# Python & MicroPython: Programmieren lernen für Einsteiger

## Die analoge Welt steuern – DACs und PWM

Teil 8

**Pulsweitenmodulation (PWM) und Digital-Analog-Wandlung (DAC) sind zwei wichtige Technologien, die es erlauben, analoge Signale zu erzeugen bzw. zu simulieren. Sie bilden in gewisser Weise das Gegenstück zu den in Teil 5 dieser Serie behandelten Analog-Digital-Wandler (ADCs). Mithilfe einfacher Python-Programme lassen sich beide Technologien problemlos z. B. für einen Raspberry Pi ein- bzw. umsetzen.**

### Pulsweitenmodulation (PWM)

Die Pulsweitenmodulation (PWM) ist eine Methode, um die Leistung, Spannung oder den Strom an einer elektronischen Last zu steuern. Dabei wird ein digitales Signal erzeugt, das aus einer Reihe von Rechteckimpulsen mit konstanter Frequenz besteht, deren Einschaltzeit, also die Dauer des „High"-Zustands, variiert wird. Diese „on"-Zeit wird im Verhältnis zur Gesamtperiode des Signals betrachtet und als Tastverhältnis (Duty Cycle) bezeichnet. Das Tastverhältnis gibt an, wie lange das Signal in einer Periode eingeschaltet ist, und wird meist in Prozent angegeben. Ein Duty Cycle von 50 Prozent bedeutet also, dass das Signal innerhalb einer Periode genauso lange eingeschaltet wie ausgeschaltet ist (Bild 1). Durch die Veränderung des Tastverhältnisses kann die mittlere Leistung, die einer Last zugeführt wird, präzise reguliert werden. Die Grundfrequenz bleibt dabei konstant, während die Impulsbreite flexibel angepasst wird. PWM findet Anwendung in zahlreichen Bereichen, z. B. zur Steuerung von Motoren, LEDs oder in der Spannungsregelung.

## Python-PWM mit dem Raspberry Pi

In den letzten Artikeln wurde die Bibliothek GPIOZero bereits für verschiedene Anwendungen wie das Ansteuern von LEDs oder das Einlesen von Tastern verwendet. Die Bibliothek bietet auch die einfache Möglichkeit, PWM-Signale mit Python zu erzeugen. Das folgende Programm (PWM_at_GPIO_04.py) liefert ein pulsweitenmoduliertes Signal an GPIO 4 eines Raspberry Pi 5:

```
from gpiozero import PWMOutputDevice
import time

PWM_PIN = 4

pwm = PWMOutputDevice(PWM_PIN, initial_value=0, frequency=1000)

while True:
    for duty_cycle in range(0, 101, 1):
        pwm.value = duty_cycle / 100.0
        time.sleep(0.01)
    for duty_cycle in range(100, -1, -1):
        pwm.value = duty_cycle / 100.0
        time.sleep(0.01)
```

Wird am entsprechenden Pin ein LED-Modul aus der PAD-Serie oder eine einzelne LED (mit Vorwiderstand) angeschlossen, kann man beobachten, wie die LED ihre Helligkeit kontinuierlich verändert (Bild 2).

Das Programm erzeugt ein PWM-Signal, dessen Tastverhältnis kontinuierlich ansteigt und wieder abfällt. Zunächst werden hierfür die erforderlichen Bibliotheken importiert:

```
from gpiozero import PWMOutputDevice
import time
```

Die Klasse PWMOutputDevice dient zur Steuerung von PWM-Ausgängen. Danach wird der PWM-Pin definiert:

```
PWM_PIN = 4
```

Die Anweisung legt fest, welcher GPIO-Pin verwendet wird. Hier wird GPIO 04 gewählt. Es folgt die Initialisierung der PWM-Ausgabe:

```
pwm = PWMOutputDevice(PWM_PIN, initial_value=0, frequency=1000)
```

Der Befehl PWMOutputDevice erstellt ein PWM-Signal auf dem angegebenen Pin (PWM_PIN). Die Parameter haben die folgenden Funktionen:

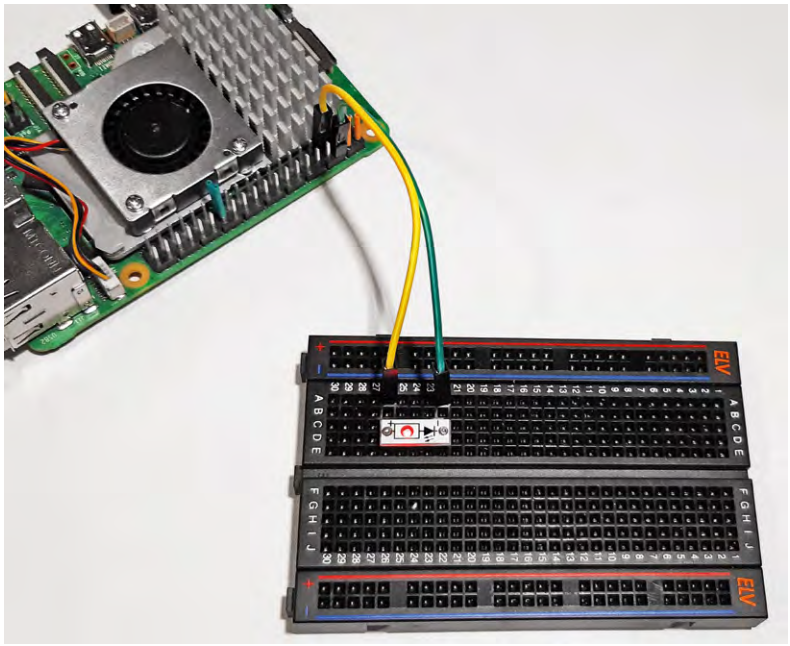

*Bild 2: PWM-gesteuerte LED am Raspberry Pi*

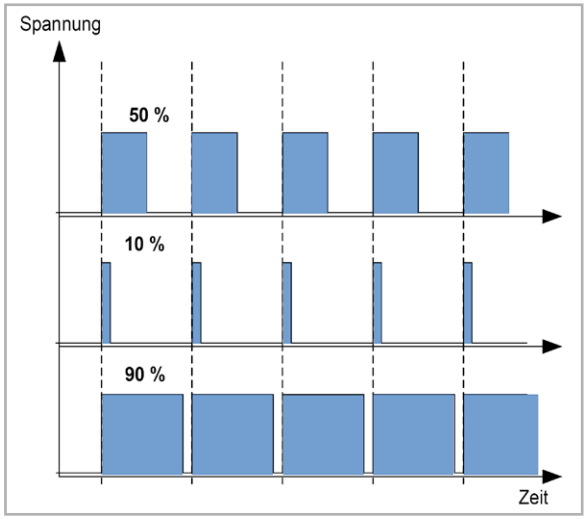


*Bild 1: PWM-Signale mit 50 %, 10 % und 90 % Duty Cycle*

- initial_value=0: Das PWM-Signal startet mit einem Duty Cycle von 0 % (Ausgang ist aus).
- frequency=1000: Die Frequenz des PWM-Signals wird auf 1000 Hz (= 1 kHz) festgelegt.

Es folgen die Schleifen zur Steuerung des Duty Cycles:

- Die Endlosschleife (while True:) stellt sicher, dass das Programm dauerhaft läuft, bis es manuell gestoppt wird.
- Die erste Schleife (for duty_cycle in range(0, 101, 1)) übernimmt die folgende Aufgabe:
  - Der Duty Cycle wird von 0 % auf 100 % in Schritten von 1 % erhöht.
  - Der Duty Cycle wird auf einen Wert zwischen 0,0 und 1,0 skaliert
  - Nach jeder Änderung des Duty Cycles wartet das Programm 10 Millisekunden, bevor es fortfährt. Dadurch entsteht ein weiches Ansteigen der LED-Helligkeit.
- Zweite Schleife (for duty_cycle in range(100, -1, -1)) reduziert die LED Helligkeit:
  - Der Duty Cycle wird nun von 100 % auf 0 % in Schritten von 1 % verringert. Ansonsten sind die Prozesse dieselben wie in der ersten Schleife, nur in umgekehrter Richtung.

## Von der PWM zum Digital-Analog-Converter (DAC)

Pulsweitenmodulation (PWM) ist eine vielseitige Technik, um analoge Signale mit digitalen Mitteln zu erzeugen. Sie wird häufig genutzt, um LEDs zu dimmen, Motoren zu steuern oder einfache Audio-Signale zu erzeugen. Für Anwendungen, die ein „echtes" analoges Signal erfordern, sind jedoch andere Techniken erforderlich. Hier kommen die sogenannten Digital-Analog-Wandler zum Einsatz.

Im einfachsten Fall kann ein solcher Wandler durch eine sogenannte R-2R-Leiter realisiert werden Diese besteht aus einem Netzwerk von Widerständen in einer speziellen Spannungsteiler-Anordnung. Damit werden die von verschiedenen Ports zur Verfügung gestellten Spannungspegel so eingeteilt, dass die digitale Eingabe eine proportionale analoge Ausgangsspannung erzeugt.

Da der Raspberry Pi keinen echten DAC enthält, muss man sich mit einer R-2R-Leiter behelfen. Bild 3 zeigt die Schaltung hierfür. Die Ausgangsspannung dieser Schaltung ist:

U = U(P7) / 2 + U(P6) / 4 + U(P5) / 8 + ... + U(P0) / 256

Damit kann also über einen 8-Bit-Port eine Analogspannung mit einer Auflösung von

5 V/256 = 20 mV

ausgegeben werden. Bild 4 zeigt einen Aufbauvorschlag für eine R-2R-Leiter.

