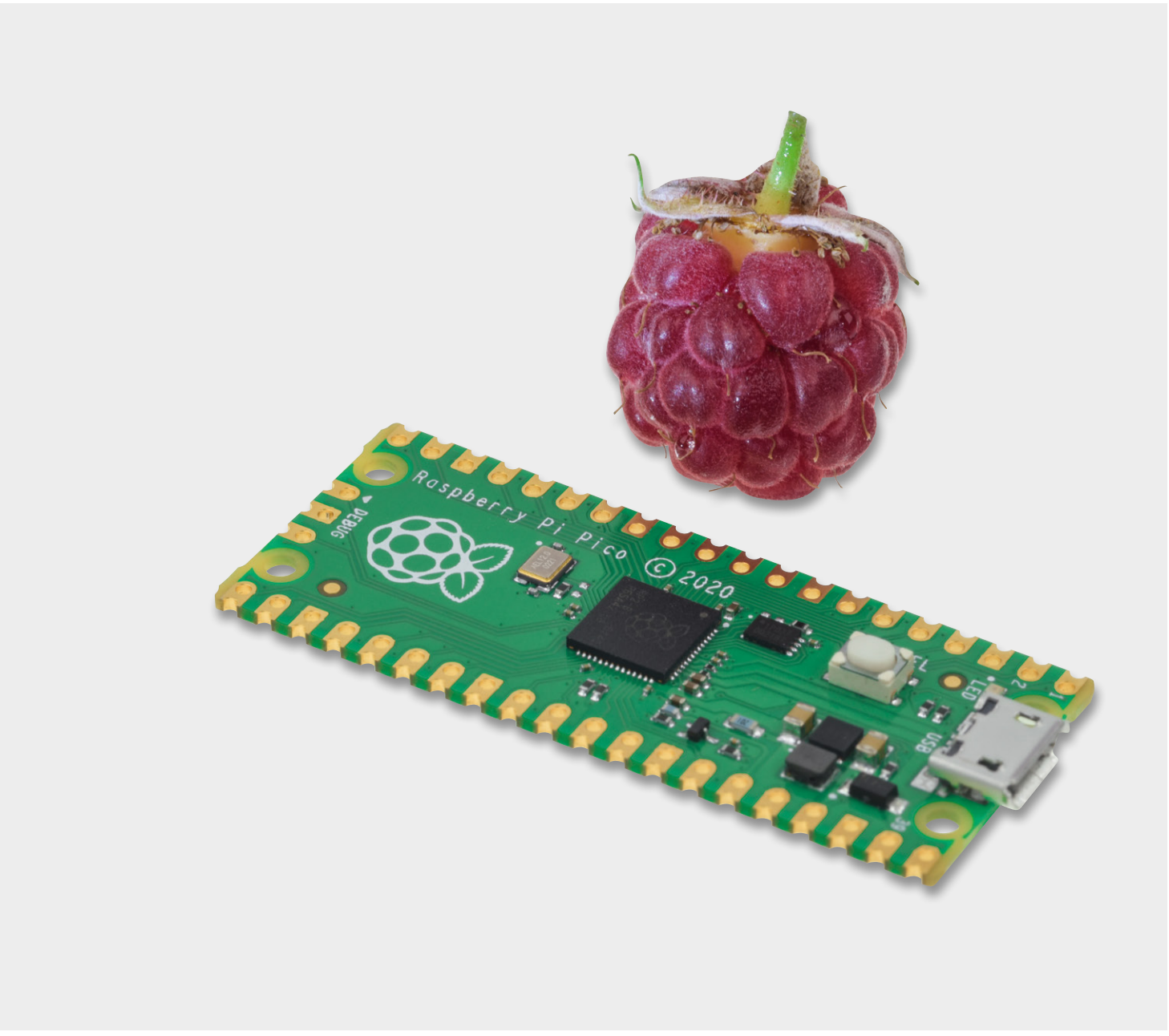
Raspberry Pi Pico

Programmieren mit MicroPython und C

Teil 1

Nachdem der „Pico“ als erster Mikrocontroller der Raspberry Pi Foundation auf den Markt kam, konnte er sofort vom überragenden Erfolg der Raspberry-Pi-Reihe profitieren. Der günstige Preis und die gute Verfügbarkeit des Controller-Boards taten ihr Übriges. Im Gegensatz zu den klassischen Raspberry Pi Boards findet sich auf dem Pico kein Betriebssystem. Als reiner Controller benötigt der Chip immer ein eigenes Programm, um arbeiten zu können.



Programmierung

Für die Programmierung des Pico kommen grundsätzlich MicroPython oder C/C++ infrage. Für C/C++ kann die bekannte Arduino-IDE genutzt werden. Damit wird der Pico auch für die Arduino-Fangemeinde äußerst attraktiv.

Mit Python bzw. MicroPython steht jedoch auch eine weitere hochinteressante Programmierbasis zur Verfügung. MicroPython ist mit Python Version 3 kompatibel und für den Betrieb auf einem Mikrocontroller optimiert. Damit wird auch die Verbindung zum klassischen „RasPi" hergestellt, der häufig mit Python programmiert wird.

Mit diesem Artikel startet eine Beitragsreihe, die den Raspberry Pi Pico ausführlich vorstellen wird. Neben allgemeinen Informationen zum Chip und seiner Programmierung sollen dabei vor allem auch praktische Anwendungen und Projekte im Vordergrund stehen.

Ungleiche Brüder

Direkt nach der Einführung des klassischen Arduinos war die Welt der Bastler, Hobbyelektroniker und „Maker" noch in Ordnung. Es gab ein einziges Board und eine dazu passende Entwicklungsumgebung (IDE – engl. Integrated Development Environment). Aber natürlich ist auch hier der Fortschritt nicht stehen geblieben. Weitere Arduino-Versionen wurden entwickelt. Neben den größeren und leistungsfähigeren Boards (Arduino Mega oder Due) entstanden auch kleinere Varianten wie der Arduino Micro oder der Nano.

Als dann im Jahr 2012 der Raspberry Pi auf dem Markt erschien, wurde bei Google häufig angefragt, wo denn der Unterschied zwischen Arduino und Raspberry Pi liege. Aber auch hier lagen die Dinge noch relativ einfach: Der RasPi war ein Mini-Computer, Arduino ein Controller-Board.

Mit dem Aufkommen der Espressif-Controller wie zuerst dem ESP8266 und seinem Nachfolger ESP32 bekam der Arduino schließlich ernsthafte Konkurrenz. Die neuen Chips und Boards waren erheblich leistungsfähiger und preisgünstiger als die Arduino-Varianten. Dennoch konnten sie sehr einfach mit der bekannten Arduino-IDE programmiert werden. Der Markt war damit wieder etwas unübersichtlicher geworden.

Mit dem RP2040 kam schließlich 2021 ein Controller der Raspberry Pi Foundation auf den Markt. Damit stehen drei recht verschiedene Systeme zur Verfügung, die rein äußerlich sehr ähnlich erscheinen (s. Bild 1). Somit fällt die Wahl des passenden Systems nicht immer leicht. Tabelle 1 liefert einen Überblick der Varianten.

Der Arduino bleibt dabei weit abgeschlagen in Hinblick auf Prozessorleistung, Speicher oder Anzahl der verfügbaren Pins. Für einfachere Projekte oder Lehrzwecke kann er allerdings immer noch eingesetzt werden.

Die Arduino-Community versuchte mit einer Vielzahl neuer Boards der Konkurrenz entgegenzutreten, allerdings konnten sich diese bislang nicht wirklich durchsetzen.

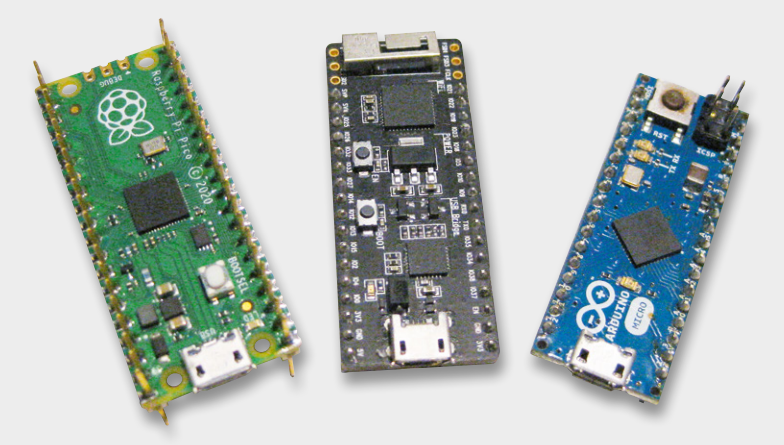


Bild 1: Drei ungleiche Brüder: Raspberry Pi Pico, ESP32 und Arduino Micro

Der ESP hat sich dagegen seinen Platz in der Maker-Gemeinschaft erobert. Die integrierte WLAN-Schnittstelle und die Möglichkeit der Programmierung mit Python bieten hier klare Vorteile.

