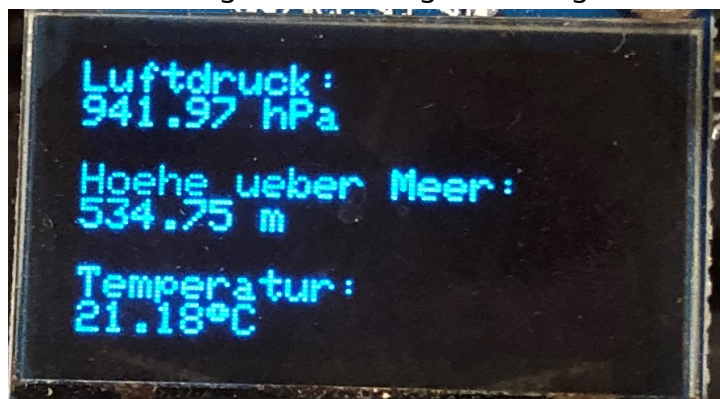


Abbildung 54: Höhenangabe in Etage 0

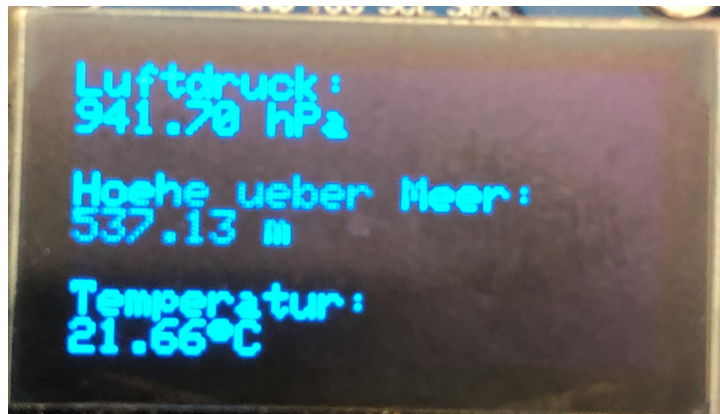


Abbildung 55: Höhenangabe in Etage 1

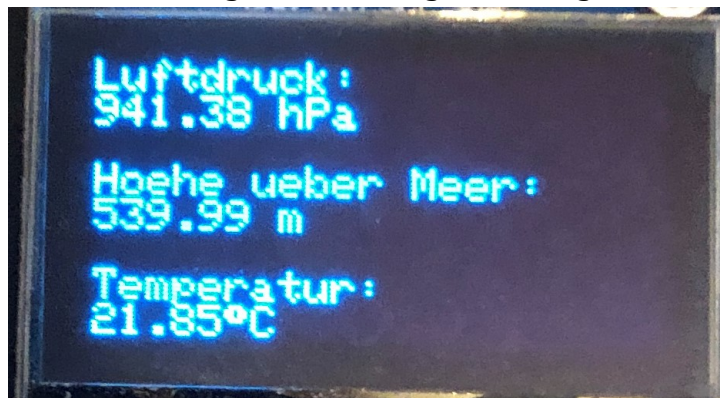


Abbildung 56: Höhenangabe in Etage 2

8.21 Luftdrucksensor BME680

Der Luftdrucksensor BME680 ist nicht standardmässig verbaut. Der Sockel passt jedoch, so dass auch dieses Modell eingesetzt werden kann. Eine entsprechende Library gibt es auch:



Abbildung 57: Library für BME680

8.22 RTC DS3231 - Echtzeituhr

Für einige Anwendungen mit einem Microcontroller benötigt man die aktuelle Uhrzeit. Sei es, um eine ganz einfache Uhr zusammen mit einem Display zu bauen, oder um Messwerte mit dem passenden Timestamp zu versehen. Der Fehler beträgt mit dieser Korrektur lediglich ± 2 ppm. Dies entspricht einer Abweichung von ca. 1 Minute pro Jahr.

Auf dem RTC-Modul ist ein Temperatursensor integriert. Dieser Sensor korrigiert die Abweichungen des Quarzes, welche durch Temperatureinflüsse entstehen.

Die für die Temperaturkompensation des Quarzes gemessene Temperatur, kann auch aus den Registern des DS3231 ausgelesen werden, wobei die mögliche Abweichung laut Datenblatt mit $\pm 3^\circ\text{C}$ jedoch sehr hoch sein kann.

Damit die RTC auch ohne Stromversorgung weiterläuft, wird sie über eine Batterie mit Strom versorgt. Auf einer Seite des Moduls ist ein Batteriehalter für die Knopfzelle CR2032 vorgesehen.



Dieser Demo-Sketch aktiviert die RTC3231.

Verwendete Library: **RTCLib** (by Adafruit)

Im Code kann das aktuelle Datum und die Uhrzeit einmal eingestellt werden. Anschliessend läuft die Uhr, solange die Batterie das Modul mit Strom versorgt.

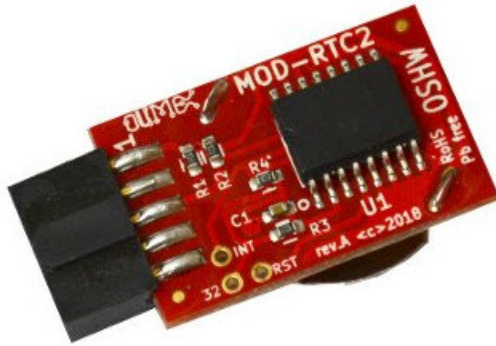


Abbildung 58: RTC 3231 Uhrenmodul - Frontseite

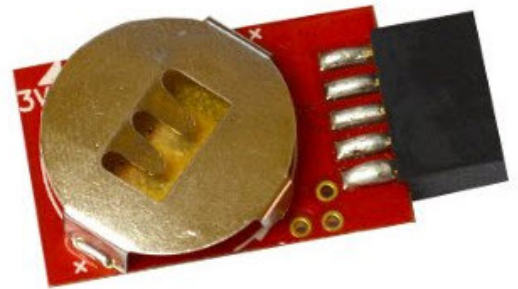


Abbildung 59: RTC 3231 Uhrenmodul - Rückseite

Source-Code [RTCds3231RTCLib.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/RTCds3231RTCLib/RTCds3231RTCLib.ino>

```
1  /*
   * Function:   Realtime-Clock RTC3231
3  * MCU:       Olimex ESP32-DevKit-LiPo
   * File:      RTCds3231RTCLib
5  * Library:   RTCLib (by Adafruit)
   * Connections: SCL and SDA from RTC-Module to SCL und SDA from ESP32.
7  *           Activate Power with Jumper "RTC 3V3"
   * Output:    Serial Monitor with 115200baud
9  * Author:    Thomas Jäggi
   * Date:      07.12.2021
11 * Modified from Example-Sketch from https://github.com/adafruit/RTCLib
   */
13
14 // Date and time functions using a DS3231 RTC connected via I2C and Wire lib
15 #include "RTCLib.h"
16
17 RTC_DS3231 rtc;
18
19 char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
   , "Friday", "Saturday"};
20
21 void setup () {
22     Serial.begin(115200);
23
24     #ifndef ESP8266
25         while (!Serial); // wait for serial port to connect. Needed for native USB
26     #endif
27
28     if (! rtc.begin()) {
29         Serial.println("Couldn't find RTC");
30         Serial.flush();
31         while (1) delay(10);
32     }
33 }
```




```
33
34
35 if (rtc.lostPower()) {
36     Serial.println("RTC lost power, let's set the time!");
37     // When time needs to be set on a new device, or after a power loss, the
38     // following line sets the RTC to the date & time this sketch was compiled
39     rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
40     // This line sets the RTC with an explicit date & time, for example to set
41     // January 21, 2014 at 3am you would call:
42     //rtc.adjust(DateTime(2021, 12, 12, 17, 45, 0));
43 }
44
45 // When time needs to be re-set on a previously configured device, the
46 // following line sets the RTC to the date & time this sketch was compiled
47 // rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
48 // This line sets the RTC with an explicit date & time, for example to set
49 // January 21, 2014 at 3am you would call:
50 // rtc.adjust(DateTime(2021, 12, 12, 17, 48, 0));
51 }
52
53 void loop () {
54     DateTime now = rtc.now();
55
56     Serial.print(now.year(), DEC);
57     Serial.print('/');
58     Serial.print(now.month(), DEC);
59     Serial.print('/');
60     Serial.print(now.day(), DEC);
61     Serial.print(" ");
62     Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
63     Serial.print(" ");
64     Serial.print(now.hour(), DEC);
65     Serial.print(':');
66     Serial.print(now.minute(), DEC);
67     Serial.print(':');
68     Serial.print(now.second(), DEC);
69     Serial.println();
70
71     Serial.print(" since midnight 1/1/1970 = ");
72     Serial.print(now.unixtime());
73     Serial.print("s = ");
74     Serial.print(now.unixtime() / 86400L);
75     Serial.println("d");
76
77     // calculate a date which is 7 days, 12 hours, 30 minutes, 6 seconds into the
78     // future
79     DateTime future (now + TimeSpan(7,12,30,6));
80
81     Serial.print(" now + 7d + 12h + 30m + 6s: ");
82     Serial.print(future.year(), DEC);
83     Serial.print('/');
84     Serial.print(future.month(), DEC);
85     Serial.print('/');
86     Serial.print(future.day(), DEC);
87     Serial.print(' ');
88     Serial.print(future.hour(), DEC);
89     Serial.print(':');
90     Serial.print(future.minute(), DEC);
91     Serial.print(':');
92     Serial.print(future.second(), DEC);
93     Serial.println();
94
95     Serial.print("Temperature: ");
96     Serial.print(rtc.getTemperature());
97     Serial.println(" C");
```



```
97   Serial.println();  
98   delay(3000);  
99 }
```

Listing 25: [RTCDs3231RTClib.ino](#)

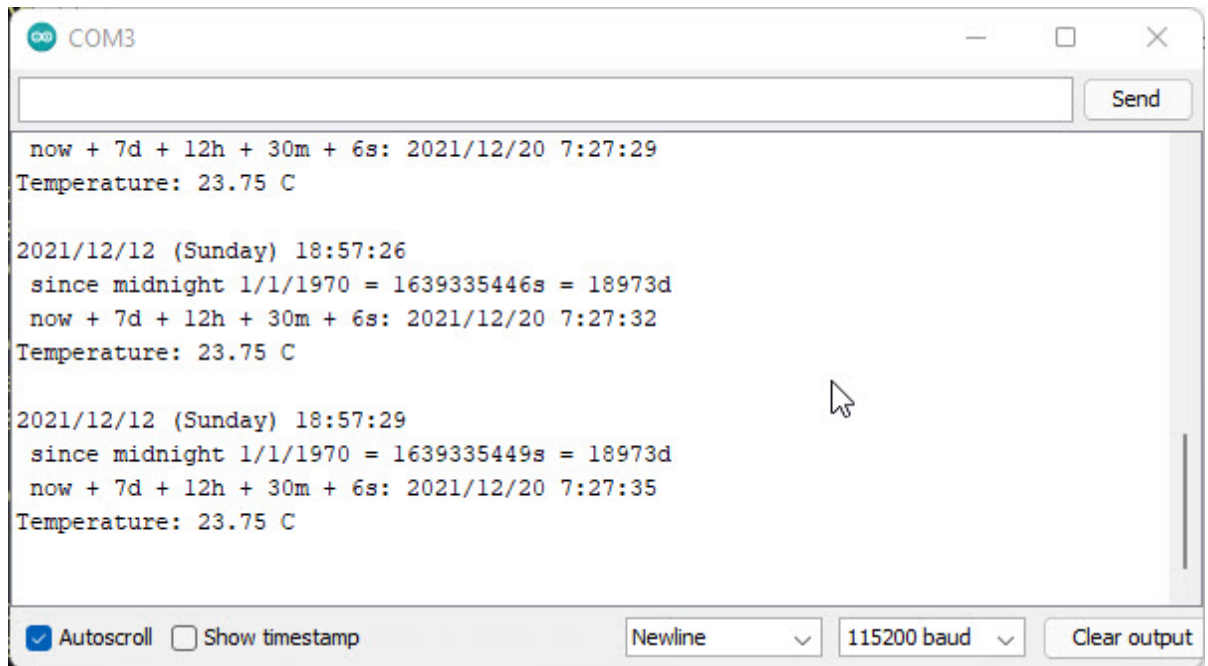


Abbildung 60: Datum-, Zeit-, und Temperatur-Ausgabe im Serial Monitor. Hier auch mit Zeitberechnungsbeispielen.

8.23 RFID RC522

RFID (Radio Frequency Identification) ist eine Technologie zur berührungsfreien und sichtkontaktfreien Datenübertragung. Ein RFID-System besteht aus einem Transponder (RFID-Tag) und einem Reader.

