Eigener Text mit ChatGPT vereinfacht

ESP32 - Analoge Messwerte auslesen

In diesem Tutorial lernst du, wie du mit einem ESP32-Mikrocontroller und MicroPython analoge Werte lesen kannst. Als Beispiel verwenden wir ein Potentiometer, um die analogen Werte zu messen.

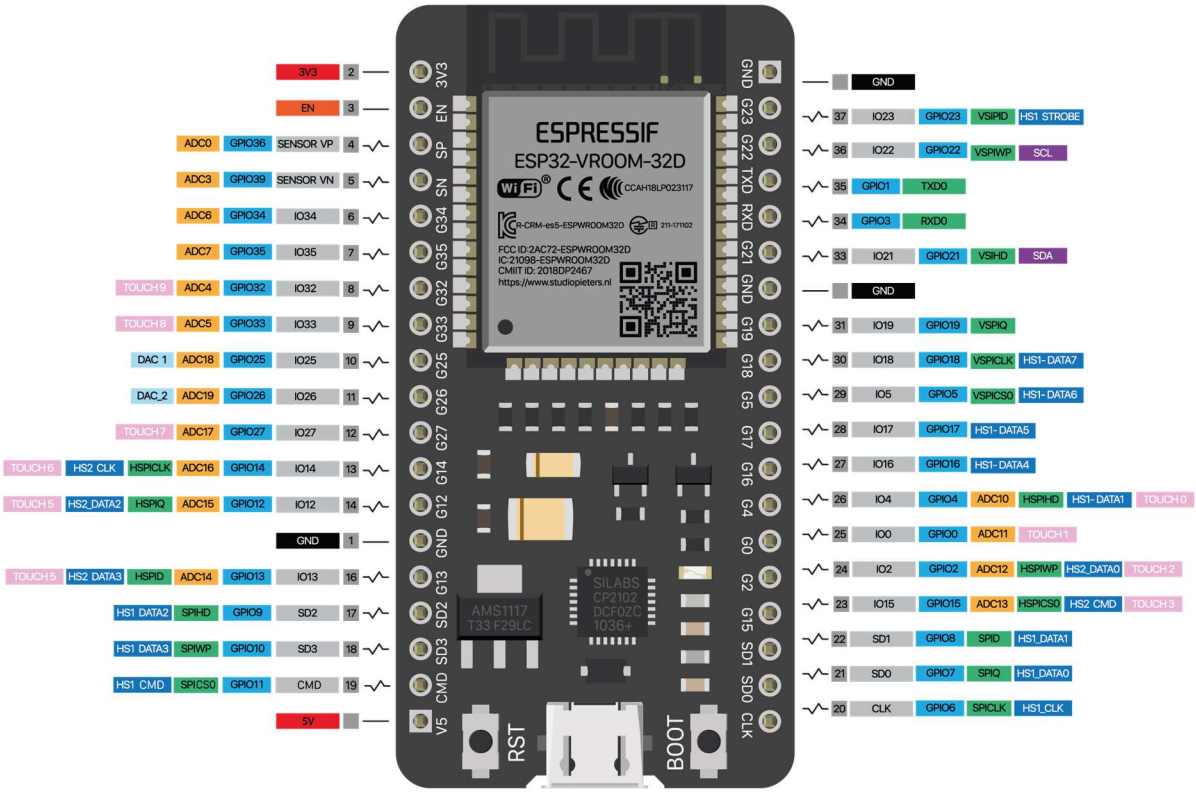
Voraussetzungen

1. **MicroPython**: Die MicroPython-Firmware muss auf deinem ESP32 installiert sein. Diese Firmware ermöglicht das Programmieren des ESP32 mit der Programmiersprache Python.
2. **Entwicklungsumgebung**: Du brauchst eine Entwicklungsumgebung (IDE) auf deinem Computer, um den Code auf den ESP32 zu laden. Eine beliebte IDE für MicroPython ist die *Thonny IDE*.

Was sind ADC-Pins beim ESP32?

ADC steht für *Analog-Digital-Converter* (Analog-Digital-Umwandler). ADC-Pins sind spezielle Pins am ESP32, die analoge Spannungen messen und diese in digitale Werte umwandeln. Diese digitalen Werte kannst du dann im Code weiterverarbeiten.

Beim ESP32 gibt es mehrere sogenannte GPIO-Pins (*General Purpose Input Output*), die als ADC-Pins verwendet werden können. Hier eine Liste der ADC-fähigen Pins: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36 und 39.



https://www.studiopieters.nl/esp32-pinout/

https://www.studiopieters.nl/esp32-pinout/ (beschnitten)