Obwohl der Pico dem ESP32 rein technisch nicht wirklich überlegen ist, konnte er sich rasch etablieren. Die fehlenden WLAN-Fähigkeiten wurden durch den hohen Bekanntheitsgrad der Raspberry Pi Foundation ausgeglichen. Zudem spielen sicher auch der günstige Preis und einige spezielle Features wie die programmierbaren Eingabe-Ausgabe-Einheiten (PIO) eine wesentliche Rolle für den Erfolg des Pico.

Inzwischen ist mit dem seit Juli 2022 erhältlichen Pico W auch eine WLAN-fähige Variante des Pico verfügbar. Es steht zu erwarten, dass diese für die aktuellen ESP32-Boards eine ernsthafte Konkurrenz darstellen wird.

Der im Pico W verwendete CYW43439-Chip von Infineon unterstützt neben WLAN sowohl Bluetooth Classic als auch Bluetooth Low-Energy. Zur Markteinführung sind diese beiden Features allerdings noch nicht aktiviert. Bleibt zu hoffen, dass dies in Zukunft nachgeholt wird.

Python oder C/C++?

Wenn man sich für den Pico als Controller entschieden hat, stellt sich zunächst die Frage, mit welcher Sprache das Board programmiert werden soll. Eine unter Entwicklern weit verbreitete Weisheit lautet: „Für kurze Entwicklungszeiten nimmt man Python, für anspruchsvolle Projekte nimmt man C." Obwohl dieser einfache Satz sicher eine gewisse

Tabelle 1

Eigenschaften der verschiedenen Mikrocontroller

	Raspberry Pi Pico (W)	Arduino Micro	ESP32
Hauptprozessor (CPU)	32-Bit RP2040 Cortex-M0 + Dual-Core	ATmega328P	Dual-Core 32-bit LX6
Taktfrequenz	133 MHz	16 MHz	240 MHz
Arbeitsspeicher (RAM)	264 kB	2 kB	520 kB
Programmspeicher (Flash)	2 MB	32 KB	4 MB
GPIO	26	14	34
Schnittstellen	2 x UART 2x I2C USART	USART I2C SPI	UART, I2C, SPI WLAN Bluetooth
PWM-Kanäle	16	6	16
ADC	3x 12 Bit	6x 10 Bit	18x 12 Bit
Timer	4	3	4
Sensoren	Onboard-Temperatur-sensor	-	Onboard-Hallsensor
Programmierung	MicroPython C, C++ Arduino-IDE	Arduino-IDE C++	MicroPython C, C++ Arduino-IDE
WLAN	Nur Pico W: 2,4 GHz - 802,11n	nein	ja

Wahrheit enthält, hilft er nicht immer weiter. Insbesondere der Einsteiger sollte sich gut überlegen, mit welcher Sprache er starten will.

Da sich Python und seine Mikrocontroller-Variante MicroPython immer mehr durchsetzen, wird in unserer Artikelreihe häufig dieser Variante der Vorzug gegeben. Zudem werden moderne und aktuelle Themen wie Machine Learning und Künstliche Intelligenz hauptsächlich in Python umgesetzt. Möchte man auch diese Bereiche mit einbeziehen, ist Python ohnehin das Mittel der Wahl.

Die Hardware-Funktionen des Picos sind in Python durch ein Standard-Maschinenmodul zugänglich. Damit können neue Projekte einfacher entwickelt werden. Aber auch bestehende MicroPython-Anwendungen sind ohne große Probleme portierbar. Zudem erlaubt Python-Code die einfache Nutzung des zweiten Prozessorkerns über das Thread-Modul.

Neben den Standardfunktionen verfügt der RP2040 über einige spezielle interne Hardware-Einheiten, die auf anderen Mikrocontrollern nicht zur Verfügung stehen.

So stellt das programmierbare I/O-System (PIO) eine vielseitige Hardware-Substruktur dar, mit dem neue I/O-Schnittstellen erstellt und mit hoher Geschwindigkeit betrieben werden können. Im zugehörigen Software-Modul finden sich umfassende PIO-Bibliotheken, mit denen die neuartige PIO-Hardware angesprochen und über MicroPython genutzt werden kann. Auch dieser Aspekt spricht für den Einsatz von MicroPython.