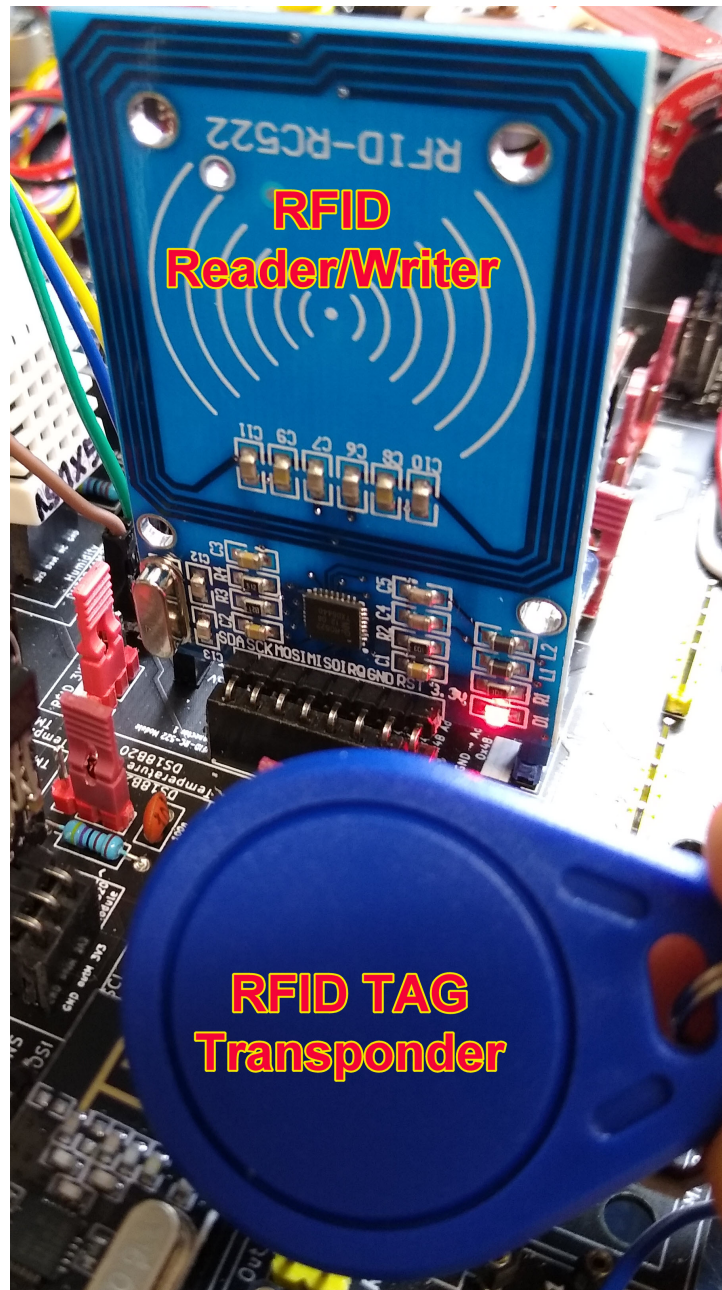


Abbildung 61: RFID-Modul RFID-RC522 mit TAG
notwendige Verbindungen für die SPI-Kommunikation:

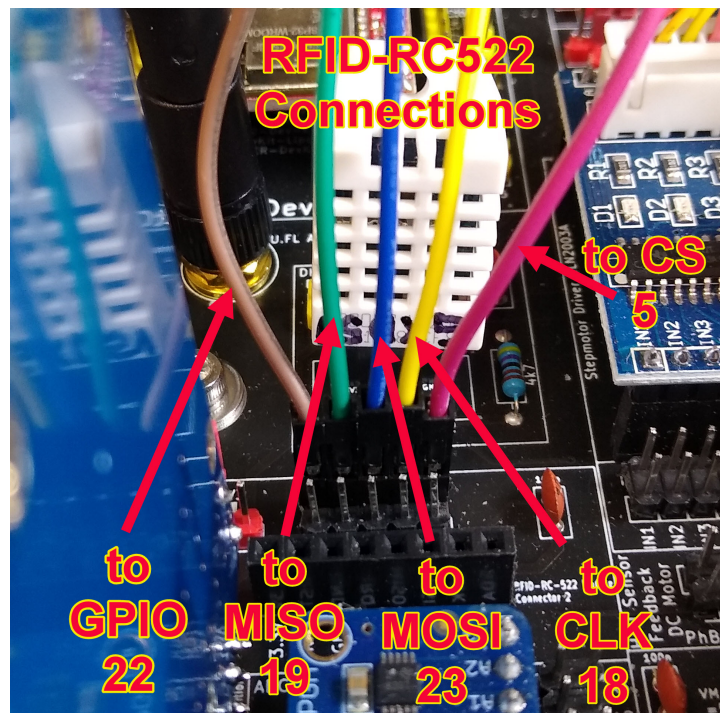


Abbildung 62: SPI-Verbindungen auf GPIOs des Microcontrollers
Library für die Demo-Sketches:

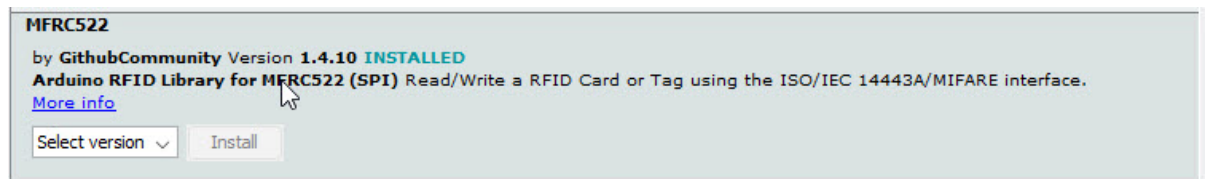


Abbildung 63: RFID-RC522 Verbindungen auf Microcontroller GPIOs
Der folgende Sketch zeigt den Inhalt eines Tags oder einer RFID-Karte. Es genügt, wenn Sie den Transponder einfach vor den Reader halten.

Source-Code [RFIDDumpCardInfo.ino](https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/RFIDDumpCardInfo/RFIDDumpCardInfo.ino):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/RFIDDumpCardInfo/RFIDDumpCardInfo.ino>

```
1  /*
2  * Function:  DUMP RFID Card-Info
3  * MCU:      Olimex ESP32-DevKit-LiPo
4  * Connections:
5  *           RFID Module      ESP32
6  *           =====
7  *           SDA               CS-5
8  *           SCK               CLK-18
9  *           MOSI              MOSI-23
10 *           MISO              MISO-19
11 *           RST               GPIO22
12 *
13 * Power On:  Jumper over white Marker Pins "RFID 3V3"
14 * File:      RFIDDumpCardInfo.ino
15 * Based on the RFID EXample.
16 * Author:    Thomas Jäggi
```



```
17  * Date:      13.12.2021
18  */
19
20 #include <SPI.h>
21 #include <MFRC522.h>
22
23 #define RST_PIN      22          // Configurable, see typical pin layout above
24 #define SS_PIN       5           // Configurable, see typical pin layout above
25
26 MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance
27
28 void setup() {
29   Serial.begin(115200); // Initialize serial communications with the PC
30   while (!Serial);     // Do nothing if no serial port is opened (added for Arduinos
31   based on ATMEGA32U4)
32   SPI.begin();          // Init SPI bus
33   mfrc522.PCD_Init();   // Init MFRC522
34   delay(4);             // Optional delay. Some board do need more time after init to be
35   ready, see Readme
36   mfrc522.PCD_DumpVersionToSerial(); // Show details of PCD - MFRC522 Card Reader
37   details
38   Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
39 }
40
41 void loop() {
42   // Reset the loop if no new card present on the sensor/reader. This saves the
43   // entire process when idle.
44   if ( ! mfrc522.PICC_IsNewCardPresent()) {
45     return;
46   }
47
48   // Select one of the cards
49   if ( ! mfrc522.PICC_ReadCardSerial()) {
50     return;
51   }
52
53   // Dump debug info about the card; PICC_HaltA() is automatically called
54   mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
55 }
```

Listing 26: [RFIDDumpCardInfo.ino](#)

Der Output kann im Serial Monitor betrachtet werden. Hier sind folgende Informationen zu sehen: Jedes RFID-Tag hat eine UID (Unique Identifier).

Der Kartentyp ist *myfare* mit einem Speichervolumen von 1KB. Die Struktur des Speichers hat 16 Sektoren 0-15 mit je 4 Blöcken 0-3. In der Ausgabe werden diese von 0 bis 63 durchnummeriert. Jeder Block kann 16 Bytes an Daten aufnehmen (Byte 0 bis 15).

Block 3 jedes Sektors wird als Sektor-Trailer bezeichnet und enthält Informationen (Zugriffsbits), um die Lese- und Schreibzugriffe auf die verbleibenden Blöcke in dem Sektor zu organisieren. Das heisst, nur die unteren 3 Blöcke (Block 0, 1 und 2) jedes Sektors stehen tatsächlich für die Datenspeicherung zur Verfügung. Die Speicherkapazität beträgt somit nur 48 Bytes netto.

Der Block 0 des Sektors 0 (am Ende des Dumps) ist der Herstellerblock mit IC-Herstellerdaten und UID.



COM3

Card UID: BA 19 F9 65

Card SAK: 08

PICC type: MIFARE 1KB

Sector	Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AccessBits
15	63	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	62	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	61	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
14	59	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	58	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	57	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
13	55	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	54	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	53	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	52	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
12	51	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	49	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	48	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
11	47	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]

Abbildung 64: Dump einer RFID-Karte

Hier der Dump einer RFID-Karte, wie sie in Hotels als Zimmerschlüssel verwendet wird. Die Sektoren 0 bis 13 sind lesegeschützt und mit einem Passwort verschlüsselt.

Card UID: 53 9D 2F 4C																		
Card SAK: 08																		
PICC type: MIFARE 1KB																		
Sector	Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AccessBits
15	63	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	62	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	61	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
14	59	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	58	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	57	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
13	55	PCD_Authenticate() failed: Error in communication.																
12	51	PCD_Authenticate() failed: Timeout in communication.																
11	47	PCD_Authenticate() failed: Timeout in communication.																
10	43	PCD_Authenticate() failed: Timeout in communication.																
9	39	PCD_Authenticate() failed: Timeout in communication.																
8	35	PCD_Authenticate() failed: Timeout in communication.																
7	31	PCD_Authenticate() failed: Timeout in communication.																
6	27	PCD_Authenticate() failed: Timeout in communication.																
5	23	PCD_Authenticate() failed: Timeout in communication.																
4	19	PCD_Authenticate() failed: Timeout in communication.																
3	15	PCD_Authenticate() failed: Timeout in communication.																
2	11	PCD_Authenticate() failed: Timeout in communication.																
1	7	PCD_Authenticate() failed: Timeout in communication.																
0	3	PCD_Authenticate() failed: Timeout in communication.																

Abbildung 65: Dump einer RFID-Karte eines Hotels als Zimmerschlüssel

Der folgende Sketch simuliert eine **Zugangskontrolle**. Passen Sie Ihre UID, sichtbar im Dump vom vorhergehenden Sketch, im Code an (Zeile 60). Hier werden zusätzlich 2 LEDs



benötigt: Die rote LED leuchtet, wenn der Zugang nicht gewährt wird; die grüne LED leuchtet, wenn die UID im Code mit der UID Ihrer Karte übereinstimmt. Entnehmen Sie die GPIOs für die beiden LEDs aus der Beschreibung im Sketch.

Source-Code [RFIDcheckTagLED.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/RFIDcheckTagLED/RFIDcheckTagLED.ino>

```
2  /*
3  * Function:  DUMP RFID Card-Info and compare UID: ok: LED green; not ok: LED red
4  * MCU:      Olimex ESP32-DevKit-LiPo
5  * Connections:
6  *           RFID Module      ESP32
7  *           =====
8  *           SDA               CS-5
9  *           SCK               CLK-18
10 *           MOSI              MOSI-23
11 *           MISO              MISO-19
12 *           RST               GPIO22
13 *           =====
14 *           LED connections
15 *           =====
16 *           LED red           GPIO4
17 *           LED green         GPIO2
18 *
19 * Power On:  Jumper over white Marker Pins "RFID 3V3"
20 * File:      RFIDcheckTagLED.ino
21 * Based on the RFID EXample.
22 * Author:    Thomas Jäggi
23 * Date:      13.12.2021
24 */
25
26 #include <SPI.h>
27 #include <MFRC522.h>           // für RFID-RC522
28
29 #define LEDr_PIN    4         // LED-Pin
30 #define LEDgn_PIN   2         // LED-Pin
31 #define RST_PIN     22        // RST-Pin
32 #define SDA_PIN     5         // SS- bzw. SDA-Pin
33
34 MFRC522 mfrc522(SDA_PIN, RST_PIN); // MFRC522-Instanz erstellen
35
36 void setup() {
37   Serial.begin(115200);
38   SPI.begin();                // SPI-Bus initialisieren
39   mfrc522.PCD_Init();         // RFID-RC522 initialisieren
40   Serial.println("Ein RFID-Tag an den Leser halten ...");
41   Serial.println();
42   pinMode(LEDr_PIN, OUTPUT);
43   pinMode(LEDgn_PIN, OUTPUT);
44   digitalWrite(LEDr_PIN, LOW);
45   digitalWrite(LEDgn_PIN, LOW);
46 }
47
48 void loop() {
49   String rfidTag = "";
50   if (mfrc522.PICC_IsNewCardPresent() &&          // liegt ein neues Tag vor
51       mfrc522.PICC_ReadCardSerial()) {            // richtige Tag-Art
52     Serial.print("RFID-Tag/Karten-ID:");
53     for (byte i = 0; i < mfrc522.uid.size; i++) { // Karten-ID extrah.
54       Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
55       Serial.print(mfrc522.uid.uidByte[i], HEX);
56       rfidTag.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
57       rfidTag.concat(String(mfrc522.uid.uidByte[i], HEX));
58     }
59   }
60 }
```



```
58 Serial.println();  
59 Serial.print("Aktion: ");  
60 rfidTag.toUpperCase();  
61 if (rfidTag.substring(1) == "41 8E B9 45") { // zuzulassendes RFID-Tag  
62     Serial.println("Zugang genehmigt \n");  
63     digitalWrite(LED_R_PIN, LOW);  
64     digitalWrite(LED_GN_PIN, HIGH);  
65     delay(3000);  
66 } else {  
67     Serial.println("Zugang verweigert!\n");  
68     digitalWrite(LED_GN_PIN, LOW);  
69     digitalWrite(LED_R_PIN, HIGH);  
70     delay(3000);  
71 }  
72 }
```

Listing 27: RFIDcheckTagLED.ino

Weitere Informationen zu RFID:

[https://learn.adafruit.com/adafruit-pn532-rfid-nfc/mifare:](https://learn.adafruit.com/adafruit-pn532-rfid-nfc/mifare)

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/DatasheetsComponents/RFIDMFRC522.p>

Demosketches der Library in der Arduino-IDE unter FILE/EXAMPLES/MFRC522

8.24 Lichtsensor BH1750

Der BH1750FVI ist ein Lichtsensor mit I^2C -Schnittstelle. Der Sensor kann in unterschiedlichen Auflösungsmodi betrieben werden. Die Ausgabe der Lichtstärke im Serial Monitor erfolgt direkt in Lux.

