Extra-Tastatur
für Videokonferenzen

Ein Hustenanfall oder der Partner läuft halb bekleidet während einer wichtigen Videokonferenz durch das Zimmer. Sie teilen versehentlich den Bildschirm, statt die *Mute*- und *Kamera*-Icons zu treffen. Dies ist alles vermeidbar mit einer speziellen Zusatztastatur für das Homeoffice. Wir zeigen, wie Sie solch eine Tastatur günstig und einfach mit dem Raspberry Pico bauen.

von Carsten Wartmann

Hatten Sie in einer Videokonferenz schon mal einen Hustenanfall und haben vor lauter Husten das kleine Icon für die Mikrofon-Stummschaltung mit der Maus nicht getroffen? Der Tastaturbefehl dafür ist kryptisch und muss mit beiden Händen eingetippt werden. Oder mussten Sie die Kamera schnell deaktivieren, weil ein weinendes Kind ins Zimmer stürmte? Hier wäre eine zusätzliche Tastatur ein Segen, mit großen und gut zu treffenden Tasten, die selbst die kryptischsten Tastenkürzel in Sekundenbruchteilen an eine Anwendung sendet.

Am Beispiel einer Erweiterungstastatur für *Microsoft Teams* zeigen wir Ihnen, wie man mit dem Raspberry Pico solch einen nützlichen Schnellschalter ganz leicht selbst baut und programmiert. In unserem Beispiel-Projekt verwenden wir die Tastenkürzel für die Funktionen wie *Hand heben*, *Video umschalten* und *Mikrofon umschalten*. Der Raspberry Pico wird in der leicht erlernbaren Sprache *Python* programmiert, damit ist eine Anpassung an andere Anwendungen leicht und wird später im Artikel erklärt.

Alles, was wir brauchen, ist ein Raspberry Pico mit *Circuit Python*, ein USB-Kabel, Tasten zum Anschluss an den Pico, ein Gehäuse und etwas Bastelzeit. Wer nicht löten mag, kann das Ganze auch auf einem Steckbrett aufbauen, hier muss man dann nur sehen, wie die Taster angeschlossen werden. Entweder man benutzt bereits mit Kabeln versehende Taster oder solche, die sich auf das Breadboard stecken lassen. Gerade für einen Test, ob man solch eine Tastatur braucht, ist diese Methode ideal, denn so sind die Bauteile und der Pico in Minuten wieder zu anderen Zwecken bereit.

Pico und Computer vorbereiten

Wir werden für die Konferenztastatur den Pico mit *Circuit Python* programmieren. Dies ist ein Python-Dialekt, der direkt auf dem Pico läuft. Für Circuit Python ist eine neue Firmware für den Raspberry Pico nötig. Keine Angst, wenn Sie jetzt eventuell an kryptische Befehle und kompliziert anzuschließende Programmier-Gadgets wie bei anderen Mikrocontrollern denken. Eine Firmware auf dem Pico wird per Drag & Drop installiert und dies ist so einfach wie eine Datei von einem Laufwerk auf ein anderes zu schieben. Laden Sie die Firmware von CircuitPython.org (wir verwenden die Stabile Version *adafruit-circuitpython-raspberry_pi_pico-de_DE-6.2.0.uf2*), Links in der Kurzinfo) herunter und speichern sie auf Ihrem Computer.

Stecken Sie den Pico bei gehaltener BOOT-Taste an den Rechner, warten Sie ein paar Sekunden und lassen den BOOT-Taster los. Der Pico erscheint im Boot-Mode immer

Kurzinfo

- **Spezial- und Zusatztastaturen mit Pico bauen**
- **Python-Modul USB-HID für Circuit Python auf Raspberry Pico nutzen**
- **Tasten anschließen an den Pico**

Checkliste

Zeitaufwand:
etwa eine Stunde

Kosten:
ab 5 Euro

Material

- **Raspberry Pico**
- **mehrere Tasten** aus der Bastelkiste, oder Marquardt 1005.0507
- **USB-Micro-Kabel**
- **Gehäuse** Blech, Holz, Plastik, 3D-Druck, Lasercut …

Werkzeug

- **Lötkolben, Lötzinn**
- **oder Breadboard**
- **Bohrer, Säge, 3D-Drucker**
  je nach Gehäusevariante