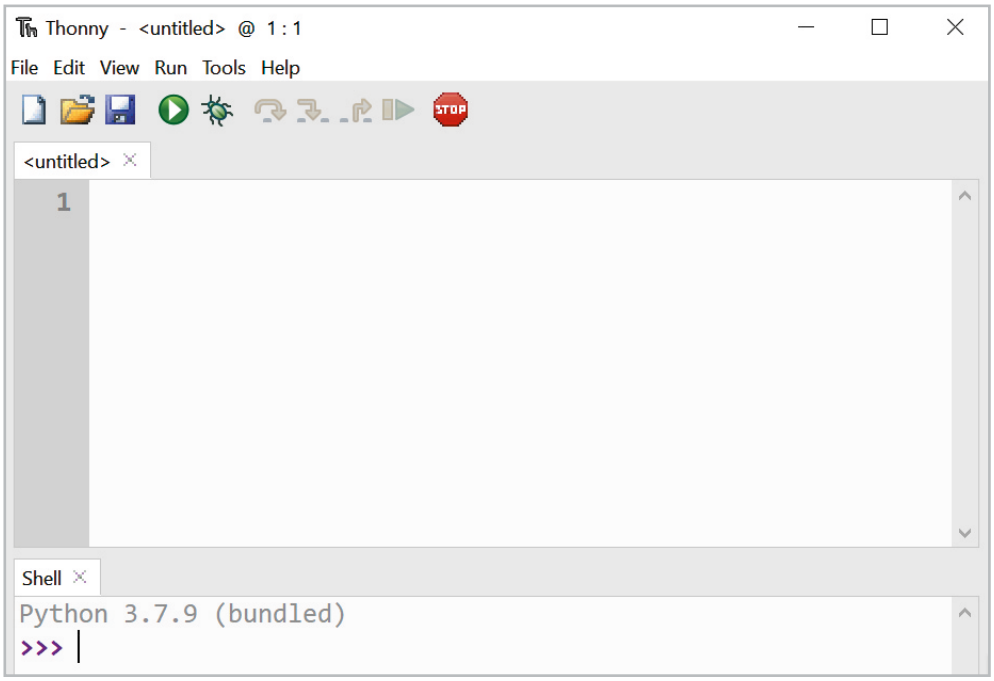


Bild 2:
Die Thonny-IDE

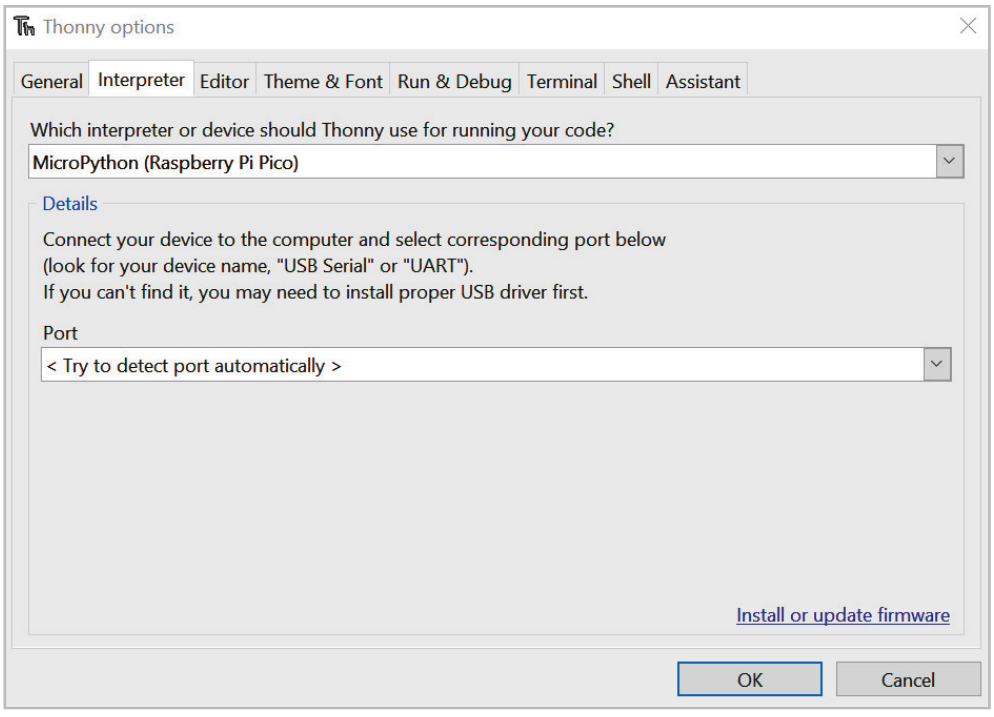


Bild 3:
Thonny Options

Python ist damit der schnellste und effizienteste Weg, um mit der Software-Entwicklung auf dem Raspberry Pico zu beginnen. Im Folgenden soll die offizielle MicroPython-Portierung für RP2040-basierte Mikrocontroller-Boards zum Einsatz kommen. Damit steht eine komplette Python-3-Implementierung für Mikrocontroller oder andere kleine eingebettete Systeme zur Verfügung.

Das Pico-Board bietet für seinen Preis eine große Menge an Systemspeicher und Rechenleistung. Zusammen mit MicroPython entsteht so ein ernst zu nehmendes Werkzeug für die Entwicklung eingebetteter Systeme.

Allerdings bietet auch C/C++ immer noch einige Vorteile. Insbesondere wenn es um Effizienz oder hohe Arbeitsgeschwindigkeiten geht, bleibt C klar im Vorteil. Zudem steht mit der bekannten Arduino-IDE eine einfache Möglichkeit zur Verfügung, diese Vorteile zu nutzen. Aus diesen Gründen sollte auch diese Variante bei einigen Gelegenheiten Verwendung finden.

Installation von MicroPython

Wenn der Pico mit MicroPython programmiert werden soll, dann muss zunächst eine Laufzeitumgebung für MicroPython vorbereitet werden. Dafür gibt es verschiedene Vorgehensweisen. Mithilfe der Thonny-IDE kann dieser Schritt einfach umgesetzt werden. Die Thonny-IDE bietet zudem eine vollständige Programmier- und Entwicklungsumgebung (IDE) für MicroPython.

Thonny kann über die entsprechende Website [1] geladen und installiert werden. Der Download der exe-Datei für Windows und die Installation sind in wenigen Minuten abgeschlossen.

Nach dem Start über das Icon steht die IDE zur Verfügung (Bild 2).

Über
Run ⇨ Select interpreter
öffnet sich „Thonny Options“ (Bild 3).

Dort wird
„Install or update firmware“
ausgewählt.

Nun wird der Pico über ein Kabel (USB-Micro) mit dem PC verbunden. Dabei muss der „BOOTSEL“-Button gedrückt werden.

Dann wird der MicroPython-Interpreter via „Install“ übertragen (Bild 4).

Danach steht der Programmierung des Pico mit MicroPython nichts mehr im Wege.

Für einen ersten Test kann man nun die On-Board-LED des Pico einschalten. Dazu kopiert man den unten stehenden Code in das Code-Fenster (<untitled>) und klickt anschließend auf den grün-weißen „Play“-Button (Bild 5).

Nun wird man zunächst aufgefordert, einen Namen für das Programm zu vergeben.

```
from machine import Pin
from utime import sleep

led_onboard = Pin(25, Pin.OUT)
led_onboard.on()
sleep(1)
led_onboard.off()
```

Nach dem Start des Programms leuchtet die LED auf dem Pico-Board für eine Sekunde auf und erlischt dann wieder.

Das Programm findet sich auch im Download-Paket [2] zu diesem Artikel (LED_tst.py).

Weitere Details und Hinweise und Information zur Arbeit mit Thonny finden sich unter [1].