Mehr Details zum Sensor, siehe

<https://wolles-elektronikkiste.de/bh1750fvi-lichtsensormodul>.

Library:

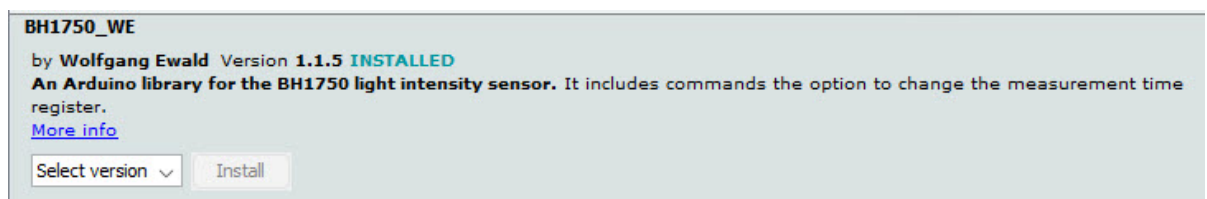


Abbildung 66: Library für BH1750FVI

Source-Code [LightSensorBH1750.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/LightSensorBH1750/LightSensorBH1750.ino>

```
1 /*****  
2 * Example sketch for the BH1750_WE library  
3 *  
4 * Mode selection / abbreviations:  
5 * CHM:    Continuously H-Resolution Mode  
6 * CHM_2:  Continuously H-Resolution Mode2  
7 * CLM:    Continuously L-Resolution Mode  
8 * OTH:    One Time H-Resolution Mode  
9 * OTH_2:  One Time H-Resolution Mode2  
10 *****/
```



```
10 * OTL:    One Time L-Resolution Mode
11 *
12 * Measuring time factor:
13 * 1.0 ==> Mesuring Time Register = 69
14 * 0.45 ==> Measuring Time Register = 31
15 * 3.68 ==> Mesuring Time Register = 254
16 *
17 * Other implemented functions, not used in the example:
18 * resetDataReg() --> rests Data Register
19 * powerOn() --> Wake Up!
20 * powerDown() --> Sleep well, my BH1750
21 *
22 * If you change the measuring time factor for calibration purpose,
23 * then you need to devide the light intensity by the measuring time factor
24 *
25 * Further information can be found on:
26 * https://wolles-elektronikkiste.de/en/bh1750fvi-gy-30-302-ambient-light-sensor
27 * or in German:
28 * https://wolles-elektronikkiste.de/bh1750fvi-lichtsensormodul
29 * *****/
30
31 #include <Wire.h>
32 #include <BH1750_WE.h>
33 #define BH1750_ADDRESS 0x23
34
35 BH1750_WE myBH1750(BH1750_ADDRESS);
36 // You may also pass a TwoWire object like wire2
37 // BH1750_WE myBH1750(&wire2, BH1750_ADDRESS);
38
39 void setup(){
40   Serial.begin(115200);
41   Wire.begin();
42   myBH1750.init(); // sets default values: mode = CHM, measuring time factor = 1.0
43   // myBH1750.setMode(CLM); // uncomment if you want to change default values
44   // myBH1750.setMeasuringTimeFactor(0.45); // uncomment for selection of value
45   // between 0.45 and 3.68
46 }
47
48 void loop(){
49   float lightIntensity = myBH1750.getLux();
50   Serial.print(F("Lichtstärke: "));
51   Serial.print(lightIntensity);
52   Serial.println(F(" Lux"));
53   delay(1000);
54 }
```

Listing 28: [LightSensorBH1750.ino](#)

Output im Serial Monitor:

```
COM3
Lichtstärke: 327.50 Lux
Lichtstärke: 327.50 Lux
Lichtstärke: 327.50 Lux
Lichtstärke: 326.67 Lux
Lichtstärke: 327.50 Lux
Lichtstärke: 327.50 Lux
Lichtstärke: 327.50 Lux
```

Abbildung 67: Luxmessung mit BH1750FVI



8.25 IMU MPU9250-9-DOF

Der MPU-9250 gehört zu den **Inertial Measurement Units (IMU)**. Es ist ein **9-Achsen MEMS Sensor** von InvenSense®. Damit kann man Bewegungsdaten aufzeichnen.

Der Sensor misst Beschleunigung (Accelerometer), Winkelgeschwindigkeit oder Rotationsgeschwindigkeit (Gyroskop) und ist zusätzlich auch ein Kompass (Magnetometer). Das sind 3 verschiedene Kräfte in 3 Dimensionen. Daher kommt der Begriff der 9 Achsen.

Der Sensor wird mit dem I²C-Busprotokoll betrieben. Es sind neben der Spannungsversorgung nur 2 Verbindungen notwendig: SCL und SDA.

Die Features des Sensors sind in folgender Tabelle dargestellt:

MPU-9250		
Gyroscope Features	Accelerometer Features	Magnetometer
Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with User-programmable full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^\circ/\text{sec}$ and integrated 16-bit ADCs Digitally-programmable low-pass filter Gyroscope operating current: 3.2mA Sleep mode current: 8 μ A Factory calibrated sensitivity scale factor Self-test	Digital-output triple-axis accelerometer with a programmable full scale range $\pm 2g$, $\pm 4g$, $\pm 8g$ and $\pm 16g$ and integrated 16-bit ADCs Accelerometer normal operating current: 450 μ A Low power accelerometer mode current: 8.4 μ A at 0.98Hz, 19.8 μ A at 31.25Hz Sleep mode current: 8 μ A User-programmable interrupts Wake-on-motion interrupt for low power operation of applications processor Self test	3-axis silicon monolithic hall-effect magnetic sensor with magnetic concentrator Wide dynamic measurement range and high resolution with lower current consumption Magnetometer normal operating current: 280 μ A at 8Hz repetition rate Output data resolution of 14 bit (0.6 μ T/LSB) Full-scale measurement range is $\pm 4800\mu$ T Self-test function with an internal magnetic source to confirm magnetic sensor operation on end products

Abbildung 68: Leistungsmerkmale des MPU9250-IMU-Sensors

Der MPU9250 wird in vielen Anwendungen eingesetzt, z. Bsp.

- › Smartphones
- › Tablets
- › 3D-Remote Control Einheiten für TVs und set-top Boxen.
- › Bewegungssensitive Gamecontroller und 3D-Mäuse
- › Portable Gaming-Geräte
- › Wearable Sensors; Sensoren, die in Kleidern getragen werden.

Eine gute Anleitung mit vielen Demo-Sketches finden Sie [hier](https://wolles-elektronikkiste.de/mpu9250-9-achsen-sensormodul-teil-1).



Installieren Sie für die Beispiele die Library

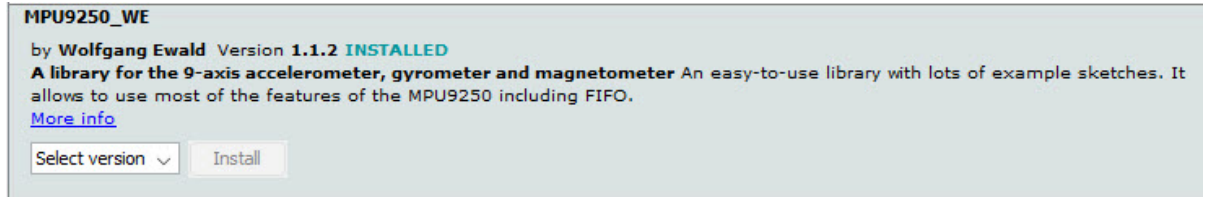


Abbildung 69: Library für MPU9250

Beispiel: mit folgendem Sketch kann die Neigung des Boards in ° für alle 3 Achsen ermittelt werden:

Source-Code [MPU9250_angles_and_orientation.ino](https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/MPU9250_angles_and_orientation/MPU9250_angles_and_orientation.ino):

https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/MPU9250_angles_and_orientation/MPU9250_angles_and_orientation.ino

```
1 /*****
2  * Example sketch for the MPU9250_WE library
3  *
4  * This sketch shows how measure x,y angles with high accuracy up to ~60Grad.
5  * In addition the sketch provides the orientation of the modules, i.e. the
6  * axis with the highest positive g value.
7  *
8  * For further information visit my blog:
9  *
10 * https://wolles-elektronikkiste.de/mpu9250-9-achsen-sensormodul-teil-1 (German)
11 * https://wolles-elektronikkiste.de/en/mpu9250-9-axis-sensor-module-part-1 (
12   English)
13 *****/
14
15 #include <MPU9250_WE.h>
16 #include <Wire.h>
17 #define MPU9250_ADDR 0x68
18
19 /* There are several ways to create your MPU9250 object:
20  * MPU9250_WE myMPU9250 = MPU9250_WE() -> uses Wire / I2C Address = 0
21   x68
22  * MPU9250_WE myMPU9250 = MPU9250_WE(MPU9250_ADDR) -> uses Wire / MPU9250_ADDR
23  * MPU9250_WE myMPU9250 = MPU9250_WE(&wire2) -> uses the TwoWire object
24   wire2 / MPU9250_ADDR
25  * MPU9250_WE myMPU9250 = MPU9250_WE(&wire2, MPU9250_ADDR) -> all together
26  * Successfully tested with two I2C busses on an ESP32
27  */
28 MPU9250_WE myMPU9250 = MPU9250_WE(MPU9250_ADDR);
29
30 void setup() {
31   Serial.begin(115200);
32   Wire.begin();
33   if(!myMPU9250.init()){
34     Serial.println("MPU9250 does not respond");
35   }
36   else{
37     Serial.println("MPU9250 is connected");
38   }
39
40   /* The slope of the curve of acceleration vs measured values fits quite well to
41    the theoretical
42    * values, e.g. 16384 units/g in the +/- 2g range. But the starting point, if you
43    position the
44    * MPU9250 flat, is not necessarily 0g/0g/1g for x/y/z. The autoOffset function
45    measures offset
```



```
41 * values. It assumes your MPU9250 is positioned flat with its x,y-plane. The
    * more you deviate
    * from this, the less accurate will be your results.
43 * The function also measures the offset of the gyroscope data. The gyroscope
    * offset does not
    * depend on the positioning.
45 * This function needs to be called at the beginning since it can overwrite your
    * settings!
    */
47 Serial.println("Position you MPU9250 flat and don't move it - calibrating...");
    delay(1000);
49 myMPU9250.autoOffsets();
    Serial.println("Done!");
51
    /* This is a more accurate method for calibration. You have to determine the
        minimum and maximum
53 * raw acceleration values of the axes determined in the range +/- 2 g.
        * You call the function as follows: setAccOffsets(xMin,xMax,yMin,yMax,zMin,zMax
        );
55 * Use either autoOffset or setAccOffsets, not both.
        */
57 //myMPU9250.setAccOffsets(-14240.0, 18220.0, -17280.0, 15590.0, -20930.0,
        12080.0);

59 /* Sample rate divider divides the output rate of the gyroscope and
        accelerometer.
        * Sample rate = Internal sample rate / (1 + divider)
61 * It can only be applied if the corresponding DLPF is enabled and 0<DLPF<7!
        * Divider is a number 0...255
        */
63 //myMPU9250.setSampleRateDivider(5);
65
    /* MPU9250_ACC_RANGE_2G      2 g
67 * MPU9250_ACC_RANGE_4G      4 g
        * MPU9250_ACC_RANGE_8G      8 g
69 * MPU9250_ACC_RANGE_16G     16 g
        */
71 myMPU9250.setAccRange(MPU9250_ACC_RANGE_2G);

73 /* Enable/disable the digital low pass filter for the accelerometer
        * If disabled the bandwidth is 1.13 kHz, delay is 0.75 ms, output rate is 4 kHz
75 *
        myMPU9250.enableAccDLPF(true);
77
    /* Digital low pass filter (DLPF) for the accelerometer (if DLPF enabled)
79 * MPU9250_DPLF_0, MPU9250_DPLF_2, ..... MPU9250_DPLF_7
        * DLPF      Bandwidth [Hz]      Delay [ms]      Output rate [kHz]
81 *      0          460                1.94             1
        *      1          184                5.80             1
83 *      2          92                 7.80             1
        *      3          41                11.80            1
85 *      4          20                 19.80            1
        *      5          10                 35.70            1
87 *      6           5                 66.96             1
        *      7          460                1.94             1
89 */
    myMPU9250.setAccDLPF(MPU9250_DLPF_6);
91 }

93 void loop() {
    xyzFloat gValue = myMPU9250.getGValues();
95 xyzFloat angle = myMPU9250.getAngles();

97 /* For g-values the corrected raws are used */
```



```
99 Serial.print("g-x      = ");
Serial.print(gValue.x);
101 Serial.print(" | g-y      = ");
Serial.print(gValue.y);
103 Serial.print(" | g-z      = ");
Serial.println(gValue.z);

105 /* Angles are also based on the corrected raws. Angles are simply calculated by
   angle = arcsin(g Value) */
107 Serial.print("Angle x = ");
Serial.print(angle.x);
109 Serial.print(" | Angle y = ");
Serial.print(angle.y);
111 Serial.print(" | Angle z = ");
Serial.println(angle.z);

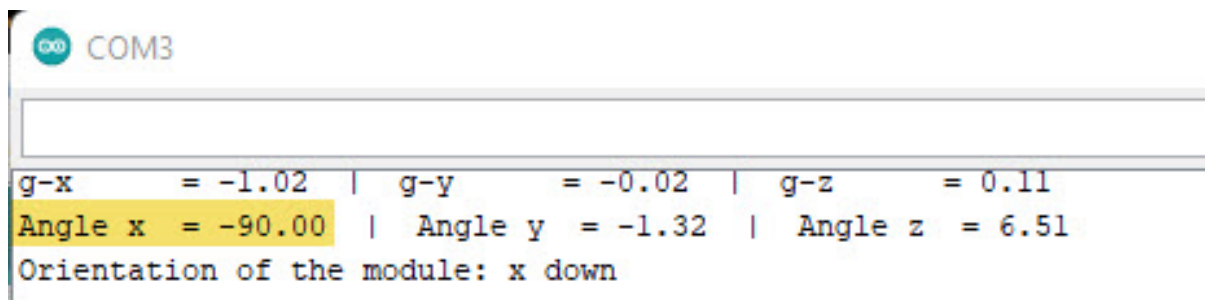
113
115 Serial.print("Orientation of the module: ");
Serial.println(myMPU9250.getOrientationAsString());

117 Serial.println();

119 delay(1000);

121 }
```