Mehr zum Thema

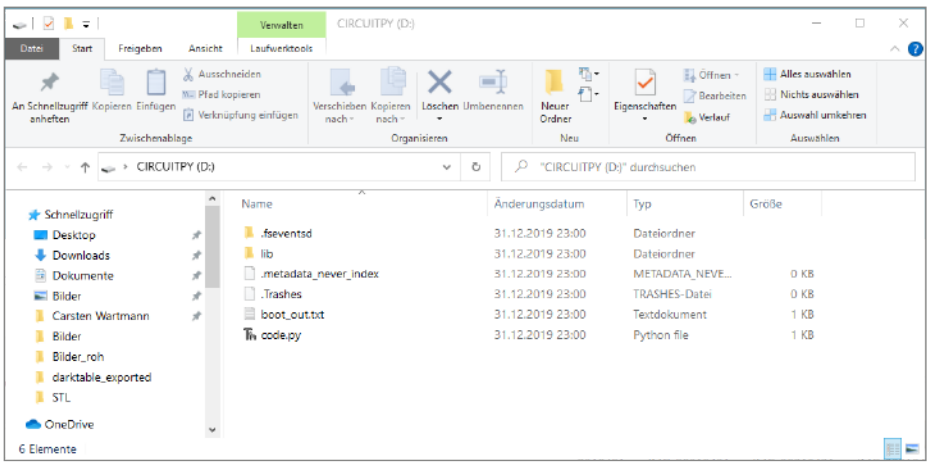
- Peter König, Ende einer Ära, Make 1/21, S. 3
- Andreas Perband, Raspberry Pi Pico, Make 1/21, S. 8

Alles zum Artikel im Web unter
make-magazin.de/x44x

Raspberry Pico

Der RP2040-Prozessor im Raspberry Pico unterstützt USB-HID, also *Human Interface Devices*, das sind unter anderem Tastaturen und Mäuse. Das bedeutet in diesem Fall allerdings nicht, dass eine Tastatur oder Maus an den Raspberry Pico angeschlossen wird, sondern, dass sich der Pico an Windows-, Linux-, Mac- oder Android-Systemen als Maus oder Tastatur ausgibt.

Der Raspberry Pico ist natürlich fast schon Overkill für solch eine einfache Anwendung und auch andere Mikrocontroller-Boards können sich per USB-HID als Tastaturen ausgeben. Die Kombination aus einfacher Programmierung mit Python, echtem USB, günstigem Preis und weiter Verbreitung wie beim Pico ist aber nicht so leicht zu finden.



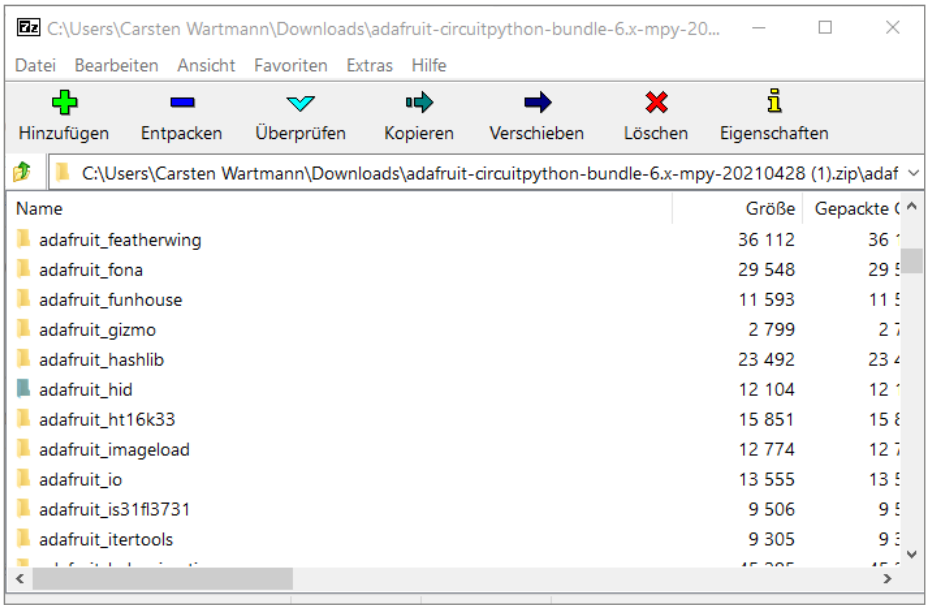
1 Das Picodrive im Dateibrowser

Problembehebung

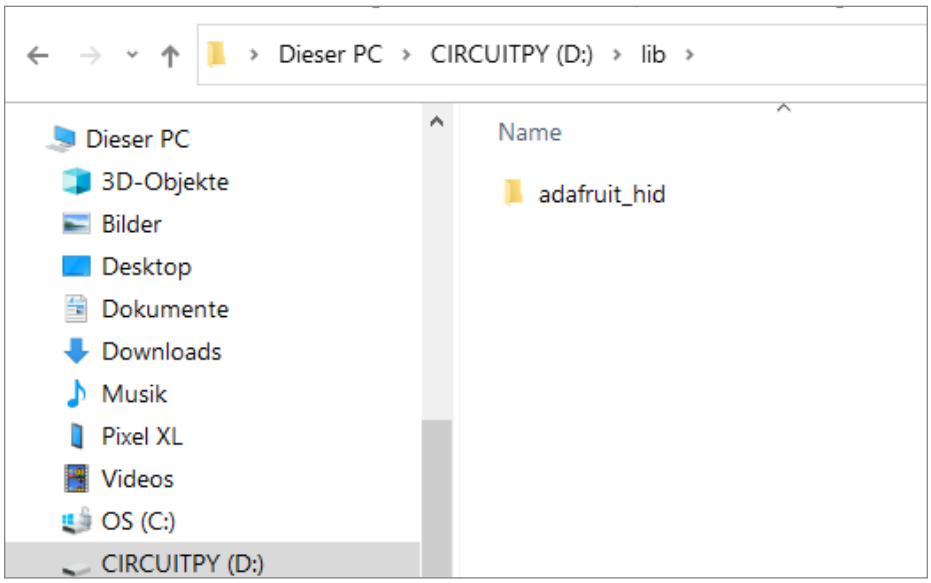
Sollte sich der Pico nicht als Laufwerk mounten lassen, versuchen Sie ein anderes USB-Kabel, am besten eines, mit dem Sie schon erfolgreich Daten auf ein Mobiltelefon oder ähnliches übertragen haben. Weiterhin blocken einige Virenscanner das Picodrive.