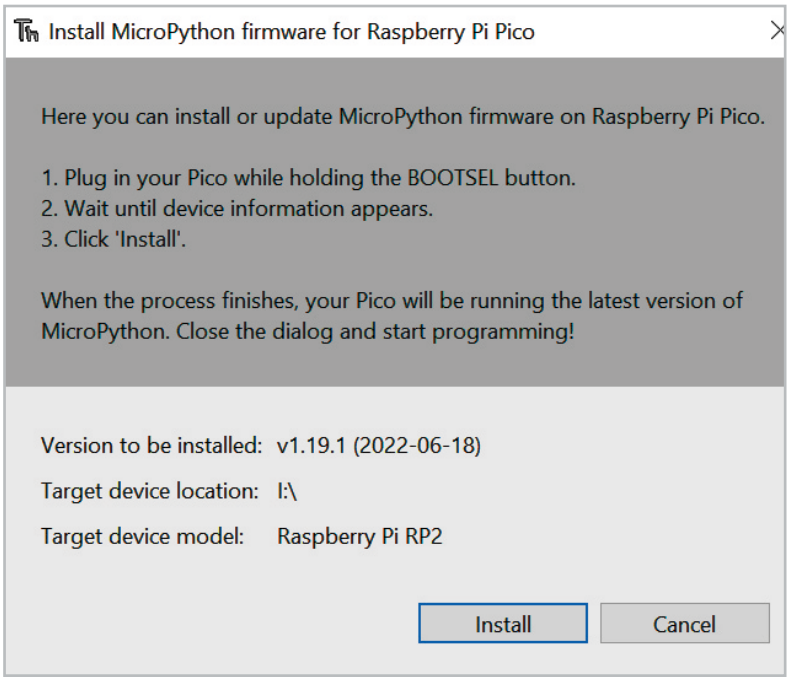


Bild 4: Installation der MicroPython Firmware

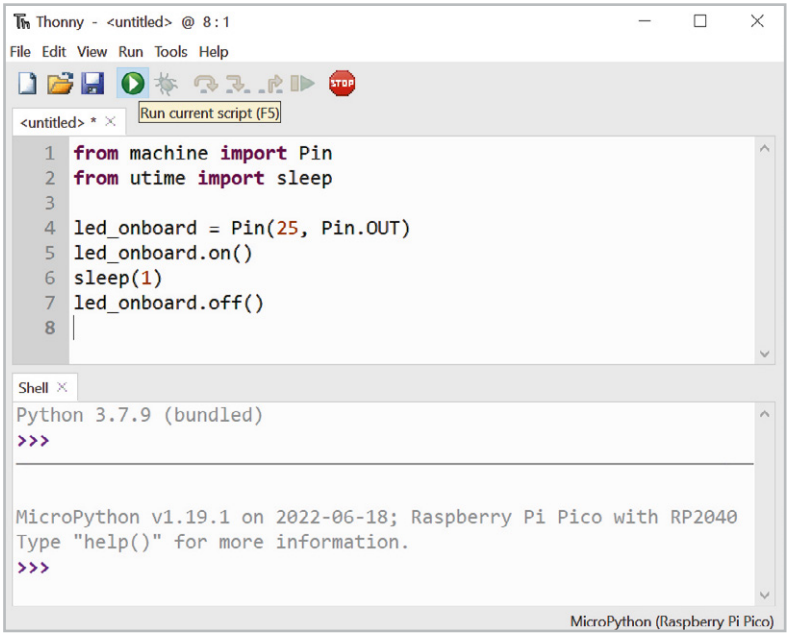


Bild 5: Starten eines Programms in der Thonny-IDE

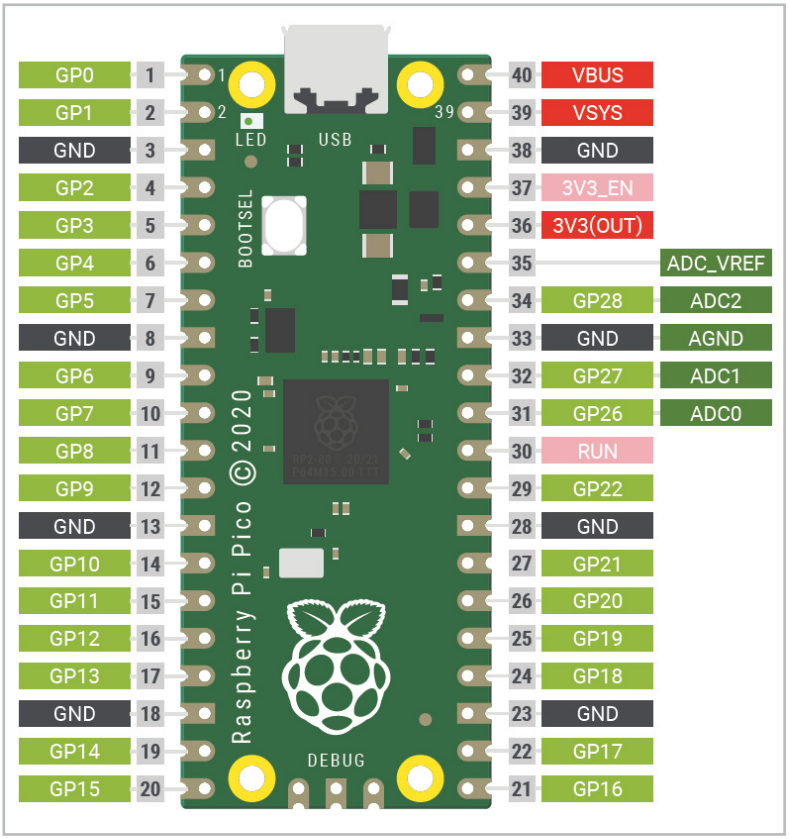


Bild 6: Die Pins des Pico-Boards

Physical computing: Einfache Anwendungen und Projekte

Nach der erfolgreichen Installation von Thonny und des Python-Interpreters steht ein komplettes System zur Entwicklung von Physical-Computing-Anwendungen zur Verfügung.

Bild 6 zeigt die wichtigsten Pin-Belegungen des Pico-Boards.

Man erkennt, dass die Ports des Controllers der Reihe nach an die Pins des Boards herausgeführt wurden. Lediglich die Ground-Anschlüsse (GND) unterbrechen die fortlaufende Reihenfolge. Ab Pin 36 sind dann die Betriebsspannungen zu finden.

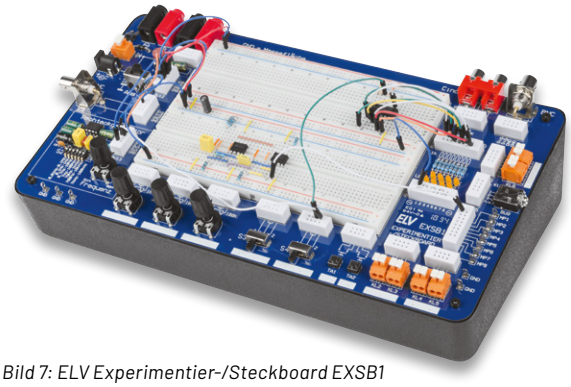


Bild 7: ELV Experimentier-/Steckboard EXSB1

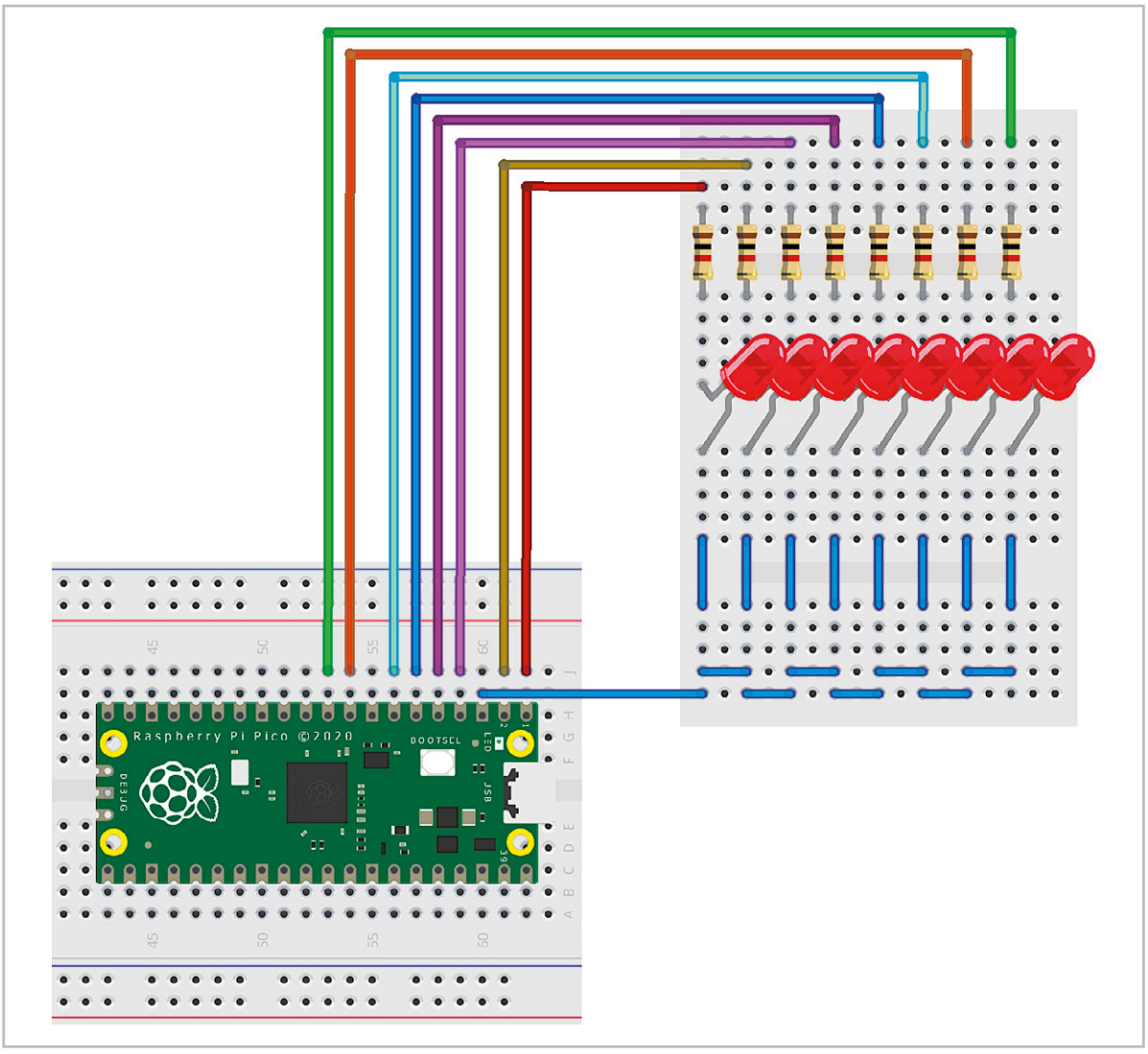


Bild 8: Realitätsnahes Schaltbild zum Lauflicht

Um die Funktionsweise der Pico-Boards besser zu verstehen, sollen hier zunächst einige Beispiele zu verschiedenen Port-Ausgaben vorgestellt werden. Für den Aufbau dieser Projekte ist das EXSB1-Board (Bild 7, s. Materialliste) besonders gut geeignet, da es alle erforderlichen Hardware-Komponenten enthält. Das Aufbausystem bietet zudem ausreichend Platz, um auch weitere Bausteine oder Module aufzunehmen. Aus diesem Grunde wird es im Verlauf der Artikelserie immer wieder zum Einsatz kommen.

Lauflicht

Sogenannte Lauflichter sind vielfältig einsetzbar. Sie finden sich als Leitsignale an Straßenbaustellen oder als Landehilfe auf Flugplätzen. Zudem hat eine Variante des laufenden Lichtpunktes im „Larson-Scanner" große Bekanntheit erlangt. In der TV-Serie „Knight Rider" ziert ein solches Lauflicht die Front des Fahrzeugs „K. I. T. T.", das den Helden der Serie immer wieder aus gefährlichen Situationen befreit. Mit dem Pico-Board kann eine Version dieses „Scanners" problemlos aufgebaut werden. Die Schaltung dazu sieht aus wie in Bild 8 gezeigt.