In den PAD-Sets stehen keine 2-kΩ-Widerstände zur Verfügung. Notfalls kann man sich hier durch Serienschaltungen von 1-kΩ-Werten behelfen. Allerdings wird der Aufbau dadurch deutlich komplexer. Widerstände in normaler Bauform sind dagegen auch mit einem Wert von 2 kΩ erhältlich. Sind präzise Ausgangsspannungswerte erforderlich, sollte man Widerstände mit einer Toleranz von 1 % oder besser verwenden.

Die Steuerung der Ports erfolgt über ein passendes Python-Programm (R2R.py):

```
from gpiozero import LED
import time

# GPIO-Pins für die R-2R-Leiter (anpassen, falls andere verwendet werden)
pins = [17, 27, 22, 5, 6, 13, 19, 26]  # Beispiel für 8-Bit-DAC

# LEDs für die GPIO-Pins erstellen
outputs = [LED(pin) for pin in pins]

def set_dac(value):
    """
    Gibt einen digitalen Wert (0-255 für 8-Bit) an die R-2R-Leiter aus.
    :param value: Integer-Wert zwischen 0 und 255.
    """
    binary = f"{value:08b}"  # Binärdarstellung mit 8 Bits
    for i, bit in enumerate(binary):
        outputs[i].value = int(bit)  # Setze den Pin entsprechend (0 oder 1)

try:
    while True:
        for value in range(256):  # Durchlaufe alle Werte von 0 bis 255
            set_dac(value)
            time.sleep(0.01)  # Warte 10 ms, um die Ausgabe zu sehen

except KeyboardInterrupt:
    print("Programm beendet.")

finally:
    for output in outputs:
        output.off()  # Schalte alle GPIO-Pins aus
```