Listing 29: MPU9250_angles_and_orientation.ino



```
COM3

g-x      = -1.02 | g-y      = -0.02 | g-z      = 0.11
Angle x = -90.00 | Angle y = -1.32 | Angle z = 6.51
Orientation of the module: x down
```

Abbildung 70: Board steht -90° zur x-Achse

8.26 Infrarot LED - Infrarot Diode

to do

8.27 TSOP4838-IR-Empfänger und Remote Control

to do

8.28 NRF24L01-2.4GHz Transceiver

Das NRF24L01 ist ein Funkmodul, welches als Sender und als Receiver (= Transceiver) verwendet werden kann. Das Modul sendet und empfängt auf dem 2.4GHz ISM Band im Bereich von 2400MHz bis 2.525Mhz Die Bandbreite eines Kanals ist 1 Mhz. Somit ergeben sich 125 mögliche Kanäle.

einige Eigenschaften (auf Englisch):

- › This transceiver module consists of a frequency generator, shock burst mode controller, power amplifier, crystal oscillator modulator, and demodulator.



- › When transmitting power is zero dBm it uses only 11.3 mA of current, while during receiving mode, it uses 13.5 mA of current.
- › This module is designed for long distance (ca. 100m) and fast transmission of data.
- › It is designed to work through an SPI protocol.
- › Air data transmission rate of NRF24L01 is around 1-2 Mbps.
- › Its high air data rate combined with power saving mode makes it very favorable for ultra-low power applications.
- › Its internal voltage regulator controls a high power supply rejection ratio and power supply range.
- › This module has a compact size, and can easily be used in confined spaces.
- › This module is designed to operate at 3.3 volts.
- › This module has an address range of 125 and it can communicate with six other modules. By using this feature, we can use it in mesh networks and other networking applications.
- › Its maximum output power is +20 dBm.

NRF24L01 SPI Interface:

NRF24L01 SPI INTERFACING

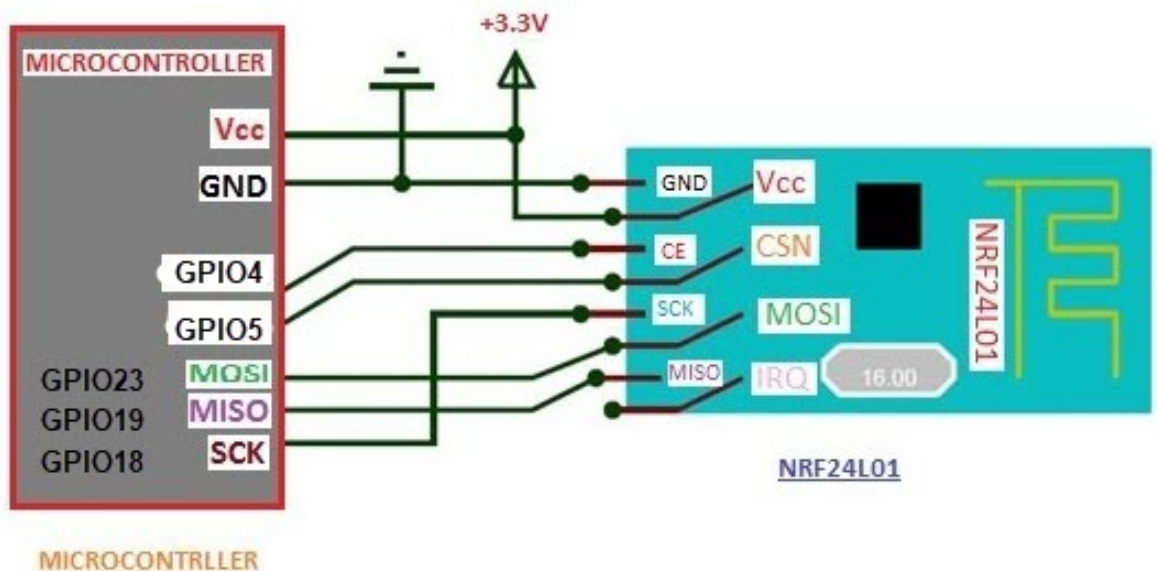


Abbildung 71: SPI-Interface des Transceiver Moduls NRF24L01

Anwendungsfälle

- › Überall dort, wo eine Anwendung ferngesteuert werden muss und kein WiFi verwendet werden kann oder soll. Beispielsweise ein Sensor liefert Daten aus dem Garten und hat dort keinen WLAN-Empfang. Das Signal kann in diesem Fall an einen Gateway gesendet werden, welcher mit dem WLAN verbunden ist. Dieses Szenario bauen wir im zweiten Beispiel. Siehe Abschnitt [8.28.2](#).
- › In "meshed" networks. Mehrere Module als Sender können mit einem Empfänger kommunizieren.
- › Das Modul wird auch in Funkfernbedienungen, z.Bsp. bei Spielzeugen, eingesetzt.

Für die folgenden zwei Beispiele gilt:



Arbeiten Sie im Tandem! Ein IoT-Education-Board ist der Sender und das andere der Empfänger. Im zweiten Beispiel verwenden wir den Empfänger auch als Gateway ins Internet. Beachten Sie auch, dass der Wert der Variable **address**, Zeile 35 im Sketch, bei Sender und Empfänger gleich sein muss!

8.28.1 Einfache Meldung über Funk senden und empfangen

Hier versenden wir eine *Hello* - Nachricht vom Sender-Board zum Empfänger-Board und stellen die Nachricht im Serial Monitor dar. Damit die Sketches funktionieren, müssen die Verbindungen wie in Abbildung 71 gesteckt werden. Die verwendeten GPIOs sind auch im Source-Code beschrieben.

Sketch für den **SENDER**:

Source-Code [NRF24L01ESP32SENDER.ino](#):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/NRF24L01ESP32SENDER/NRF24L01ESP32SENDER.ino>

```
1  /*
2   * Function:
3   * Send Data with NRF24L01 over 2.4GHz This Sketch is the Sender-Part and works
4   * together with NRF24L01ESP32RECEIVER.ino
5   *
6   * MCU:      Olimex ESP32-DevKit-LiPo
7   * File:      NRF24L01ESP32SENDER.ino
8   * Libraries: RF24 TMRh20 2020 - Optimized fork of the nRF24L01+ driver
9   * Connections:
10  *
11  * NRF24L01      ESP32
12  *=====
13  *CE              4
14  *SCK             18
15  *MISO            19
16  *CNS=CSN         5
17  *MOSI            23
18  *IRQ             not used
19  *
20  * Power activate: Jumper over Marker NRF24L01 3V3
21  *
22  * Output:        Browser, Serial Monitor with 115200baud
23  * Author:        Thomas Jäggi
24  * Date:          05.01.2022
25  *
26  */
27 #include <SPI.h>
28 #include <nRF24L01.h>
29 #include <RF24.h>
30
31 #define RF24_CHANNEL 83
32
33 RF24 radio(4, 5); // CE, CSN on ESP32
34 //const uint64_t address = 0xF0F0F0F0E1LL; // Address as a long HEX-Value
35 const int address = 999; // change this address to correspond with receiver.
36 int counter = 0;
37
38 void setup()
```




```
39 {
  Serial.begin(115200);
41  radio.begin(); //Starting the Wireless communication
  radio.openWritingPipe(address); //Setting the address where we will send the data
43  radio.setPALevel(RF24_PA_MIN); //You can set it as minimum or maximum depending on
    the distance between the transmitter and receiver.
  radio.stopListening(); //This sets the module as transmitter
45 }

47 void loop()
  {
49  char text[] = "Hi Gebäudeinformatiker!!";
  char str[32];
51  sprintf(str,"%s %d",text,counter);
  radio.write(&str, sizeof(str));
53
  Serial.println(str);
55  counter++;
  delay(20);
57 }
```

Listing 30: [NRF24L01ESP32SENDER.ino](#)

Sketch für den **EMPFÄNGER**:

Source-Code [NRF24L01ESP32RECEIVER.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/NRF24L01ESP32RECEIVER/NRF24L01ESP32RECEIVER.ino>

```
1  /*
   * Function:
3  * Read Data from NRF24L01. This Sketch is the Receiver-Part and works
   * together with NRF24L01ESP32SENDER.ino
5  *
   * MCU:      Olimex ESP32-DevKit-LiPo
7  * File:      NRF24L01ESP32RECEIVER.ino
   * Libraries: RF24 TMRh20 2020 - Optimized fork of the nRF24L01+ driver
9  * Connections:
   *
11 * NRF24L01      ESP32
   *=====
13 *CE              4
   *SCK             18
15 *MISO           19
   *CNS=CSN         5
17 *MOSI           23
   *IRQ             not used
19 *
   * Power activate: Jumper over Marker NRF24L01 3V3
21 *
   * Output:        Browser, Serial Monitor with 115200baud
23 * Author:        Thomas Jäggi
   * Date:          05.01.2022
25 *
   */
27
   #include <SPI.h>
29 #include <nRF24L01.h>
   #include <RF24.h>
31
   RF24 radio(4, 5); // CE, CSN on ESP32
33 //const uint64_t address = 0xF0F0F0F0E1LL; // Address as a long HEX-Value
   const int address = 999; // change this address to correspond with sender.
```



```
35 boolean button_state = 0;

37 void setup()
{
39   Serial.begin(115200);
   radio.begin();
41   Serial.print("ADDRESS :");
   radio.openReadingPipe(0, address); //Setting the address at which we will receive
   the data
43   radio.setPALevel(RF24_PA_MIN); //You can set this as minimum or maximum
   depending on the distance between the transmitter and receiver.
   radio.startListening(); //This sets the module as receiver
45 }
void loop()
47 {
   if (radio.available()) //Looking for the data.
49 {
   Serial.print("NRF24L01 snifft auf 2.4GHz ...");Serial.println("");
51
   char text[32] = ""; //Saving the incoming data
53   radio.read(&text, sizeof(text)); //Reading the data
   Serial.print(text);Serial.println("");
55 }
}
```

Listing 31: [NRF24L01ESP32RECEIVER.ino](#)

Ausgaben im Serial Monitor des Senders und des Empfängers:

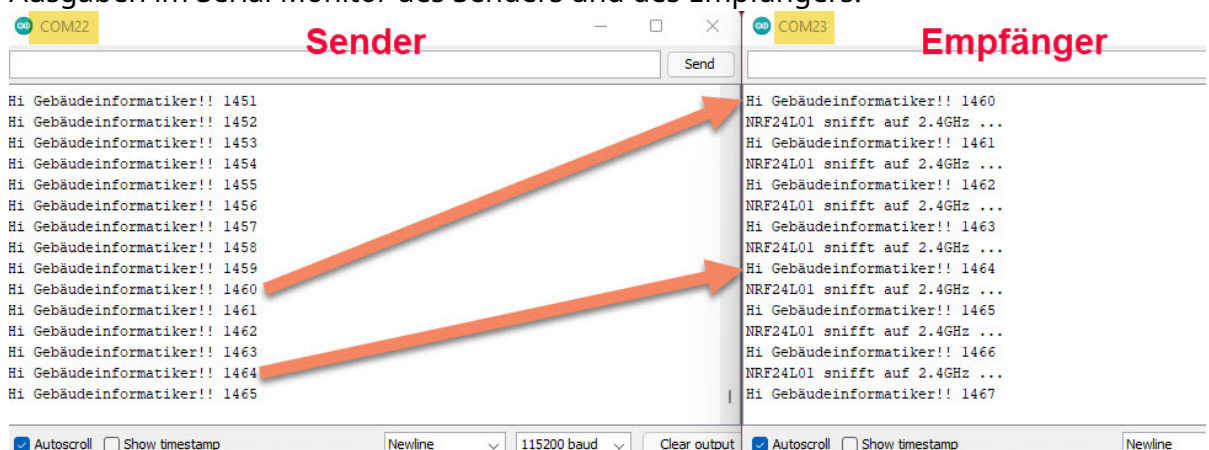


Abbildung 72: Die gesendete Nachricht wird beim Sender und Empfänger angezeigt. Jede Nachricht ist nummeriert.