Bild 9 zeigt den Aufbauvorschlag auf einem EXSB1-System. Das Python-Programm dazu wurde so aufgebaut, dass es leicht auf weitere LEDs erweitert werden kann:

```
#LED_chaser_list.py

from machine import Pin
from time import sleep

LED=[0,1,2,3,4,5,6,7]  # 8,9,10,11,12,13,14,15,16,17,18,19,
20,21,22,26,27,28
No_of_LEDs=len(LED)

del_time = 0.1

for n in range(No_of_LEDs):
  LED[n] = Pin(LED[n],Pin.OUT)


while True:
  for n in range(No_of_LEDs):
    LED[n].value(1)
    sleep(del_time)
    LED[n].value(0)
```

Nach dem Import der erforderlichen Module „machine" und „time" werden die verwendeten Pins in einem Array definiert. Damit kann man das System leicht ausbauen und auch andere Pin-Reihenfolgen wären problemlos zu realisieren, wie im Kommentar zu dieser Zeile angedeutet wird:

```
# 8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,26,27,28
```

Zu beachten ist lediglich, dass ab Port 22 nicht alle I/O-Pin-Nummern zur Verfügung stehen.

In der Variablen „No_of_LEDs" wird die Anzahl der aktuell verwendeten Leuchtdioden festgehalten. Die Variable „del_time" bestimmt die Laufgeschwindigkeit des Lichtpunkts.

In der ersten „for"-Schleife werden alle benutzten Pins als Ausgänge definiert. In der Hauptschleife schließlich werden die LEDs der Reihe nach ein- und wieder ausgeschaltet. Auf diese Weise entsteht der gewünschte Lauflichteffekt.

Messen heißt Wissen

Neben den digitalen I/O-Pins zählen die ADCs (Analog-Digital-Converter) zu den wichtigsten Funktionseinheiten eines Mikrocontrollers. Auch wenn die Digitalisierung zunehmend bedeutender wird, ist die reale Welt letztlich doch analog aufgebaut. So ändern sich Temperaturen nicht sprunghaft, sondern allmählich. Auch ein Tag beginnt nicht abrupt, sondern es wird langsam heller. Um diese kontinuierlich veränderlichen Werte erfassen zu können, ist ein einfacher Digitaleingang nicht ausreichend. Vielmehr müssen die entsprechenden Messwerte von einem Analog-Digital-Wandler (ADC) umgesetzt werden.

Der Pico verfügt über vier analoge Messkanäle. Allerdings sind davon nur drei (ADC0, ADC1 und ADC2) verfügbar, da der vierte Kanal bereits mit der internen Temperaturmessung des Chips belegt ist.

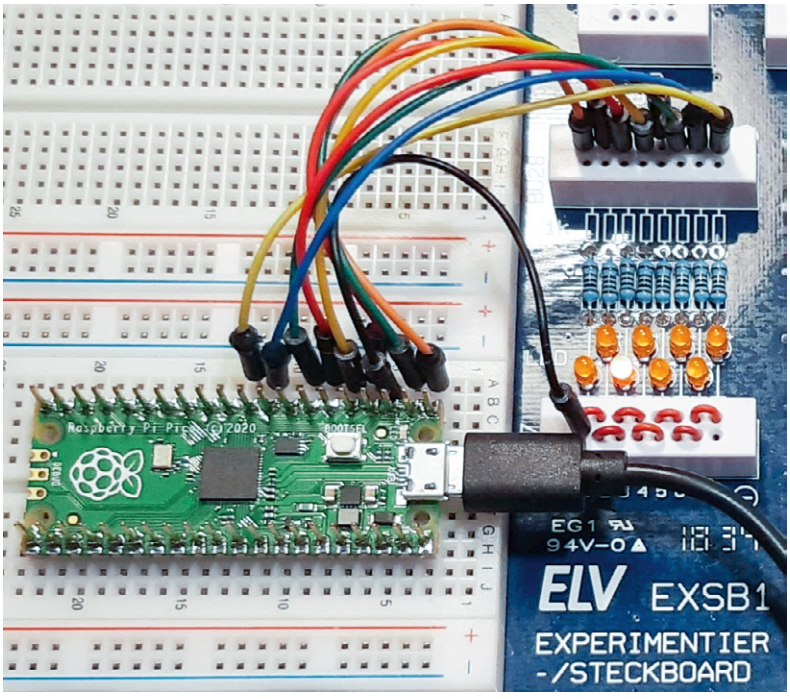


Bild 9: Lauflicht auf dem EXSB1-Board

Im Folgenden soll der ADC0 genutzt werden, um eine externe Spannung zu messen. Auf diese Weise entsteht ein vielseitig einsetzbares Computer-Digitalvoltmeter. Da die Eingänge des Pico nur mit maximal 3,3 V belastet werden dürfen, wird ein Spannungsteiler verwendet, der es erlaubt, Spannungen von bis zu 30 V zu erfassen. Höhere Spannungen sind potenziell lebensgefährlich und dürfen daher ohnehin nur mit speziell dafür zugelassenen Messgeräten erfasst werden. Bild 10 zeigt den zugehörigen Schaltplan.

Der Spannungsteiler sorgt dafür, dass die Eingangsspannungen auf 1/11 reduziert werden. Die beiden Schottky-Dioden SD1 und SD2 schützen den Analogeingang des Pico. SD1 leitet zu hohe Spannungen an die Versorgung V_{CC} = 3,3 V ab. SD2 schützt den Controller vor Verpolung.

Der Aufbau auf dem EXSB1-Board nutzt die vorhandenen 4-mm-Bananenbuchsen für den Anschluss von robusten Messkabeln. Der Aufbau wird so zum praxistauglichen Messsystem (Bild 11).

Das Programm zum Messsystem sieht so aus:

```
from machine import ADC, Pin
import time

conversion_factor=0.975*11*3.3/(1<<16) #ADC scaled to 16 bit
adc=ADC(Pin(26))

while True:
    print(adc.read_u16()*conversion_factor)
    time.sleep(1)
```

Von besonderem Interesse ist der Konvertierungsfaktor:

0.975*11*3.3/(1<<16)

Dieser ergibt sich aus:

- der Auslösung des ADC von 16 Bit: 2^16 = 65536
- der verwendeten Referenzspannung: 3,3 V
- dem Faktor des Spannungsteilers: 1 MΩ +100 KΩ / 100 KΩ = 11
- einer Kalibrationskonstante: 0,975

Die Kalibrationskonstante berücksichtigt insbesondere die Toleranzen der verwendeten Widerstände. Falls es also zu Abweichungen zwischen der tatsächlichen Spannung und dem angezeigten Messwert kommt, kann der Wert entsprechend angepasst werden.