Messwerte und Auflösung

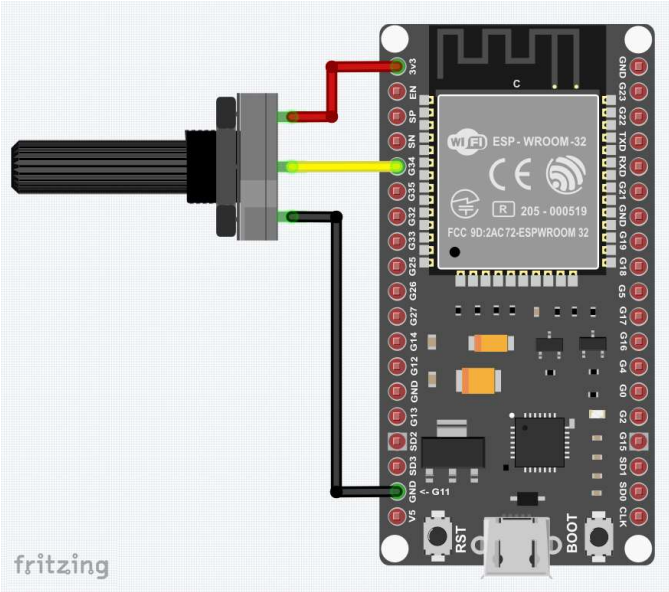
Die ADC-Pins des ESP32 haben standardmäßig eine *12-Bit-Auflösung*. Das bedeutet:

- Die Pins lesen Spannungen zwischen **0 und 3,3 Volt**.
- Diese Spannungen werden in digitale Werte zwischen **0 und 4095** umgewandelt. Ein Wert von 0 entspricht dabei 0 Volt, und ein Wert von 4095 entspricht 3,3 Volt.

Du kannst die Auflösung auch im Programmcode anpassen. Zum Beispiel kannst du eine 10-Bit-Auflösung einstellen, dann liegen die möglichen Werte zwischen 0 und 1023.

Schaltung – ESP32

Verbinde das Potentiometer mit dem ESP32, wie im folgenden Schaltplan gezeigt.



In diesem Beispiel nutzen wir GPIO 34, um analoge Werte vom Potentiometer zu lesen. Du kannst aber auch jeden anderen GPIO verwenden, der ADC unterstützt.

Das Beispielprogramm

Das folgende Programm liest analoge Werte vom GPIO 34 des ESP32 aus:

```
from machine import Pin, ADC
from time import sleep

pot = ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)       # Voller Spannungsbereich: 0 - 3,3 V

while True:
  pot_value = pot.read()
  print(pot_value)
  sleep(0.1)
```

Und so funktioniert der Code

Um analoge Eingänge lesen zu können, musst Du die Pin- und ADC-Klasse aus der Bibliothek machine importieren. Außerdem verwenden wir die sleep-Methode aus der time-Bibliothek, um eine Pause zwischen den Messungen einzubauen.

```
from machine import Pin, ADC
from time import sleep
```

Nun erstellen wir ein „ADC-Objekt“ mit dem Namen pot und verbinden es mit GPIO 34. Der Name pot ist frei wählbar.

```
pot = ADC(Pin(34))
```

Die folgende Zeile stellt sicher, dass der gesamte Spannungsbereich von 0 bis 3,3 V genutzt wird:

```
pot.atten(ADC.ATTN_11DB)
```

Das bedeutet, dass Spannungen zwischen 0 und 3,3 V gemessen und konvertiert werden können. Diese Einstellung erfolgt mit der Methode `atten()` und dem Argument `ADC.ATTN_11DB`. Die Methode `atten()` akzeptiert folgende Einstellungen:

- ADC.ATTN_0DB (Spannungsbereich : 0 – 1,2V)
- ADC.ATTN_2_5DB (Spannungsbereich: 0 – 1,5V)
- ADC.ATTN_6DB (Spannungsbereich: 0 – 2,0V)
- ADC.ATTN_11DB (Spannungsbereich: 0 – 3,3V)

In der `while`-Schleife wird dann der Spannungswert des Potentiometers ausgelesen und in der Variablen `pot_value` gespeichert. Dafür wird die Methode `read()` auf das `pot-Objekt` angewendet:

```
pot_value = pot.read()
```

Anschließend wird der Wert im Terminal ausgegeben:

```
print(pot_value)
```

Schließlich erfolgt eine Pause von 100 ms, um die Messungen nicht zu schnell hintereinander auszuführen:

```
sleep(0.1)
```

Ergebnis

Wenn Du das Potentiometer über seinen gesamten Bereich drehst, erhältst Du Messwerte von 0 bis 4095. Diese Werte kommen daher, dass die ADC-Pins standardmäßig eine 12-Bit-Auflösung haben.

Mit der `width()`-Methode kannst Du die Auflösung anpassen:

```
ADC.width(bit)
```

Die `width`-Methode akzeptiert folgende Argumente:

- `ADC.WIDTH_9BIT` – Bereich 0 bis 511
- `ADC.WIDTH_10BIT` – Bereich 0 bis 1023
- `ADC.WIDTH_11BIT` – Bereich 0 bis 2047
- `ADC.WIDTH_12BIT` – Bereich 0 bis 4095

Beispiel:

```
ADC.width(ADC.WIDTH_12BIT)
```

Zusammenfassung:

- Um einen analogen Wert zu lesen, musst Du die `ADC`-Klasse importieren.
- Ein `ADC`-Objekt wird mit `ADC(Pin(GPIO))` erstellt, wobei `GPIO` die Nummer des Pins ist, der ausgelesen wird.
- Um den analogen Wert auszulesen, verwendest Du die Methode `read()` auf das `ADC`-Objekt