8.28.2 ESP32 als Funkempfänger und Gateway ins Internet

Nun wollen wir Sensordaten über Funk an den Empfänger senden. Der Empfänger soll gleichzeitig auch die Rolle eines Gateways ins Internet übernehmen. Das Ziel soll sein, dass die Sensordaten via dem Gateway an einen Cloud-Dienst gesendet werden.

Ein Board konfigurieren wir als **Sensorboard** (ohne WLAN, nur mit Funkmodul) und das andere als **Gateway** (mit Funkmodul und WLAN), welches die Werte an einen Cloud-Dienst sendet. Der Dienst bereitet anschliessend die Sensordaten grafisch auf. Als Sensor verwenden wir den BME280. Dieser liefert Temperatur, Feuchtigkeit, Luftdruck und Meereshöhe Ihres Standortes.



Mit diesem Szenario können auch Microcontroller internet-fähig gemacht werden, ohne dass diese eine WLAN- oder eine Ethernet-Schnittstelle haben müssen.

Als Alternative könnte auch LoRa-WAN mit einem LoRa-WAN-Gateway verwendet werden. LoRa-WAN ist jedoch eine andere spannende Thematik...

Als kostenlosen Cloud-Dienst verwenden wir thingspeak.com.
Registrieren Sie sich und erstellen Sie einen Channel für die Sensorwerte mit folgenden Feldern:

ThingSpeak™ Channels Apps Devices Support

Private View Public View Channel Settings Sharing API Keys D

Channel Settings

Percentage complete 30%

Channel ID 1626265

Name ESP32-Test

Description

Field 1 Temperature ☒

Field 2 Pressure ☒

Field 3 Altitude ☒

Field 4 Humidity ☒

Field 5 ☐

Abbildung 73: Channel-Settings auf Thingspeak.com

Machen Sie sich auch eine Notiz des *Write API-Keys*, welchen Sie im Reiter *API-Keys* finden. Dieser Key wird für die Authentisierung beim Dienst verwendet und muss im Sketch von Listing 33, Zeile 37, eingetragen werden. Den Key finden Sie gemäß folgender Abbildung. Verwenden Sie nicht diesen Key, sondern Ihren eigenen!



ThingSpeak™

Channels ▾ Apps ▾ Devices ▾ Support ▾

ESP32-Test

Channel ID: 1626265
Author: mwa0000025197602
Access: Public

Private View Public View Channel Settings Sharing API Keys

Write API Key

Key

Generate New Write API Key

Abbildung 74: Ihr eigener *Write API Key* wird für die Authentisierung benötigt.
Sketch für das **SENSORBOARD**:

Aktivieren Sie den BME280-Sensor. Hier muss lediglich SCL und SDA mit dem ESP32 verbunden werden. Vergessen Sie nicht, die Spannung einzuschalten: Jumper auf weissen Marker bei *BMP280/388* 3V3 setzen.

Source-Code [NRF24L01SensorNodeSender.ino](https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/NRF24L01SensorNodeSender/NRF24L01SensorNodeSender.ino):

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/ArduinoDemoSketches/NRF24L01SensorNodeSender/NRF24L01SensorNodeSender.ino>

```
1  /*
2  * Function:
3  * Reads Sensor-Data from BME280 and sends them with NRF24L01 over 2.4GHz to the
4  * Gateway.
5  *
6  * Use Sketch together with NRF24L01ESP32GatewayThingspeak.ino
7  *
8  * MCU:      Olimex ESP32-DevKit-LiPo
9  * File:      NRF24L01SensorNodeSender.ino
10 * Libraries to install:  RF24 TMRh20 2020 - Optimized fork of the nRF24L01+
11 * driver
12 *                      Adafruit_Sensor and Adafruit_BME280
13 * Connections:
14 *
15 * NRF24L01      ESP32
16 *=====
17 *CE              4
18 *SCK             18
19 *MISO            19
20 *CNS=CSN         5
21 *MOSI            23
22 *IRQ             not used
23 *
24 * Power activate: Jumper over Marker NRF24L01 3V3
25 *
26 * Output:      Serial Monitor with 115200baud and thingspeak.com
27 * Author:      Thomas Jäggi
28 * Date:        05.01.2022
29 *
30 */
#include <SPI.h>
```



```
32 #include <Wire.h>
    #include <Adafruit_Sensor.h>
34 #include <Adafruit_BME280.h>
    #include <nRF24L01.h>
36 #include <RF24.h>

38 RF24 radio(4, 5); // CE, CSN on ESP32
    const uint64_t address = 0xF0F0F0F0E1LL; // Make sure, Receiver has the same
        address!!
40 // Adresse ist hier als HEX-Wert dargestellt. Kann beliebig geändert werden.
    // Der Empfänger muss dieselbe Adresse haben.
42 // Bedeutung von LL am Ende: LL is the suffix for long-long,
    // which is 64-bit on most C/C++ implementations.
44 // So 0LL is a 64-bit literal with the value of 0.

46 int counter = 0;

48
50 float temperature;
51 float humidity;
52 float altitude;
53 float pressure;

54 #define SEALEVELPRESSURE_HPA (1014.00)
    // see Sealevel-Pressure of today: https://meteologix.com/observations/pressure-qnh.html
56 Adafruit_BME280 bme;

58 struct MyData
    {
60     int counter;
        float temperature;
62     float humidity;
        float altitude;
64     float pressure;
    };
66 MyData data;

68 void setup()
    {
70     Serial.begin(115200);
        radio.begin(); //Starting the Wireless communication
72     radio.openWritingPipe(address); //Setting the address where we will send the data
        radio.setPALevel(RF24_PA_MIN); //You can set it as minimum or maximum depending on
            the distance between the transmitter and receiver.
74     radio.stopListening(); //This sets the module as transmitter

76     if (!bme.begin(0x76))
        {
78         Serial.println("Could not find a valid BME280 sensor, check wiring!");
            while (1);
80     }

82 }

84 void loop()
    {
86     data.counter = counter;
        data.temperature = bme.readTemperature();
88     data.pressure = bme.readPressure() / 100.0F;
        data.altitude = bme.readAltitude(SEALEVELPRESSURE_HPA);
90     data.humidity = bme.readHumidity();

92     Serial.print("Packet No. = ");
```




```
94 Serial.println(data.counter);
95
96 Serial.print("Temperature = ");
97 Serial.print(data.temperature);
98 Serial.println("°C");
99
100 Serial.print("Pressure = ");
101 Serial.print(data.pressure);
102 Serial.println("hPa");
103
104 Serial.print("Approx. Altitude = ");
105 Serial.print(data.altitude);
106 Serial.println("m");
107
108 Serial.print("Humidity = ");
109 Serial.print(data.humidity);
110 Serial.println("%");
111
112 Serial.println();
113
114 radio.write(&data, sizeof(MyData));
115
116 Serial.println("Data Packet Sent");
117 Serial.println("");
118
119 counter++;
120 delay(5000);
121 }
```

Listing 32: [NRF24L01SensorNodeSender.ino](#)

Sketch für den **GATEWAY**:

Source-Code [NRF24L01ESP32GatewayThingspeak.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/NRF24L01ESP32GatewayThingspeak/NRF24L01ESP32GatewayThingspeak.ino>

```
1 /*
2  * Function:
3  *   Receives Sensor-Data with NRF24L01 over 2.4GHz from SensorNode and acts as a
4  *   Gateway
5  *   to the thingspeak.com-Cloud.
6  *
7  *   Use Sketch together with NRF24L01SensorNodeSender.ino
8  *
9  *   MCU:      Olimex ESP32-DevKit-LiPo
10  *   File:      NRF24L01ESP32GatewayThingspeak.ino
11  *   Libraries to install: RF24 TMRh20 2020 - Optimized fork of the nRF24L01+
12  *   driver
13  *   Connections:
14  *
15  *   NRF24L01      ESP32
16  *   =====
17  *   *CE            4
18  *   *SCK            18
19  *   *MISO            19
20  *   *CNS=CSN        5
21  *   *MOSI            23
22  *   *IRQ            not used
23  *
24  *   Power activate: Jumper over Marker NRF24L01 3V3
25  *
26  *   YOU NEED A WRITE API-KEY !! Note it below into the apiKey-Variable.
```



```
26 * Change your SSID-Settings!  
27 *  
28 * Output:      Serial Monitor with 115200baud and thingspeak.com  
29 * Author:      Thomas Jäggi  
30 * Date:        05.01.2022  
31 *  
32 */  
33 #include <WiFi.h>  
34 #include <SPI.h>  
35 #include <nRF24L01.h>  
36 #include <RF24.h>  
37  
38 String apiKey = "CKV42YZ1J617FQC5"; // get YOUR Key from thingspeak.com.  
39                                     // You have to register. It's free!  
40  
41 const char* ssid = "YourSSID";  
42 const char* password = "yourSSIDPassword";  
43  
44 const char* server = "api.thingspeak.com";  
45  
46 RF24 radio(4, 5); // CE, CSN on ESP32  
47 const uint64_t address = 0xF0F0F0F0E1LL; // Make sure, Sender has the same address  
48 !!  
49 // Adresse ist hier als HEX-Wert dargestellt. Kann beliebig geändert werden.  
50 // Der Empfänger muss dieselbe Adresse haben.  
51 // Bedeutung von LL am Ende: LL is the suffix for long-long,  
52 // which is 64-bit on most C/C++ implementations.  
53 // So 0LL is a 64-bit literal with the value of 0.  
54 struct MyData  
55 {  
56     int counter;  
57     float temperature;  
58     float humidity;  
59     float altitude;  
60     float pressure;  
61 };  
62 MyData data;  
63  
64 WiFiClient client;  
65  
66 void setup()  
67 {  
68     Serial.begin(115200);  
69     radio.begin();  
70  
71     Serial.println("Receiver Started....");  
72     Serial.print("Connecting to ");  
73     Serial.println(ssid);  
74     Serial.println();  
75     WiFi.begin(ssid, password);  
76     while (WiFi.status() != WL_CONNECTED)  
77     {  
78         delay(500);  
79         Serial.print(".");  
80     }  
81     Serial.println("");  
82     Serial.println("WiFi connected");  
83  
84     radio.openReadingPipe(0, address); //Setting the address at which we will  
85                                     //receive the data  
86     radio.setPALevel(RF24_PA_MIN); //You can set this as minimum or maximum  
87                                     //depending on the distance between the transmitter and receiver.  
88     radio.startListening(); //This sets the module as receiver
```



```
86 }
88 int recvData()
89 {
90     if ( radio.available() )
91     {
92         radio.read(&data, sizeof(MyData));
93         return 1;
94     }
95     return 0;
96 }
98
99 void loop()
100 {
101     if(recvData())
102     {
103
104         Serial.println("Data Received:");
105         Serial.print("Packet No. = ");
106         Serial.println(data.counter);
107
108         Serial.print("Temperature = ");
109         Serial.print(data.temperature);
110         Serial.println("°C");
111
112         Serial.print("Pressure = ");
113         Serial.print(data.pressure);
114         Serial.println("hPa");
115
116         Serial.print("Approx. Altitude = ");
117         Serial.print(data.altitude);
118         Serial.println("m");
119
120         Serial.print("Humidity = ");
121         Serial.print(data.humidity);
122         Serial.println("%");
123
124         Serial.println();
125
126         if (client.connect(server, 80))
127         {
128             String postStr = apiKey;
129             postStr += "&field1=";
130             postStr += String(data.temperature);
131             postStr += "&field2=";
132             postStr += String(data.pressure);
133             postStr += "&field3=";
134             postStr += String(data.altitude);
135             postStr += "&field4=";
136             postStr += String(data.humidity);
137             postStr += "\r\n\r\n\r\n\r\n";
138
139             client.print("POST /update HTTP/1.1\n");
140             client.print("Host: api.thingspeak.com\n");
141             client.print("Connection: close\n");
142             client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
143             client.print("Content-Type: application/x-www-form-urlencoded\n");
144             client.print("Content-Length: ");
145             client.print(postStr.length());
146             client.print("\n\n");
147             client.print(postStr);
148             delay(1000);
149             Serial.println("Data Sent to Server");
```



```
150 }  
151     client.stop();  
152 }  
}
```

Listing 33: [NRF24L01ESP32GatewayThingspeak.ino](#)

Sensorwerte werden gesendet, empfangen und an den Thingspeak-Dienst übermittelt:

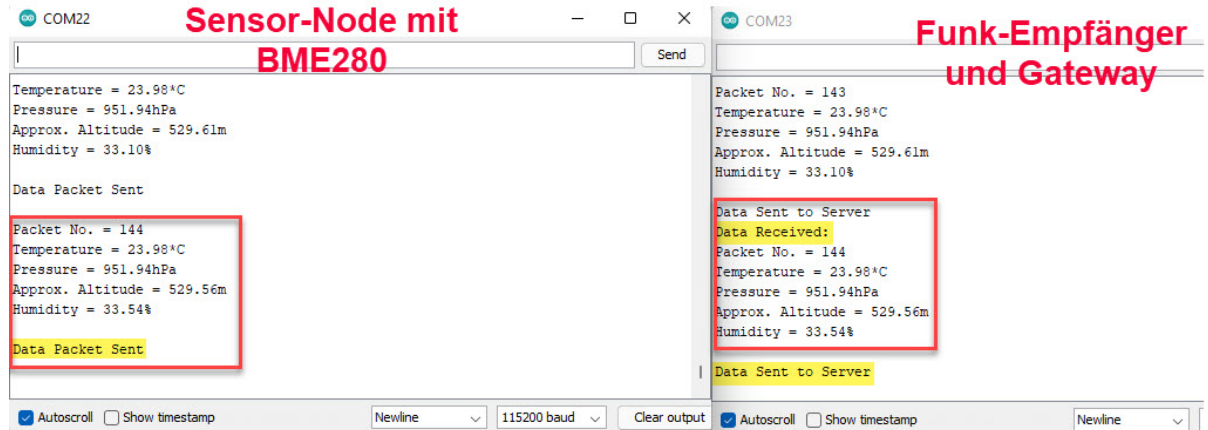


Abbildung 75: Die gesendeten und empfangenen Sensorwerte können im Serial Monitor beobachtet werden. Das Senden an den Thingspeak-Server wird auf dem Gateway quittiert.