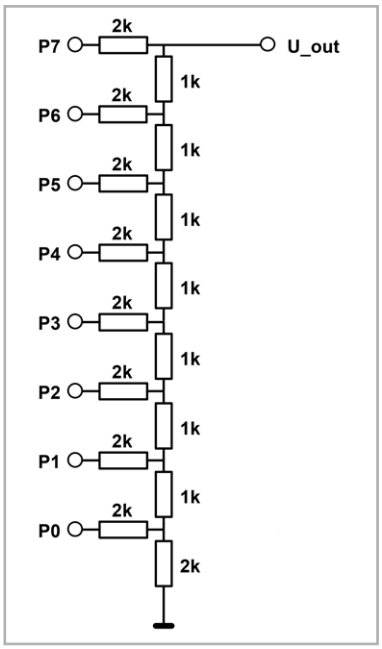


Bild 3: R-2R-Leiter als DAC

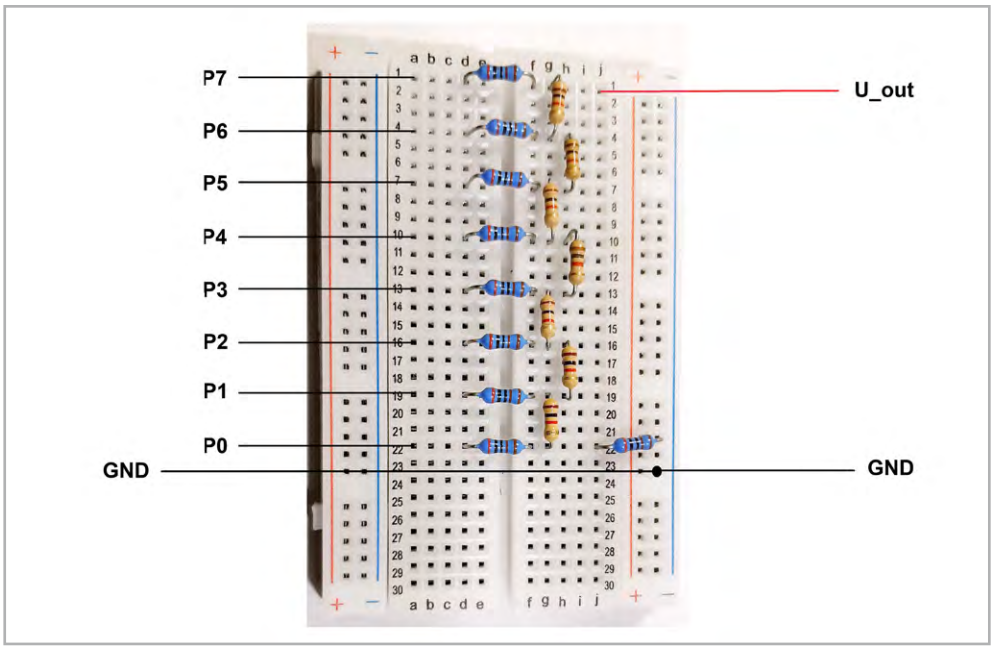


Bild 4: Praktischer Aufbau einer R-2R-Leiter

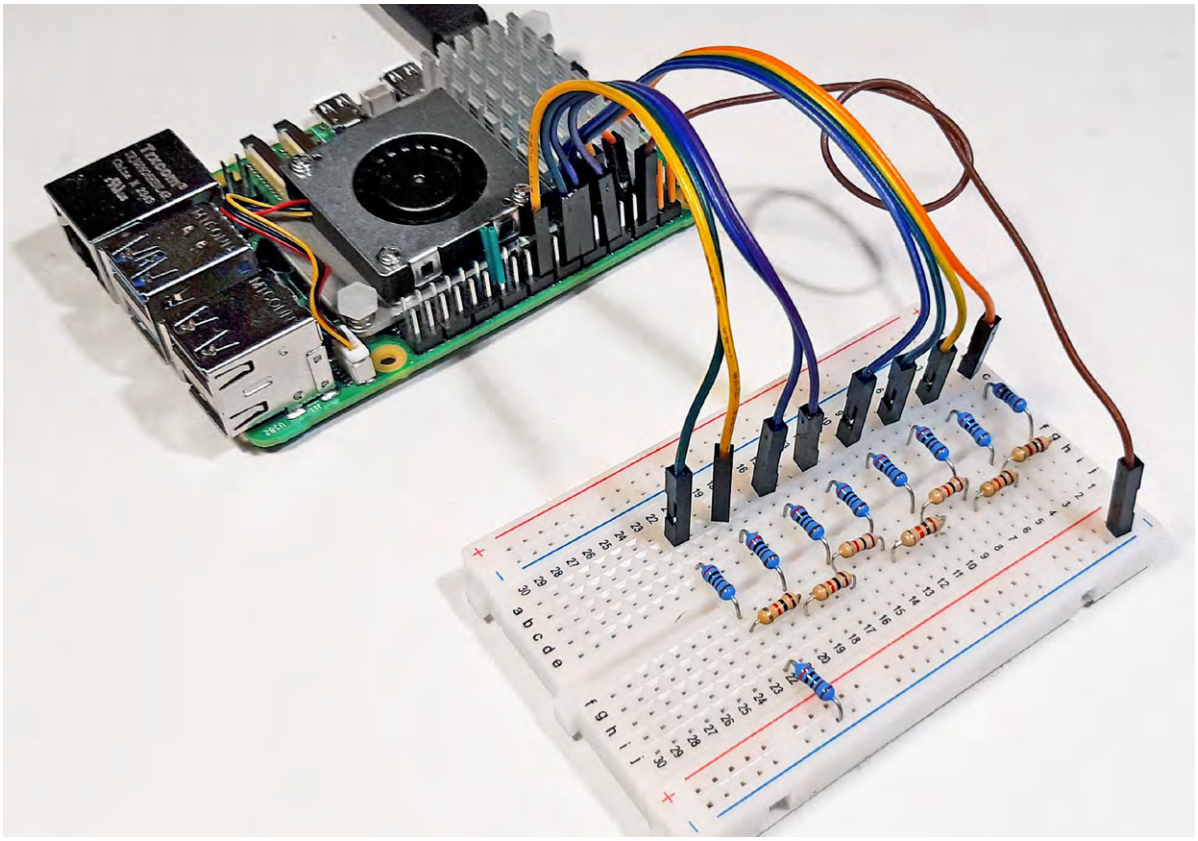

*Bild 5: R-2R-Leiter am Rapsberry Pi*

Zu Beginn des Programms werden die GPIO-Pins definiert, die mit der R-2R-Schaltung verbunden sind. Insgesamt werden acht Pins genutzt, da es sich um einen 8-Bit-DAC handelt. Dabei kommen die Pins 17, 27, 22, 5, 6, 13, 19 und 26 zum Einsatz. Bild 5 zeigt den zugehörigen Aufbau.

Diese Pins werden in einer Liste gespeichert:

```
pins = [17, 27, 22, 5, 6, 13, 19, 26]
```

Sie werden mithilfe der Klasse `LED` aus der `gpiozero`-Bibliothek als digitale Ausgänge eingerichtet.

Die Funktion `set_dac(value)` ist der zentrale Bestandteil des Programms. Sie nimmt einen ganzzahligen Wert zwischen 0 und 255 entgegen, wandelt diesen in eine 8-Bit-Binärdarstellung um und steuert damit die GPIO-Pins an.

Die Funktion gibt also einen digitalen Wert (zwischen 0 und 255) an die R-2R-Widerstandsleiter aus, indem sie die entsprechenden GPIO-Pins setzt.