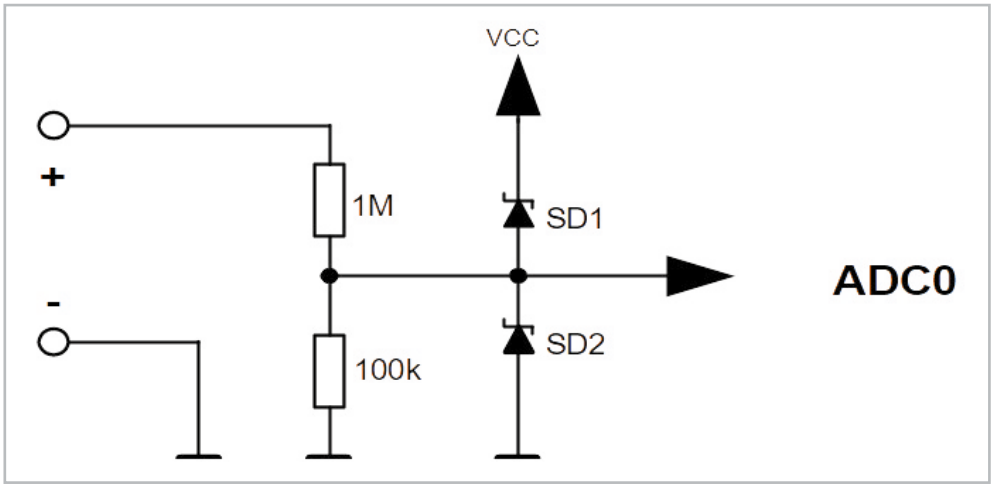


Bild 10: Schaltung zum Computer-Digitalvoltmeter

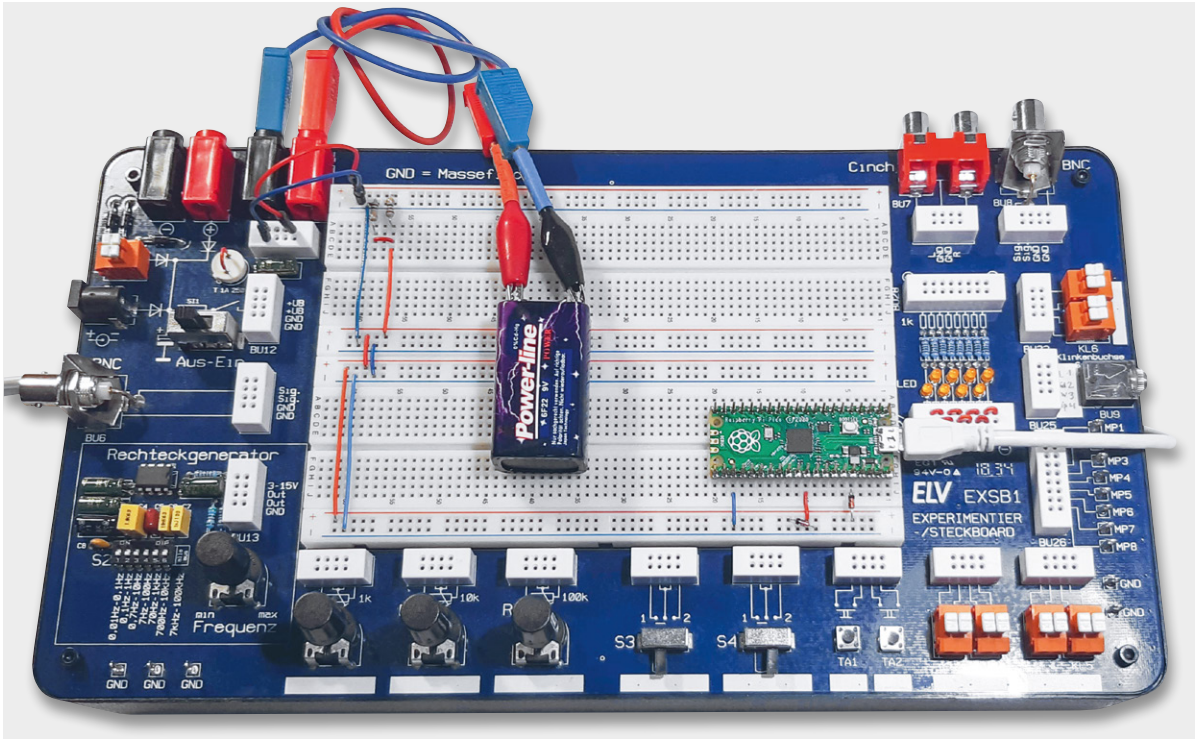


Bild 11: Praxistaugliches Messsystem

Die Ergebnisse können in Thonny abgelesen werden. Wird dort zusätzlich die Plotter-Funktion über

View ⇨ Plotter

aktiviert, werden die Werte zusätzlich auch grafisch dargestellt (s. Bild 12).

Ein universeller Datenlogger

Bislang war der Raspberry Pi Pico über den Micro-USB-Anschluss mit einem PC verbunden. Es gibt jedoch keinen Grund, den Controller stets „an der Leine" zu halten. Er ist ein voll funktionsfähiges System, das auch ohne die Unterstützung eines anderen Rechners eigenständig arbeiten kann.

Im Folgenden wird gezeigt, wie das interne Dateisystem des Pico verwendet werden kann, um Daten für den späteren Zugriff aufzuzeichnen. Das Board kann so an einem beliebigen Ort autark betrieben werden, ohne dass eine Verbindung zu einem Host-Computer erforderlich ist.

Auf diese Weise entsteht ein sehr praxistauglicher Datenlogger. Die Stromversorgung des Pico kann von einem beliebigen Micro-USB-Ladegerät oder einem Akku übernommen werden. Wird beispielsweise eine Powerbank (s. Bild 13) mit integrierter Solarzelle verwendet, kann das Pico-Board Daten über Wochen oder Monate hinweg zuverlässig autonom aufzeichnen.

Das Dateisystem des Pico entspricht in seiner Funktion der microSD-Karte eines klassischen Raspberry Pi bzw. der Festplatte oder dem SSD-Laufwerk im Laptop oder PC. Es bietet eine nicht flüchtige Speichermöglichkeit, d. h., dass alles, was abgelegt wurde, auch erhalten bleibt, selbst wenn die Stromversorgung des Controller-Boards unterbrochen wird.

In der Shell der Thonny-IDE kann über

```
file = open("test.txt", "w")
```

eine Datei namens test.txt zum Schreiben („write") geöffnet werden.

Bild 13: Autarker Datenlogger mit Solar-Powerbank

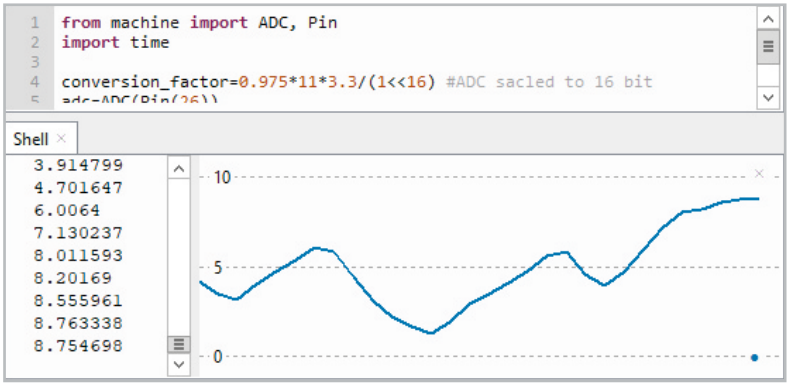


Bild 12: Numerische und grafische Darstellung von Spannungswerten

Mit

```
file.write("Hallo, Pico!")
```

werden zwölf Datenwerte in die Datei geschrieben. Zur Bestätigung wird die Zahl Zwölf in der Konsole ausgegeben.

Wenn in eine Datei geschrieben wurde, muss diese über

```
file.close()
```

auch wieder geschlossen werden. Nur dann ist sichergestellt, dass die Daten tatsächlich im Dateisystem gespeichert sind. Über das Öffnen-Symbol in der Symbolleiste von Thonny kann die Datei wieder geöffnet und angezeigt werden.

Mit „File ⇨ Save as" wird die Datei auf einen Rechner kopiert.

Mit dem folgenden Programm (s. Temperature_Logger_Internal.py):

```
import machine, utime

sensor_temp=machine.ADC(machine.ADC.CORE_TEMP)
conversion_factor=3.3/(1<<16)
file = open("temperatures.txt", "w")

while True:
    reading = sensor_temp.read_u16()*conversion_factor
    temperature=round(30-(reading-0.71)/0.0017,1)
    print(temperature)
    file.write(str(temperature)+ "\n")
    file.flush()
    utime.sleep(1)
```

zeichnet der Pico automatisch einmal pro Sekunde die interne Prozessor-Temperatur auf. Wenn der Prozessor nicht allzu stark belastet wird, stimmt dieser Wert relativ gut mit der Umgebungstemperatur überein. So erhält man einen komplett autonomen Temperatur-Datenlogger, ohne dass man auch nur ein einziges externes Bauelement an den Pico anschließen müsste.

Natürlich können auch weitere Sensoren mit dem Pico verbunden werden, sodass dann auch Helligkeitswerte, Druck oder Gaskonzentrationen etc. aufgezeichnet werden können.

Das Dateisystem des Pico funktioniert auch, wenn er nicht mit einem PC verbunden ist. Für die Verwendung ohne angeschlossenen Computer, also im sogenannten „Headless-Betrieb" muss das Datenlogger-Programm unter dem speziellen Dateinamen „main.py" gespeichert werden. Die Datei main.py wird bei jedem Einschalten oder Zurücksetzen des Controllers automatisch ausgeführt.