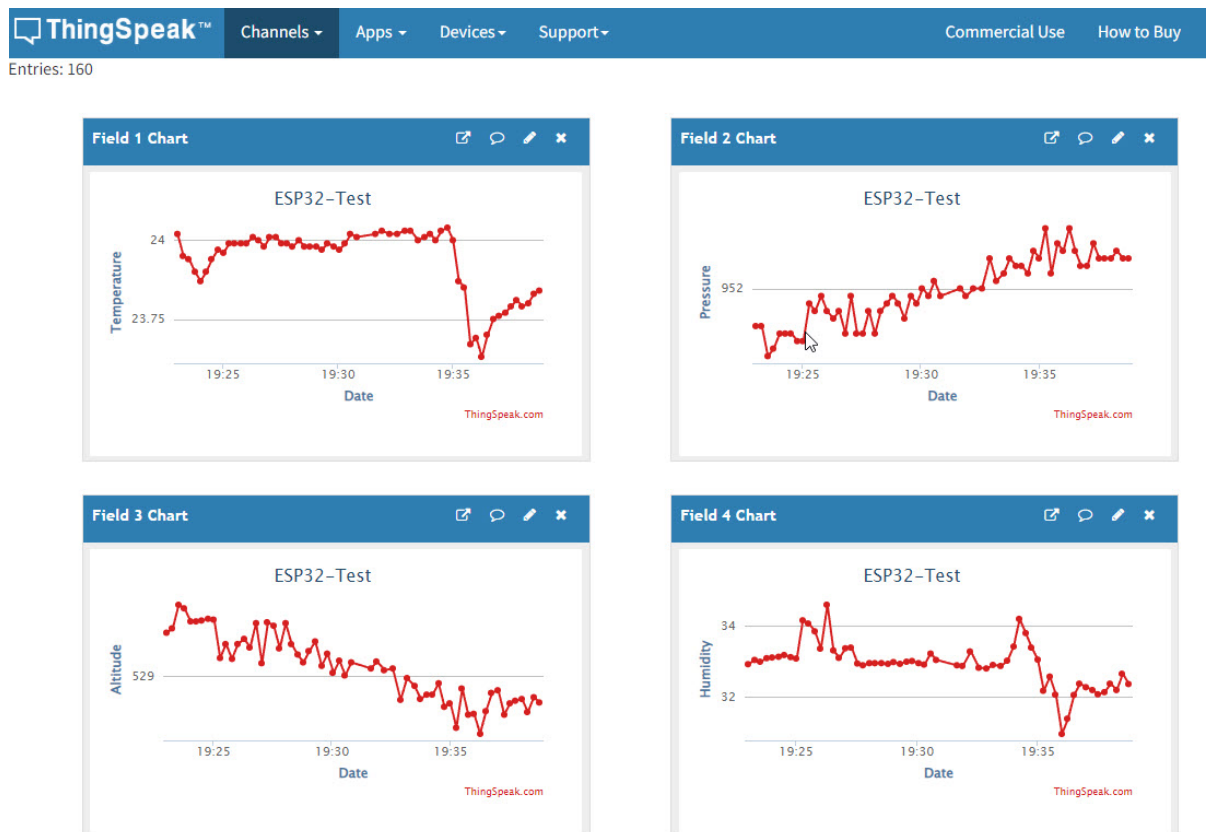


Abbildung 76: ... und hier die grafisch aufbereiteten Sensorwerte auf thingspeak.com



8.29 NI1000/PT1000 - OpAmp LM358

to do

8.30 ESP32 als Webserver

Der folgende Sketch publiziert die Sensordaten des BME280 auf einer Webseite. Der ESP32 fungiert dabei selber als Webserver. Für diesen Sketch benötigen Sie ein WLAN mit bekannter SSID und Passwort. Diese Angaben müssen Sie im Code anpassen (Zeilen: 28 und 29). Aus dem Serial Monitor können Sie die IP-Adresse entnehmen und dann mit dem Browser (z. Bsp. mit Smartphone oder PC) darauf zugreifen.

Folgende Sensordaten werden angezeigt:

ESP32 with BME280

MEASUREMENT	VALUE
Temp. Celsius	22.44 °C
Temp. Fahrenheit	72.39 °F
Pressure	970.33 hPa
Approx. Altitude	540.98 m
Humidity	31.08 %

Abbildung 77: Sensorwerte des BME280 auf Webseite. Der ESP32 dient als Webserver.

Source-Code [BME280WebServer.ino](#):

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/BME280WebServer/BME280WebServer.ino>

```
1  /*
   * Function:  Read Pressure and Temperatur from BME280 and display values on
               webserver
3  * MCU:      Olimex ESP32-DevKit-LiPo
   * File:      BME280WebServer.ino
5  * Libraries: BME280 from Adafruit, Adafruit GFX und Adafruit SSD1306 Library
               wird verwendet, WiFi
   * Connections: SCL and SDA from BME280 to SCL und SDA from ESP32.
7  *           Activate Power with Jumper "BME280/388 3V3". Place Jumper over
               white field.
   *
9  * WiFi:     Your SSID with Passphrase, WPA2
   * Output:    Browser, Serial Monitor with 115200baud
11 * Author:    Thomas Jäggi
   * Date:      07.12.2021
13 * Modified from Example-Sketch
   */
15
16 // Load Wi-Fi library
17 #include <WiFi.h>
   #include <Wire.h>
19 #include <Adafruit_BME280.h>
   #include <Adafruit_Sensor.h>
21
```




```
#define SEALEVELPRESSURE_HPA (1035) // see:
23 // https://meteologix.com/observations/pressure-qnh.html
25 Adafruit_BME280 bme; // I2C
27 // Replace with your network credentials
const char* ssid      = "SSID";
29 const char* password = "Passphrase";
31 // Set web server port number to 80
WiFiServer server(80);
33
// Variable to store the HTTP request
35 String header;
37 // Current time
unsigned long currentTime = millis();
39 // Previous time
unsigned long previousTime = 0;
41 // Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
43
void setup() {
45   Serial.begin(115200);
   bool status;
47
   // default settings
49   // (you can also pass in a Wire library object like &Wire2)
   //status = bme.begin();
51   if (!bme.begin(0x76)) {
       Serial.println("Could not find a valid BME280 sensor, check wiring!");
53       while (1);
   }
55
   // Connect to Wi-Fi network with SSID and password
57   Serial.print("Connecting to ");
   Serial.println(ssid);
59   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
61       delay(500);
       Serial.print(".");
63   }
   // Print local IP address and start web server
65   Serial.println("");
   Serial.println("WiFi connected.");
67   Serial.println("IP address: ");
   Serial.println(WiFi.localIP());
69   server.begin();
71 }
void loop(){
73   WiFiClient client = server.available(); // Listen for incoming clients
75   if (client) { // If a new client connects,
       currentTime = millis();
       previousTime = currentTime;
77       Serial.println("New Client."); // print a message out in the serial
       port
79       String currentLine = ""; // make a String to hold incoming data
       from the client
       while (client.connected() && currentTime - previousTime <= timeoutTime) { //
       loop while the client's connected
           currentTime = millis();
81           if (client.available()) { // if there's bytes to read from the
```



```
client,
83     char c = client.read();           // read a byte, then
      Serial.write(c);                 // print it out the serial monitor
85     header += c;
      if (c == '\n') {                 // if the byte is a newline character
87         // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
89         if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK
          )
91         // and a content-type so the client knows what's coming, then a blank
line:
          client.println("HTTP/1.1 200 OK");
93         client.println("Content-type:text/html");
          client.println("Connection: close");
95         client.println();

97         // Display the HTML web page
          client.println("<!DOCTYPE html><html>");
99         client.println("<head><meta name=\"viewport\" content=\"width=device-
width, initial-scale=1\">");
          client.println("<link rel=\"icon\" href=\"data:,\>");
101         // CSS to style the table
          client.println("<style>body { text-align: center; font-family: \"
Trebuchet MS\", Arial;}");
103         client.println("table { border-collapse: collapse; width:35%; margin-
left:auto; margin-right:auto; }");
          client.println("th { padding: 12px; background-color: #0043af; color:
white; }");
105         client.println("tr { border: 1px solid #ddd; padding: 12px; }");
          client.println("tr:hover { background-color: #bcbcbc; }");
107         client.println("td { border: none; padding: 12px; }");
          client.println(".sensor { color:white; font-weight: bold; background-
color: #bcbcbc; padding: 1px; }");
109
          // Web Page Heading
111         client.println("</style></head><body><h1>ESP32 with BME280</h1>");
          client.println("<table><tr><th>MEASUREMENT</th><th>VALUE</th></tr>");
113         client.println("<tr><td>Temp. Celsius</td><td><span class=\"sensor\">");
;
          client.println(bme.readTemperature());
115         client.println(" *C</span></td></tr>");
          client.println("<tr><td>Temp. Fahrenheit</td><td><span class=\"sensor\">");
>");
117         client.println(1.8 * bme.readTemperature() + 32);
          client.println(" *F</span></td></tr>");
119         client.println("<tr><td>Pressure</td><td><span class=\"sensor\">");
          client.println(bme.readPressure() / 100.0F);
121         client.println(" hPa</span></td></tr>");
          client.println("<tr><td>Approx. Altitude</td><td><span class=\"sensor\">");
>");
123         client.println(bme.readAltitude(SEALEVELPRESSURE_HPA));
          client.println(" m</span></td></tr>");
125         client.println("<tr><td>Humidity</td><td><span class=\"sensor\">");
          client.println(bme.readHumidity());
127         client.println(" %</span></td></tr>");
          client.println("</body></html>");
129
          // The HTTP response ends with another blank line
131         client.println();
          // Break out of the while loop
133         break;
      } else { // if you got a newline, then clear currentLine
135         currentLine = "";
```



```
137     } else if (c != '\r') { // if you got anything else but a carriage return
character,
    currentLine += c;      // add it to the end of the currentLine
139     }
141 }
// Clear the header variable
143 header = "";
// Close the connection
145 client.stop();
Serial.println("Client disconnected.");
147 Serial.println("");
149 }
```

Listing 34: [BME280WebServer.ino](#)

Die IP-Adresse erscheint im Serial Monitor nach einem Reset des Microcontrollers. Voraussetzung: Die SSID und das Passwort im Code sind korrekt und ein DHCP-Server liefert eine IP-Adresse.

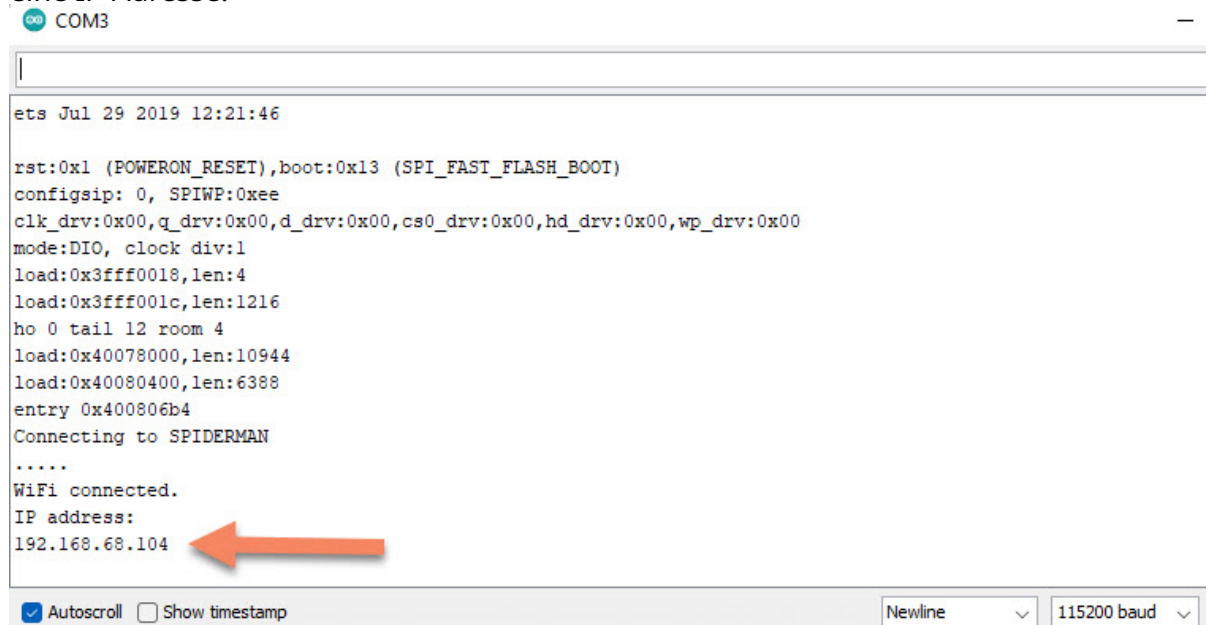


Abbildung 78: im Serial Monitor kann die IP-Adresse entnommen werden.
Browserverbindungen werden im Serial Monitor mit Details dokumentiert:



Abbildung 79: Hier eine Clientverbindung mit dem Chrome-Browser
Zugriff mit Browser auf den Webservice des ESP32:

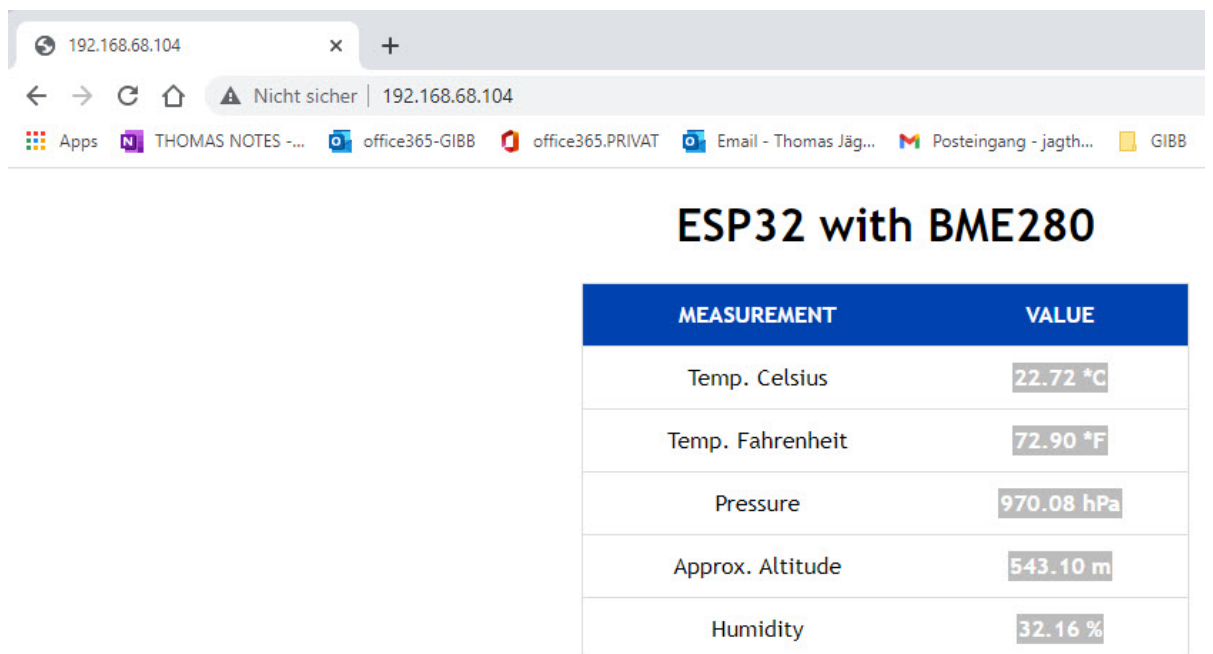


Abbildung 80: *http*-Verbindung auf ESP32 mit Chrome-Browser. *https* haben wir in diesem Beispiel nicht implementiert.

9 Der M5StickC-Plus als Oszilloskop

Für das Experimentieren mit dem Board ist es ab und zu hilfreich, Signale visualisieren zu können. Da wir auf dem Board zwei Microcontroller zur Verfügung haben, können wir einen als Oszilloskop verwenden. Dabei verbinden wir das zu visualisierende Signal mit einem GPIO-Pin des M5StickC-Plus (oranger Microcontroller).

Der M5StickC-Plus benötigt eine eigene Library. Diese kann einfach unter `SKETCH/INCLUDE LIBRARY/MANAGE LIBRARIES` installiert werden. Suchen Sie nach *M5StickCPlus* und installie-



ren Sie diese.

Die zu visualisierenden Signale können mit dem GPIO26 und GPIO36 des M5StickCPlus verbunden werden. Wir können also bei Bedarf zwei Signale gleichzeitig darstellen. Achten Sie darauf, dass der Signalpegel 3.3V nicht übersteigt, sonst geht der Microcontroller kaputt!

Neben der Library benötigen wir noch den Source-Code für die Darstellung der Signale:

Diesen finden Sie [hier](#). Da dieser 500 Zeilen umfasst, ist er hier nicht abgebildet.

<https://gitlab.com/tjaeggli/iot-education-board/-/blob/master/ArduinoDemoSketches/M5StickCPlusOszil/M5StickCPlusOszil.ino>

Das Oszilloskop hat natürlich technische Limiten und kann nicht mit einem professionellen Gerät verglichen werden. Für qualitative Messungen von einfachen Signalen mit tiefen Frequenzen, kann es zu Schulungszwecken verwendet werden.

Beispiele:

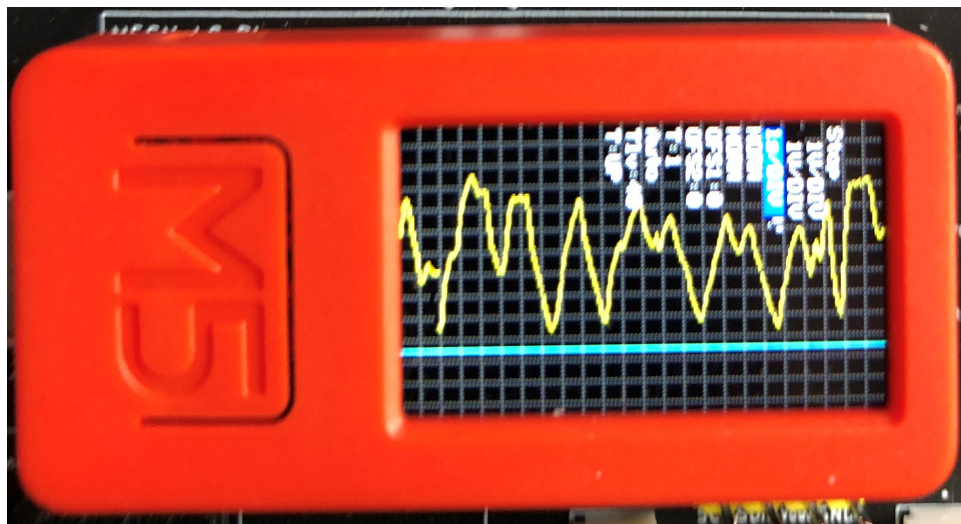


Abbildung 81: analoges Signal durch Drehen am Potentiometers

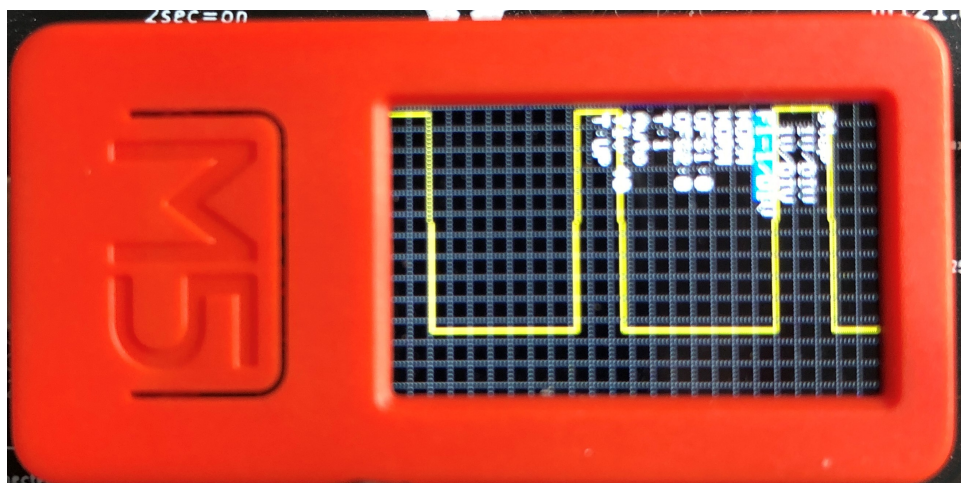


Abbildung 82: digitales Signal - Rechteck-Signal

Und hier noch ein kleines [Video](#) eines PWM-Signals mit variablem Duty-Cycle, wie es beim Dimmen einer LED vorkommt.



10 Demo-Sketches M5StickC Plus

Übrigens: Der M5StickC Plus Microcontroller hat einen eingebauten Li-Po-Akku (120mAh), kann vom Board abgezogen und mobil verwendet werden.

10.1 Watch

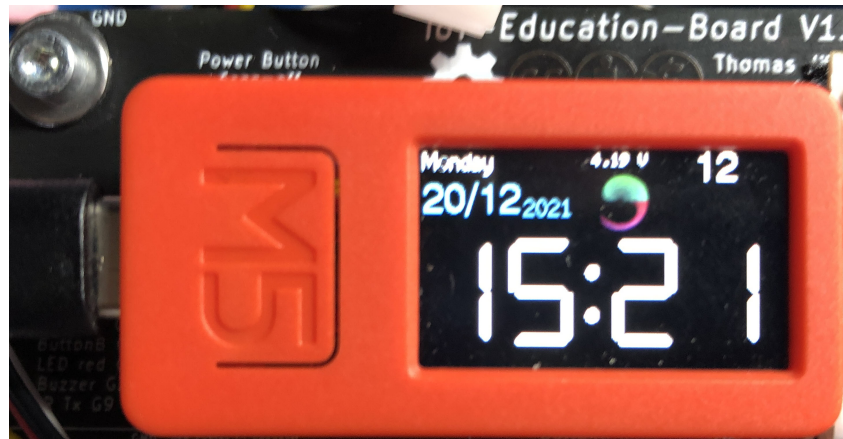


Abbildung 83: Display Watch

Source-Code [M5StickCPlusClock](#)

<https://gitlab.com/tjaeggi/iot-education-board/-/tree/master/ArduinoDemoSketches/M5StickCPlusClock>



10.2 Stopwatch

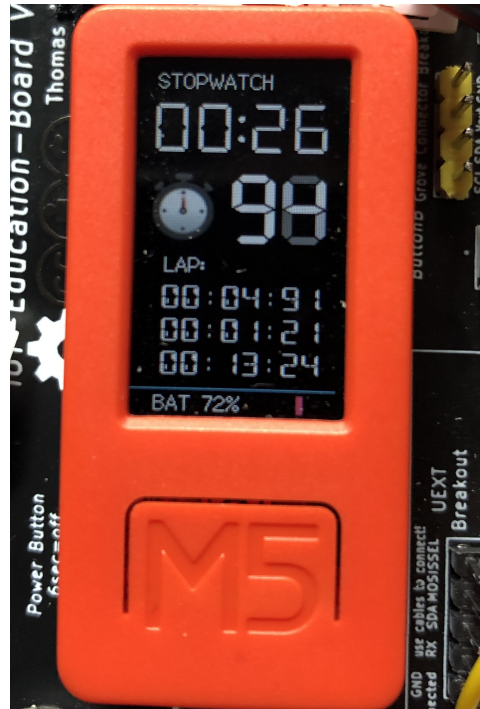


Abbildung 84: Display Stopwatch

Source-Code [M5StickCPlusStopWatch](#)

<https://gitlab.com/tjaeggi/iot-education-board/-/tree/master/ArduinoDemoSketches/M5StickCPlusStopWatch>

10.3 Tetris Game

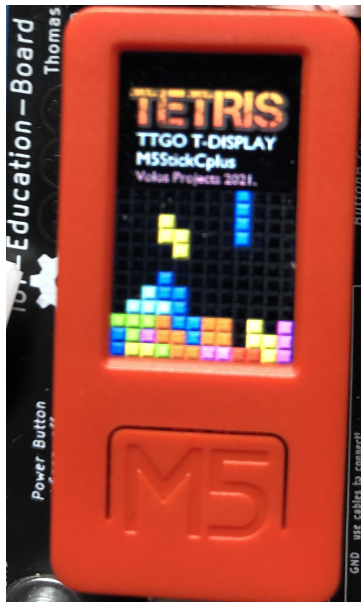


Abbildung 85: Tetris Game



Abbildung 86: Tetris Game
in Action

Source-Code [Tetris Game](#)

<https://gitlab.com/tjaeggi/iot-education-board/-/tree/master/ArduinoDemoSketches/M5StickCPlusTetrisGame>



Bauteil	Hex-Adresse
OLED Display	0x3C
BMP280	0x76
MPU9250	0x76
	0x68
BH1750	0x23
ADS1115-ADC	0x48
	0x00
BMP388	0x77
RTC	0x68

11.2 GPIO-Map ESP32-Devkit-Lipo

<https://www.olimex.com/Products/IoT/ESP32/ESP32-DevKit-LiPo/resources/ESP32-DevKit-Lipo-GPIOs.png>



<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>

Die beiden Layouts sind identisch.

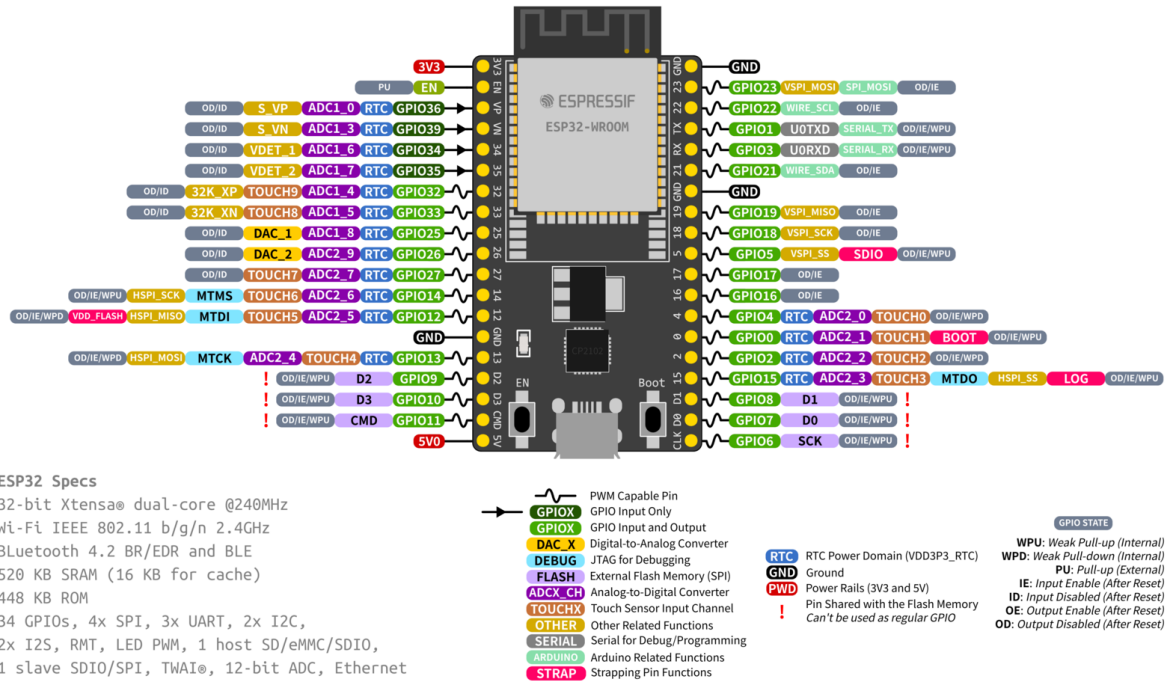


Abbildung 88: Pin Layout ESP32-DevKitC

11.2.1 Funktionen und verwendbare GPIOs

Grün markierte GPIOs können verwendet werden.