Hierzu werden die folgenden Schritte umgesetzt:

1. Der `value`-Parameter wird als Ganzzahl zwischen 0 und 255 (8-Bit-Wert) übernommen
2. Die Zahl wird in eine 8-stellige Binärzahl umgewandelt (`f"{value:08b}"`).
3. Die folgende Schleife durchläuft jedes Bit der Binärzahl.
4. Die GPIO-Ausgänge (`outputs[i]`) werden entsprechend auf 0 oder 1 gesetzt, um die R-2R-Widerstandsleiter mit digitalen Werten zu speisen.

Wenn beispielsweise der Wert `value = 5` übergeben wird, ergibt sich die Binärdarstellung `00000101`, und die Pins würden folgendermaßen gesetzt:

| Bitposition | Wert | GPIO |
|---|---|---|
| 7 | 0 | Low |
| 6 | 0 | Low |
| 5 | 0 | Low |
| 4 | 0 | Low |
| 3 | 0 | Low |
| 2 | 1 | High |
| 1 | 0 | Low |
| 0 | 1 | High |

Diese binäre Ausgabe steuert die R-2R-Widerstandsleiter, die daraus eine analoge Spannung erzeugt.

Im Hauptteil des Programms wird eine Endlosschleife verwendet, um kontinuierlich alle möglichen Werte von 0 bis 255 zu durchlaufen. Für jeden Wert wird die Funktion `set_dac(value)` aufgerufen, und nach jeder Ausgabe erfolgt eine kurze Pause von 10 Millisekunden. Diese Verzögerung dient dazu, die analoge Ausgabe des DACs deutlich sichtbar zu machen und sicherzustellen, dass jede Spannung ausreichend lange gehalten wird.

Das Programm ist darauf ausgelegt, durch den Benutzer mit einer Tastenkombination (Strg + C) beendet zu werden. In diesem Fall werden alle GPIO-Pins automatisch deaktiviert, um sicherzustellen, dass keine ungewollten Zustände an den Ausgängen verbleiben.

**Hinweis:** Die im Text grün markierten Programme sind im Download-Paket enthalten.

Tabelle 1

**Vor- und Nachteile von PWM und R-2R-Leitern**

| Kriterium | PWM | R-2R-Leiter |
|---|---|---|
| Genauigkeit | abhängig von der PWM-Auflösung (Bit-Tiefe) und der Glättung (Filter) | hängt von der Genauigkeit der Widerstände ab (Toleranzen und Stabilität) |
| Geschwindigkeit | kann analoge Werte schnell erzeugen, benötigt jedoch eine Filterung für saubere Signale | sehr schnell, da die Umwandlung direkt erfolgt |
| Signalqualität | Ohne Filterung entstehen Welligkeiten im Signal (hochfrequentes PWM-„Rauschen"). | saubere analoge Signale direkt ohne Filterung |
| Aufwand | benötigt nur einen GPIO-Pin und eventuell einen Tiefpassfilter | benötigt mehrere GPIO-Pins (entsprechend der Bit-Tiefe) und präzise Widerstände |
| Flexibilität | einfach zu implementieren, softwaregesteuerte Anpassung (z. B. variable Grundfrequenz) | hardwareabhängig; Änderungen erfordern Anpassung der Schaltung |
| Kosten | minimal, da nur der Raspberry Pi und ggf. ein Widerstand und ein Kondensator für den Filter benötigt werden | höher, da präzise Widerstände und ggf. weitere externe Komponenten benötigt werden |

## PWM und R-2R-Leiter: Vor- und Nachteile

Die beiden vorgestellten Methoden erlauben es, digitale Signale in analoge Spannungen umzuwandeln. Beide Verfahren haben spezifische Vor- und Nachteile und eignen sich für unterschiedliche Anwendungen.

Bei der PWM wird die Höhe der analogen Spannung durch das Verhältnis zwischen Ein- und Aus-Zeit des Signals (Duty Cycle) bestimmt. Durch eine ausreichende Glättung mittels eines Tiefpassfilters lässt sich eine annähernd konstante analoge Spannung erzeugen. PWM ist eine einfache und ressourcensparende Methode, da sie nur einen einzigen GPIO-Pin benötigt und direkt von Mikrocontrollern oder Einplatinencomputern wie dem Raspberry Pi unterstützt wird.

PWM ist jedoch frequenzabhängig: Eine zu niedrige Frequenz kann zu sichtbaren Fluktuationen („Ripple") in der Ausgangsspannung führen, während bei sehr hohen Frequenzen die Anforderungen an die Filterung steigen. PWM eignet sich besonders für Anwendungen wie Motorsteuerungen, die Regelung von LEDs oder überall dort, wo hohe Effizienz bei geringem Hardwareaufwand gefragt ist.