Ist der Datenlogger später wieder mit einem PC verbunden, muss zunächst die Aufzeichnung über das rote „Stopp"-Symbol angehalten

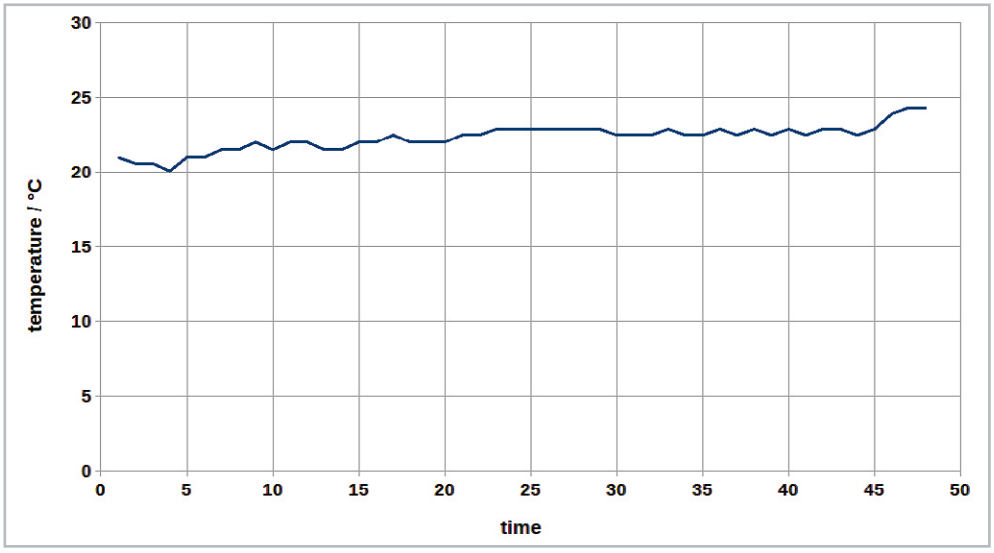


Bild 14: Daten aus dem Logger in einer Tabellenkalkulation

werden. Dann kann die aufgezeichnete Datei – in unserem Fall temperatures.txt – zum Rechner übertragen werden. Die Daten können dort z. B. mit einer Tabellenkalkulation dargestellt und weiterverarbeitet werden (s. Bild 14).

Anwendungen in C: Ein Spannungswächter

Als Anwendungsbeispiel für die C-Programmierung des Pico soll im Folgenden das Voltmeter und die Bargraph-Anzeige zu einem Spannungswächter kombiniert werden.

Zuvor muss der Raspberry Pi Pico aber erst einmal der Arduino-IDE bekannt gemacht werden. Dazu muss für das später gewählte Paket unter „Datei ⇨ Voreinstellungen" als zusätzliche URL folgende eingetragen werden (Bild 15):

https://github.com/earlephilhower/arduino-pico/releases/download/➥
global/package_rp2040_index.json

Nach dem Eintragen der URL wird das Fenster mit OK bestätigt. Will man das alternative Arduino-Paket für den Pico nutzen (s. u.), ist dieser Schritt nicht notwendig. Hier ist der Pico W allerdings noch nicht enthalten.

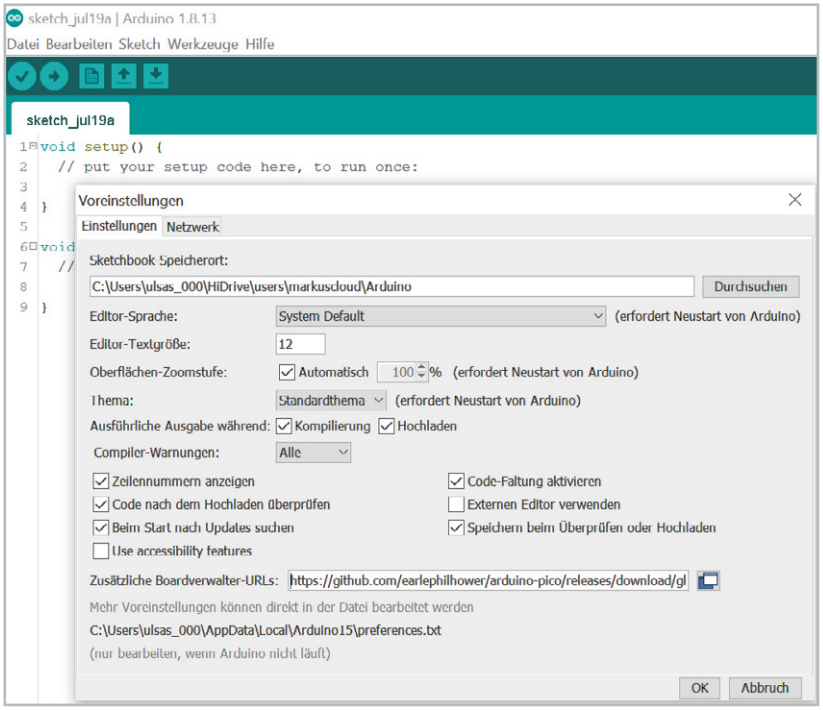


Bild 15: Eintragen der Boardverwalter-URL zur Integration des Raspberry Pi Pico in die Arduino-IDE

Zur Auswahl des entsprechenden Pakets geht man im Menü auf Werkzeuge ⇨ Board ⇨ Boardverwalter und gibt in das Suchfeld „pico" ein. Man kann zwischen zwei Paketen wählen: Arduino Mbed OS RP2040 Boards (von Arduino) oder Raspberry Pi Pico RP2040 (Bild 16). Bei dem letzteren Paket ist auch schon der Pico mit WLAN integriert. Durch einen Klick auf Installieren wird das Paket in die Arduino-IDE integriert.

Bei der Arbeit mit der Arduino-IDE ist lediglich zu beachten, dass während des Einsteckens des USB-Kabels in die Pico-USB-Buchse der BOOTSEL-Taster gedrückt werden muss. Der Pico muss dann noch als Board unter Werkzeuge ⇨ Board ⇨ Raspberry Pi RP2040 Boards ⇨ Raspberry Pi Pico ausgewählt werden (Bild 17). Analog gilt dieses Vorgehen auch für das Paket von Arduino (Arduino Mbed OS RP2040 Boards). Zudem muss u. U. noch ein USB-Treiber installiert werden.

Danach ist der Pico mit der IDE verbunden und die Sketche können genau wie beim Arduino UNO auf den Pico geladen werden. Wir testen dies mit dem Blink-Sketch aus den Beispielen. Dabei kann man beim ersten Anschließen des Picos den Sketch nach Auswahl des Boards einfach hochladen, da noch kein COM-Port für den Pico existiert. Dieser erscheint nach der ersten Übertragung eines Sketches.

Den Erfolg kann man wie immer durch eine blinkende LED auf dem Pico in Augenschein nehmen.

Arduino-Beispiel

Der Hardware-Aufbau basiert auf dem bereits in Bild 8 dargestellten Schaltbild. Für die Erzeugung der variablen Eingangsspannung wird das 100-KΩ-Potentiometer des EXSB1-Systems verwendet (Bild 18).