GPIO	Input	Output	Bemerkungen
0	pulled up	OK	gibt PWM-Signal beim Boot aus
1	TX PIN	OK	Debug Output beim Boot
2	OK	OK	verbunden mit Board-LED
3	OK	RX Pin	HIGH beim Boot
4	OK	OK	
5	OK	OK	gibt PWM-Signal beim Boot aus
6	X	X	verbunden mit internem SPI-Flash
7	X	X	verbunden mit internem SPI-Flash
8	X	X	verbunden mit internem SPI-Flash
9	X	X	verbunden mit internem SPI-Flash
10	X	X	verbunden mit internem SPI-Flash
11	X	X	verbunden mit internem SPI-Flash

Abbildung 89: Funktionen der GPIO 1-11



12	OK	OK	bei Boot-Fehler auf HIGH gesetzt
13	OK	OK	
14	OK	OK	gibt PWM-Signal beim Boot aus
15	OK	OK	gibt PWM-Signal beim Boot aus
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
20	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
24	OK	OK	
25	OK	OK	

Abbildung 90: Funktionen der GPIO 12-25



26	OK	OK	
27	OK	OK	
28	OK	OK	
29	OK	OK	
30	OK	OK	
31	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		nur Input

Abbildung 91: Funktionen der GPIO 26-34

Interner SPI-Flash des ESP-WROOM-32 Die GPIOs 6 bis 11 können nicht genutzt werden, obwohl manche Boards sie nach aussen führen. Grund ist, dass sie für den internen SPI reserviert sind.

Nur als Input-Pins verwendbar Die GPIOs 34 bis 39 sind keine richtigen GPIOs, da sie nur als Input funktionieren (GPI). Sie haben keine internen Pull-up- oder Pull-down-Widerstände. Folgende GPIOs sind also nur Input-GPIOs:

- › GPIO34
- › GPIO35
- › GPIO36
- › GPIO39

PWM Der ESP32-LED-PWM-Controller hat 16 unabhängige Kanäle, die mit unterschiedlichen Eigenschaften konfiguriert werden können. Alle richtigen Output-Pins können PWM generieren (also nicht die GPIOs 34 bis 39).

Interrupts Alle GPIOs können für Interrupts konfiguriert werden.

Touch-GPIOs Der ESP32 verfügt über zehn interne kapazitive Berührungssensoren. Diese können Schwankungen von elektrischen Ladungen registrieren, die z. B. durch Kontakt mit der menschlichen Haut entstehen (Berühren der GPIOs mit einem Finger). Anwendungsfall ist u. a. die Integration in kapazitive Touchpads, wo sie mechanische Tasten



ersetzen können. Über die kapazitiven Touch-Pins kann der ESP32 auch aus dem Tiefschlaf erweckt werden. Die Touch-Pins sind folgendermaßen zugeordnet:

- › T0 (GPIO4)
- › T1 (GPIO0)
- › T2 (GPIO2)
- › T3 (GPIO15)
- › T4 (GPIO13)
- › T5 (GPIO12)
- › T6 (GPIO14)
- › T7 (GPIO27)
- › T8 (GPIO33)
- › T9 (GPIO32)

Analog to Digital Converter (ADC) Der ESP32 hat 18 ADC-Input-Kanäle, die mit 12-Bit-Auflösung operieren. Das bedeutet, dass die Kanäle analoge Messwerte im Bereich von 0 bis 4095 aufnehmen können, wobei 0 0 V und 4095 3,3 V entspricht. Auflösung und ADC-Bereich können softwareseitig eingestellt werden.

Wenn exakte Spannungsmessungen gemacht werden sollen, empfiehlt sich der Einsatz eines externen ADC. z. B. ein ADS 111x.

- › ADC1_CH0 (GPIO36)
- › ADC1_CH1 (GPIO37)
- › ADC1_CH2 (GPIO38)
- › ADC1_CH3 (GPIO39)
- › ADC1_CH4 (GPIO32)
- › ADC1_CH5 (GPIO33)
- › ADC1_CH6 (GPIO34)
- › ADC1_CH7 (GPIO35)
- › ADC2_CH0 (GPIO4)
- › ADC2_CH1 (GPIO0)
- › ADC2_CH2 (GPIO2)
- › ADC2_CH3 (GPIO15)
- › ADC2_CH4 (GPIO13)
- › ADC2_CH5 (GPIO12)
- › ADC2_CH6 (GPIO14)
- › ADC2_CH7 (GPIO27)
- › ADC2_CH8 (GPIO25)
- › ADC2_CH9 (GPIO26)

Digital to Analog Converter (DAC) Es gibt zwei DAC-Kanäle mit einer 8-Bit-Auflösung. Sie konvertieren digitale Signale oder analoge Werte in analoge Signale. Die Kanäle sind:

- › DAC1 (GPIO25)
- › DAC2 (GPIO26)

RTC-GPIOs RTC steht für Real Time Clock. Der ESP32 bietet RTC-GPIO-Unterstützung. Das RTC-System selbst benötigt extrem wenig Strom und ist verantwortlich für die verschiedenen Energiesparmodi, die bis zum Tiefschlaf reichen. Mit den RTC-GPIOs kann der ESP32 geweckt werden. Die folgenden GPIOs können als externe Weckquelle verwendet werden.



- › RTC_GPIO0 (GPIO36)
- › RTC_GPIO3 (GPIO39)
- › RTC_GPIO4 (GPIO34)
- › RTC_GPIO5 (GPIO35)
- › RTC_GPIO6 (GPIO25)
- › RTC_GPIO7 (GPIO26)
- › RTC_GPIO8 (GPIO33)
- › RTC_GPIO9 (GPIO32)
- › RTC_GPIO10 (GPIO4)
- › RTC_GPIO11 (GPIO0)
- › RTC_GPIO12 (GPIO2)
- › RTC_GPIO13 (GPIO15)
- › RTC_GPIO14 (GPIO13)
- › RTC_GPIO15 (GPIO12)
- › RTC_GPIO16 (GPIO14)
- › RTC_GPIO17 (GPIO27)

I²C Der ESP32 hat zwei I²C-Kanäle. Dabei kann jeder Pin als SDA oder SCL fungieren. Die Arduino-IDE hat die Pins für einen Kanal vorbelegt (SDA: GPIO21, SCL: GPIO22).

SPI Es stehen zwei SPI-Kanäle zur Verfügung. Voreinstellung:

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO23	GPIO19	GPIO18	GPIO5
HSPI	GPIO13	GPIO12	GPIO14	GPIO15

Abbildung 92: GPIOs mit SPI

11.3 GPIO-Map M5StickC Plus

Der M5StickC-Plus beinhaltet bereits viele Sensoren und Aktoren, so dass die frei verfügbaren Pins beschränkt sind. Siehe dazu Abbildung 95. Trotzdem können über den Grove-Anschluss weitere Elemente angeschlossen werden. Eine Übersicht verfügbarer Module finden Sie [hier](https://shop.m5stack.com/) und [hier](https://www.bastelgarage.ch/index.php?route=product/search&search=m5stick).



RED LED & IR Transmitter & BUTTON A & BUTTON B

ESP32	GPIO10	GPIO9	GPIO37	GPIO39	GPIO2
RED LED	LED Pin				
IR Transmitter		Transmitter Pin			
BUTTON A			Button Pin		
BUTTON B				Button Pin	
Buzzer					Buzzer Pin

TFT LCD

Driver IC: ST7789v2

Resolution: 135 * 240

ESP32	GPIO15	GPIO13	GPIO23	GPIO18	GPIO5
TFT LCD	TFT_MOSI	TFT_CLK	TFT_DC	TFT_RST	TFT_CS

Abbildung 93: Pinmap M5StickC Plus

GROVE PORT

ESP32	GPIO33	GPIO32	5V	GND
GROVE port	SCL	SDA	5V	GND

MIC (SPM1423)

ESP32	GPIO0	GPIO34
MICPHONE	CLK	DATA

6-Axis posture sensor (MPU6886) & power management IC (AXP192)

ESP32	GPIO22	GPIO21
6-Axis IMU sensor	SCL	SDA
Power management IC	SCL	SDA

AXP192

Microphone	RTC	TFT backlight	TFT IC	ESP32/3.3V MPU6886	5V GROVE
LDOio0	LDO1	LDO2	LDO3	DC-DC1	IPSOUT

Abbildung 94: Pinmap M5StickC Plus



Power structure block diagram

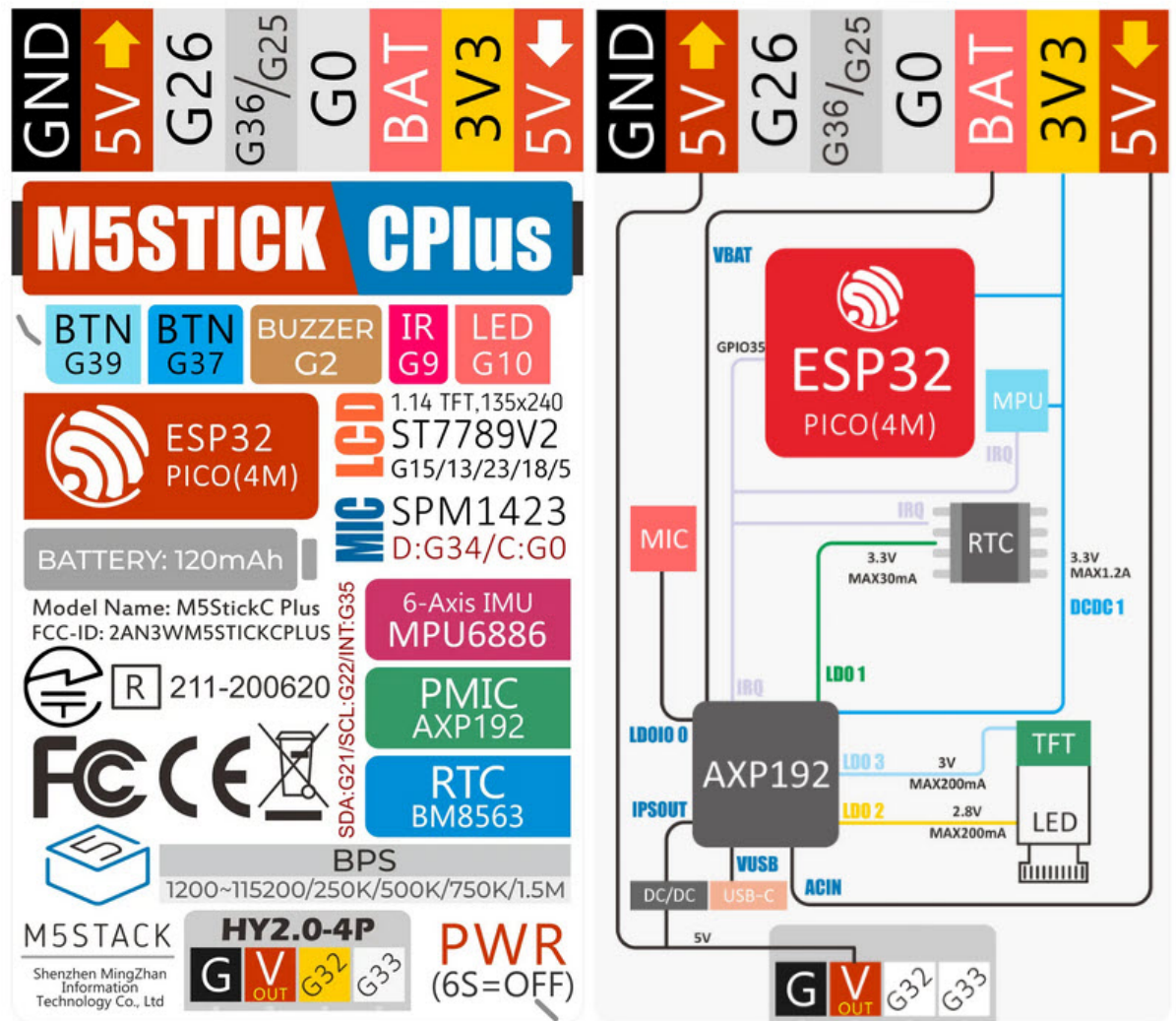


Abbildung 95: Pinmap M5StickC Plus

Siehe auch [M5Stick-Dokument](#).

<https://gitlab.com/tjaeggi/iot-education-board/-/blob/master/DatasheetsComponents/ESP32M5StickCPLUS.pdf>