Das R-2R-Verfahren ist konzeptionell einfacher, da es ohne Frequenzmodulation auskommt und direkt eine analoge Spannung proportional zur binären Eingabe liefert. Ein R-2R-DAC benötigt jedoch für jedes Bit des digitalen Eingangs einen separaten GPIO-Pin, was bei hochauflösenden DACs (z. B. 16 Bit) schnell viele Pins belegt. Zudem hängt die Genauigkeit der analogen Ausgabe stark von der Präzision der Widerstände ab. R-2R-DACs sind ideal für Anwendungen, die eine gleichmäßige und präzise analoge Ausgabe erfordern, wie etwa Audioanwendungen, präzise Messsysteme oder Steuerungen, bei denen eine ausreichende elektronische Filterung mit zu hohem Aufwand verbunden wäre.

Tabelle 1 fasst die Vor- und Nachteile der beiden Verfahren zusammen.

Letztendlich hängt die Wahl zwischen beiden Verfahren von den Anforderungen an Kosten, Signalqualität und Geschwindigkeit ab.

## Für Messtechnikprofis: Arbitrary Function Generator

In der Welt der Analogelektronik existieren sogenannte Funktionsgeneratoren (engl. Arbitrary Function Generator). Die damit erzeugbaren „Funktionen" umfassen meist ein Sinus-, ein Rechteck- und ein Dreiecksignal. Diese Signalformen sind mit analog-elektronischen Mitteln leicht zu erzeugen. Andere Signalformen dagegen sind bei solchen Geräten kaum zu finden.

Mithilfe von Python und einem Raspberry Pi kann man diese Einschränkung leicht umgehen. Hier können nahezu beliebige Signalformen mit hoher Präzision erzeugt werden. Die gewünschte Kurvenform kann dabei als Datensequenz in digitaler Form im Speicher des Raspberry abgelegt werden. Die Ausgabe erfolgt über einen Parallelport mit angeschlossener R-2R-Leiter. Dazu müssen in einem Programm die gewünschten Funktionswerte in einem eindimensionalen Array abgelegt werden. Eine Schleife sorgt dann dafür, dass die Werte nacheinander an die I/O-Pins ausgegeben werden. Dort erfolgt mit dem R2R-Netzwerk die Umwandlung in eine analoge Spannung.

| | A | B | C | E |
|---|---|---|---|---|
| 1 | Grad | 256er Teil | Sinus | DA-Wert |
| 2 | 0 | 0,0 | 0,00 | 128 |
| 3 | 1 | 1,4 | 0,02 | 131 |
| 4 | 2 | 2,8 | 0,05 | 134 |
| 5 | 3 | 4,2 | 0,07 | 137 |
| 6 | 4 | 5,6 | 0,10 | 141 |
| 7 | 5 | 7,1 | 0,12 | 144 |
| 8 | 6 | 8,5 | 0,15 | 147 |
| 9 | 7 | 9,9 | 0,17 | 150 |
| 10 | 8 | 11,3 | 0,20 | 153 |
| 11 | 9 | 12,7 | 0,22 | 156 |
| 12 | 10 | 14,1 | 0,24 | 159 |
| 13 | 11 | 15,5 | 0,27 | 162 |
| 14 | 12 | 16,9 | 0,29 | 165 |
| 15 | 13 | 18,4 | 0,31 | 168 |
| 16 | 14 | 19,8 | 0,34 | 171 |
| 17 | 15 | 21,2 | 0,36 | 174 |
| 18 | 16 | 22,6 | 0,38 | 177 |
| 19 | 17 | 24,0 | 0,41 | 180 |
| 20 | 18 | 25,4 | 0,43 | 183 |
| 21 | 19 | 26,8 | 0,45 | 186 |
| 22 | 20 | 28,2 | 0,47 | 189 |

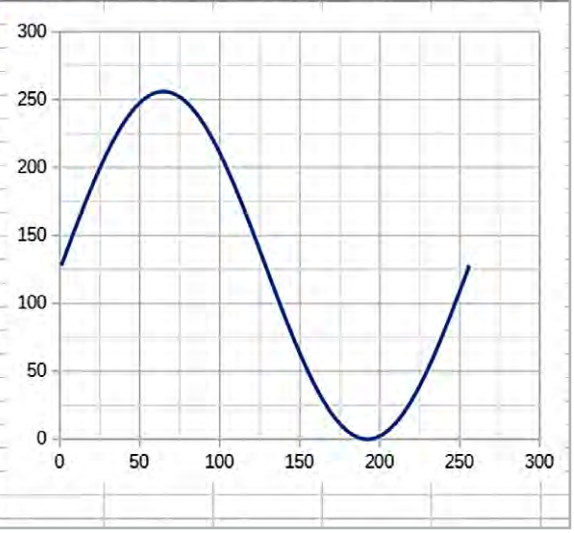


*Bild 6: Daten in einer Tabellenkalkulation*


*Bild 7: „Gestörte Sinusfunktion" aus Daten einer Tabellenkalkulation*

Mit einer Excel- oder LibreOffice-Datei können die Funktionswerte komfortabel erzeugt werden. Die Tabelle kann aus Datenfeldern bestehen, die jeweils 256 Werte fassen. Das erste Feld enthält einfach die Zahlen 0 bis 255 als „Indices". Im zweiten Feld („Values") kann eine beliebige Funktion eingetragen werden. So liefert

=GANZZAHL(127+127*SIN(A_X*2*3,14/255))

z. B. einen sinusförmigen Spannungsverlauf. Die gewünschte Funktion muss mit Copy & Paste in alle 256 Zellen kopiert werden. Bild 6 zeigt einen Ausschnitt aus der entsprechenden Tabelle. Ein passendes Spreadsheet-Beispiel (FuncTable.xls) findet sich zudem im Download-Paket.