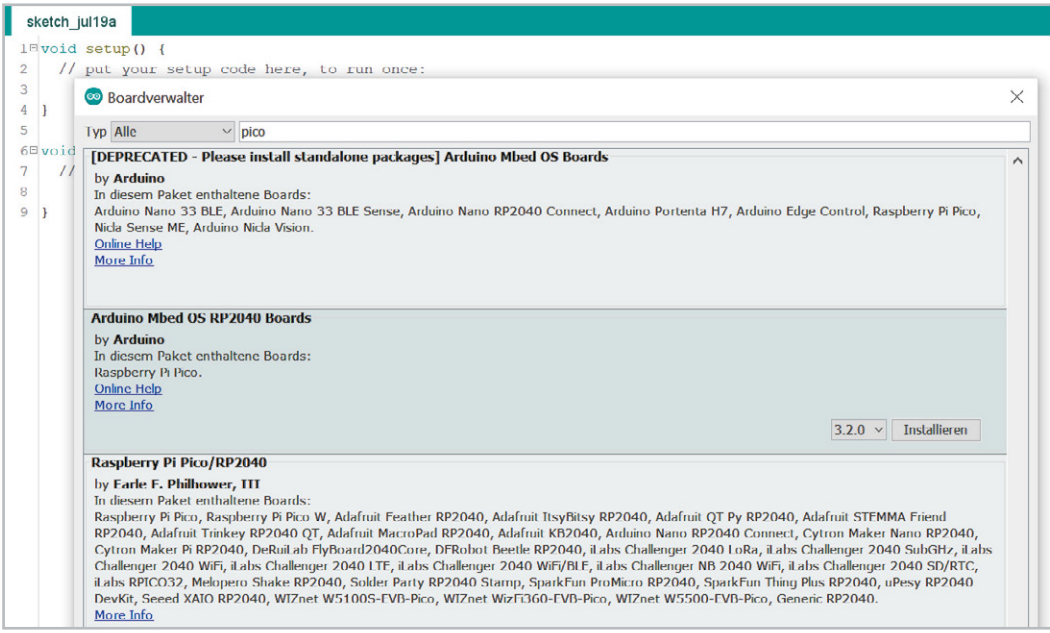


Bild 16: Auswahl des Paketes für den Pico in der Arduino-IDE

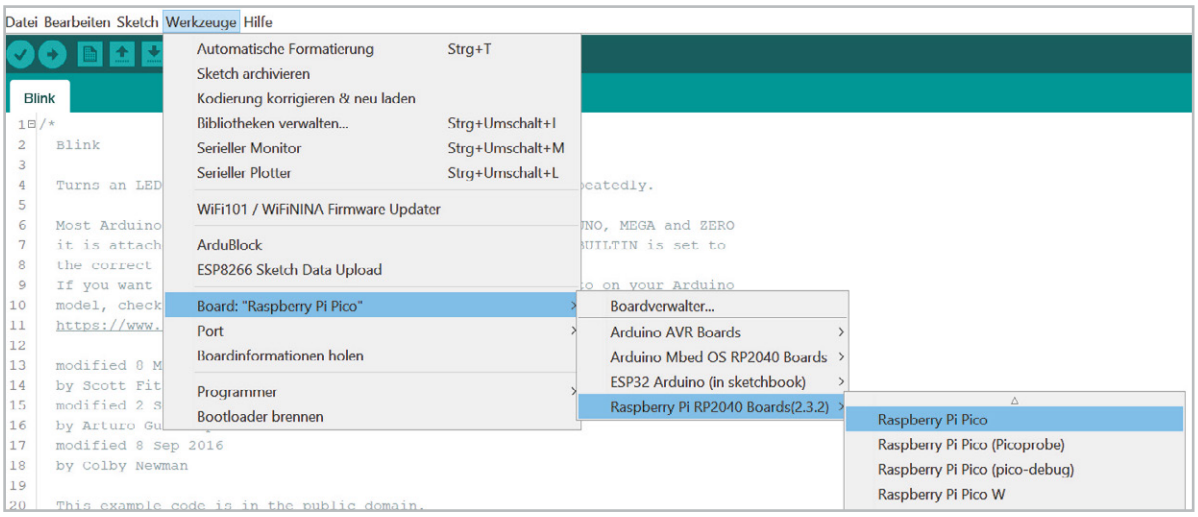


Bild 17: Auswahl des Picos

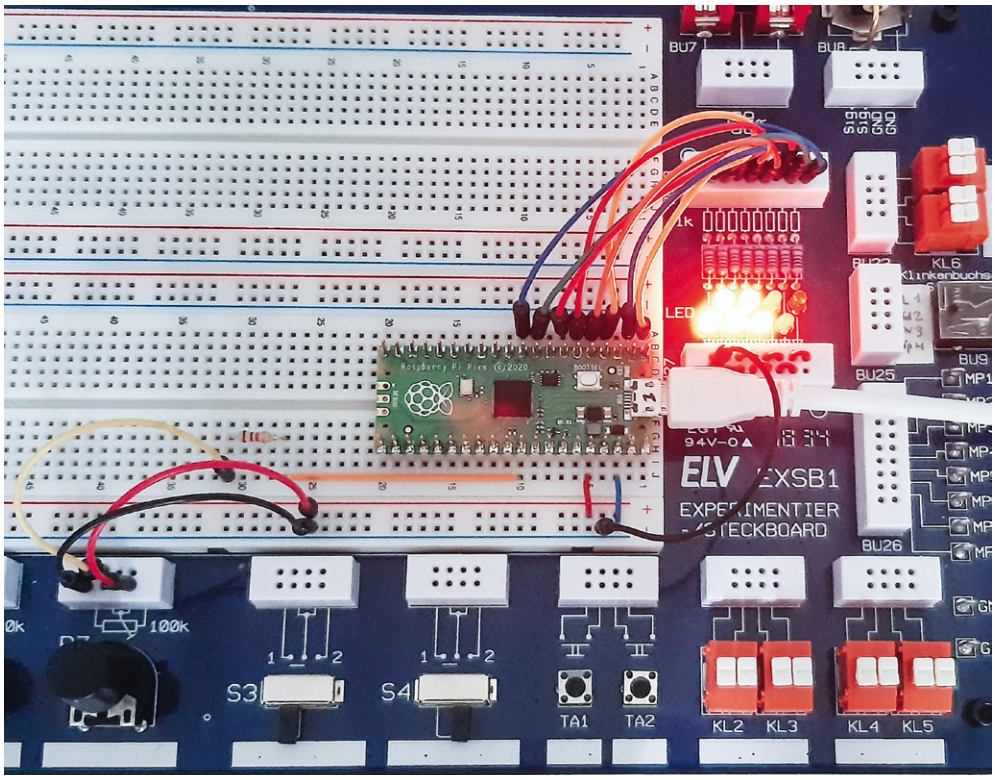


Bild 18: Test des Spannungswächters mit dem EXSB1-Board

Der zugehörige Arduino- bzw. C/C++-Sketch sieht so aus:

```
// analog_bargraph.ino
// Pico @ IDE 1.8.16

#define ADCmax 4095

const int analogPin=A0;   // the pin that the potentiometer is attached to
const int LED[]={2,3,4,5,6,7,8,9}; // pins connected to LEDs
const int LEDcount=sizeof(LED)/4; // the number of LEDs in the bar graph

void setup()
{ Serial.begin(115200);
  for (int i=0; i<LEDcount; i++)
    pinMode(LED[i], OUTPUT);
}

void loop()
{ int sensorReading = analogRead(analogPin);
  int ledLevel = map(sensorReading, 0, ADCmax, 0, LEDcount-1);
  Serial.println(ledLevel);
  for (int i=0;i<LEDcount;i++)
    { if (i==ledLevel) digitalWrite(LED[i], HIGH); // == dot, <= band
      else digitalWrite(LED[i], LOW);
    }
}
```

Nach dem Hochladen des Sketches leuchten die LEDs in Abhängigkeit von der Potentiometerstellung auf. Nun kann man noch die Spannungsteiler- und Schutzschaltung anstelle des Potentiometers anschließen, um einen praxistauglichen Spannungswächter aufzubauen.

Wenn der Pico wieder mit einer Powerbank betrieben wird, entsteht so ein vom PC unabhängiges Messsystem, mit dem beispielsweise die Bordspannung in einem Kraftfahrzeug überwacht werden kann. Alternativ könnte auch die Bordspannung selbst zur Versorgung des Picos verwendet werden. Dazu ist dann allerdings ein Kfz-taugliches 12-V-USB-Ladegerät erforderlich.

Ausblick

Nachdem in diesem Beitrag die Grundlagen der Programmierung des Pico ausführlicher beleuchtet wurden, soll im nächsten Artikel zu dieser Serie das Zusammenspiel des Controllers mit einem klassischen Raspberry Pi im Vordergrund stehen. Diese herkömmliche Version des Raspberry verfügt beispielsweise über keinerlei Analogeingänge. Eine direkte Messung analoger Werte ist damit nicht möglich. So können weder Photodioden noch NTCs oder Hallsensoren unmittelbar ausgelesen werden.

Mit dem Pico kann dieses Problem elegant gelöst werden. Er kann als sogenanntes „Frontend" problemlos vielfältige Messaufgaben übernehmen. Der klassische Raspberry Pi und der Pico werden so zum „idealen Duo", mit dem auch anspruchsvolle Messaufgaben gelöst werden können. ELV

Material	Artikel-Nr.
Raspberry Pi Pico	251905
Raspberry Pi Pico W	252796
EXSB1 Experimentierboard	153753

Für Kleinteile wie Widerstände oder Dioden können die Prototypenadapter-Sets (PAD 1–5) verwendet werden.

Weitere Infos

[1] Thonny-IDE - https://thonny.org/
[2] Download-Paket: Artikel-Nr. 253018

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links