Um spezielle Funktionen zu erzeugen, kann man nun auch einzelne Zellen manuell verändern und beliebige Werte zwischen 0 und 255 eintragen. Im Beispiel wurden in den vier Zellen, welche die Maximal- bzw. Minimalwerte des Sinus enthalten hatten, die Zahl 128 eingetragen. Dadurch entsteht die in Bild 7 gezeigte „gestörte" Sinusfunktion, die jeweils an den Scheitelwerten „Aussetzer" aufweist. Damit kann man nun z. B. die Reaktion einer Analogschaltung auf dieses „fehlerhafte" Sinussignal prüfen.

Mit dem folgenden Programm (R2R_sinus.py) kann die so erzeugte Funktion über die R-2R-Leiter als elektrisches Signal ausgegeben werden:

```
from gpiozero import LED
import time, array

# Ein Array erstellen (Typ 'i' steht für ganze Zahlen)
zahlen = array.array('i',
[128,131,134,137,141,144,147,150,153,156,159,162,165,168,171,174,
177,180,183,186,189,191,194,197,199,202,205,207,209,212,214,217,
219,221,223,225,227,229,231,233,235,236,238,240,241,243,244,245,
246,247,248,249,250,251,252,253,253,254,254,255,255,255,255,255,
255,255,255,255,255,254,254,254,253,253,252,251,250,249,248,247,
246,245,243,242,240,239,237,236,234,232,230,228,226,224,222,220,
218,215,213,211,208,206,203,201,198,195,193,190,187,184,181,179,
176,173,170,167,164,161,158,155,152,148,145,142,139,136,133,130,
126,123,120,117,114,111,108,104,101,98,95,92,89,86,83,80,
77,75,72,69,66,63,61,58,55,53,50,48,45,43,41,38,
36,34,32,30,28,26,24,22,20,19,17,16,14,13,11,10,
9,8,7,6,5,4,3,3,2,2,1,1,0,0,0,0,
0,0,0,1,1,1,2,2,3,4,4,5,6,7,8,10,
11,12,13,15,16,18,20,21,23,25,27,29,31,33,35,37,
39,42,44,47,49,51,54,57,59,62,65,67,70,73,76,79,
82,85,88,91,94,97,101,103,106,109,112,115,119,122,125])
```

*Fortsetzung nächste Seite*

```
# GPIO-Pins für die R-2R-Leiter (anpassen, falls andere verwendet werden)
pins = [4, 17, 27, 22, 18, 23, 24, 25]  # Beispiel für 8-Bit-DAC


# Ausgänge ("LEDs") für die GPIO-Pins erstellen
outputs = [LED(pin) for pin in pins]

def set_dac(value):
 binary = f"{value:08b}"  # Binärdarstellung mit 8 Bits
 # print(binary)
 for i, bit in enumerate(binary):
 outputs[i].value = int(bit)  # Setze den Pin entsprechend (0 oder 1)

try:
 while True:
 for value in range(255):  # Durchlaufe alle Werte von 0 bis 255
 set_dac(zahlen[value])
 time.sleep(.01)  # Warte 10 ms, um die Ausgabe zu sehen
except KeyboardInterrupt:
    print("Programm beendet.")
finally:
    for output in outputs:
        output.off()  # Schalte alle GPIO-Pins aus
```

Bild 8 zeigt den erzeugten Spannungsverlauf auf einem analogen Oszilloskop. Beim Vergleich der Bilder erkennt man sofort die gute Übereinstimmung. Die Zahlenwerte in der Excel-Tabelle werden präzise in die analoge, auf dem Oszilloskop dargestellte Ausgabe umgesetzt.

## Vom R-2R-Netzwerk zum echten Digital-Analog-Wandler

Ein R-2R-Widerstandsnetzwerk ist eine einfache und kostengünstige Methode, um digitale Signale in analoge Spannungen umzuwandeln. Es basiert auf einer Kette von Widerständen mit den Werten R und 2R, die zusammen einen mehrstufigen Spannungsteiler bilden. Durch Anlegen von digitalen High- oder Low-Pegeln an die Eingänge entsteht eine gewichtete Summe, die eine analoge Ausgangsspannung liefert.

Aufgrund seiner Einfachheit hat das R-2R-Netzwerk einige Nachteile. Es ist empfindlich gegenüber Widerstandstoleranzen, die die Genauigkeit der Ausgabe beeinflussen können. Zudem kann es bei hohen Schaltfrequenzen zu Verzerrungen kommen, da parasitäre Kapazitäten die Signaltreue beeinträchtigen.

Ein echter Digital-Analog-Wandler (DAC) hingegen verwendet präzisere Widerstandsnetzwerke oder stromgesteuerte Wandlungsmethoden. Moderne DACs nutzen oft spezielle Ladungspumpen, Sigma-Delta-Modulation oder segmentierte Widerstandsarrays, um eine höhere Genauigkeit und geringere Verzerrungen zu erreichen. Außerdem verfügen sie über eine bessere Linearität, höhere Auflösung (bis zu 24 Bit) und kurze Wandlungszeiten, was für anspruchsvolle Anwendungen wie Audioverarbeitung oder präzise Sensorsysteme entscheidend ist.

Der Übergang von einem R-2R-Netzwerk zu einem „echten" DAC verbessert also die Signalqualität erheblich und ermöglicht Anwendungen, die eine exakte und stabile analoge Ausgangsspannung erfordern.

Weitere Details dazu werden in späteren Artikeln behandelt, in welchen auch DACs für spezielle Bussysteme (I2C oder SPI) vorgestellt werden.

## Ergänzungen und Übungen

- Wie könnte ein Analogfilter zur PWM-Ausgabe des Raspberry Pi aussehen (Hinweis: RC-Tiefpass)?
- Wie beeinflusst die Grundfrequenz des PWM-Signals den „Ripple", also das hochfrequente „Rauschen" des Signals?
- Wie könnte ein Programm aussehen, das eine 12-Bit-R-2R-Leiter ansteuert?
- Wo liegen die praktischen Grenzen einer diskret aufgebauten R-2R-Leiter?

## Ausblick

In diesem Artikel wurde die Technik der Pulsweitenmodulation (PWM) zur Ansteuerung der Helligkeit einer LED verwendet. Zudem wurden R-2R-Leitern als spezielle Digital-Analog-Converter eingesetzt. Diese erlauben es, mithilfe eines entsprechenden Python-Programms beliebige Analogsignale zu erzeugen. Sie können z. B. als universelle Test- und Prüfsignale im Elektroniklabor nutzbringend eingesetzt werden.

Im nächsten Beitrag soll es um den Einsatz von Sensoren gehen. Die Daten dieser in der gesamten Technik so wichtigen Bauelemente können mit Python-Programmen eingelesen und verarbeitet werden.

Dabei sollen insbesondere auch Signalverarbeitungsmethoden wie Mittelung, Filterung und Signalkonditionierung mit Python näher betrachtet werden. ELV

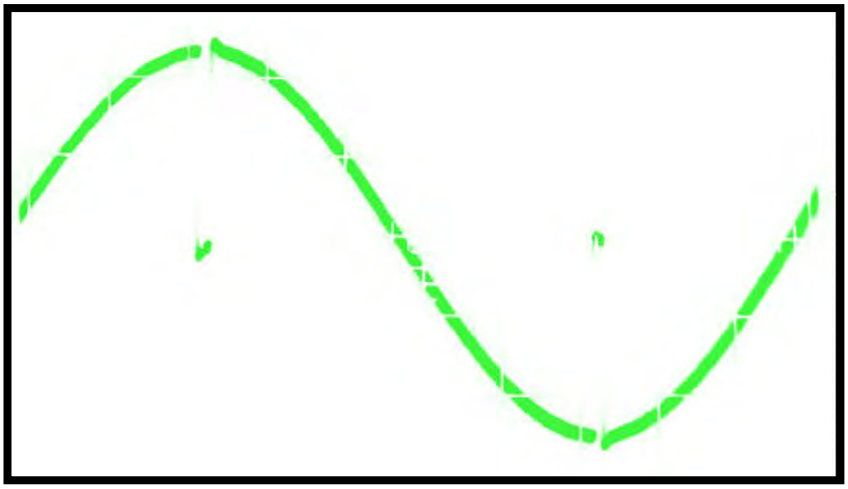

*Bild 8: Sinusfunktion mit „Störung" (100 ms bzw. 500 mV pro Skalenteil)*

### Material

Raspberry Pi mit Netzteil

| | |
|---|---|
| z. B. Raspberry Pi 4 Model B, 8 GB RAM | Artikel-Nr. 250567 |
| z. B. Raspberry Pi 4 USB-Netzteil Typ C | Artikel-Nr. 250962 |
| Bedienpanel MEXB-BP1 | Artikel-Nr. 157431 |

jeweils ca. 10 Stück Widerstände 1 kΩ und 2 kΩ
Breadboard und Jumper-Kabel

**Zum Download-Paket**