Das Picodrive sollte immer per System ausgeworfen werden. Zieht man einfach das USB-Kabel vom Rechner ab, kann es zu einem Datenverlust kommen.

Wenn Ihr Pico sich nachhaltig seltsam verhält, können Sie als letztes Mittel probieren, die Firmware *flash_nuke.uf2* (Link in Kurzinfo) zu installieren. Dabei gehen alle Dateien und Programme verloren, die auf dem ursprünglichen Laufwerk gespeichert waren! Nach dem „Nuking" können Sie, wie im Artikel beschrieben, eine neue Firmware aufspielen. Ihre Programme sollten Sie immer auch auf einem anderen Datenträger am Computer speichern.



2 *7Zip*-Archivmanager



3 lib-Ordner auf Picodrive

als Laufwerk mit dem Namen *RPI-RP2* am Computer. Sollten Sie Schwierigkeiten haben, finden Sie im Kasten *Problembehebung* Lösungsvorschläge.

Kopieren Sie nun die heruntergeladene Firmware (*.uf2*) auf das Laufwerk *RPI-RP2*, am einfachsten per Drag & Drop oder Copy & Paste im Dateimanager Ihres Computers. Danach bootet der Pico und das Circuit-Python-Laufwerk mit dem Namen *CIRCUITPY* (im Artikel ab hier als *Picodrive* bezeichnet) erscheint als Laufwerk am Rechner 1. Auf diesem Laufwerk befinden sich schon einige Dateien, hierbei ist *code.py* die Python-Datei, die beim Starten des Pico ausgeführt wird. Mehr dazu später.

Die Circuit-Python-Firmware, die wir gerade installiert haben, enthält nur die wichtigsten Komponenten, um den Pico mit Python zu programmieren. Für spezielle Geräte und Anwendungen braucht man weitere Module. Ein ZIP-Archiv (*adafruit-circuitpython-bundle-6.x-mpy-20210428.zip*) mit allen zur Zeit unterstützten Modulen für Sensoren und Geräte kann man bei Adafruit herunterladen (alle Links in der Kurzinfo). Öffnen Sie dieses Archiv nach dem Download mit einem Archivprogramm 2 oder dem Dateimanager Ihres Betriebssystems.

In dem Archiv sind mehrere Unterordner enthalten. Klicken Sie sich bis in den *lib*-Ordner durch und scrollen darin zu dem Ordner *adafruit_hid*. Kopieren den gesamten Ordner in den *lib*-Ordner auf dem Picodrive 3. Adafruit hat auch immer eine gute Dokumentation und alternative Downloadmöglichkeiten zu allen Modulen, zum Beispiel über Adafruits GitHub.

Als Nächstes werden wir *Thonny* als Entwicklungsumgebung (IDE) installieren. Laden Sie die für Ihren Computer passende Thonny-Version von Thonny.org. Für diesen Artikel benötigen Sie mindestens Version 3.3.7! Für Windows gibt es einen Installer (*.exe*), die Mac-Version kommt als *.dpk*. Im Falle von Windows klicken Sie das .exe doppelt an und folgen den Anweisungen des Installers. Auf dem Mac benutzen sie den üblichen Weg zur Installation von .dpk-Paketen. Beim Installationsvorgang kann, je nach Aktualität der Thonny-Version, eventuell der Viruswarner oder *Windows Defender* ansprechen, dies darf ignoriert werden. Für erfahrende Anwender stehen auch portable Versionen (nur entpacken) und der Quellcode zur Verfügung.

Wenn Sie nun Thonny starten, dann sollte es sich wie in der Abbildung 4 darstellen.

Falls der Pico nicht am Rechner steckt, dann stecken Sie Ihn bitte wieder an, aber ohne den BOOT-Taster zu drücken! Damit Thonny mit dem Circuit-Python-Pico zusammenarbeitet, ist der Interpreter *Circuit Python (generic)* im Menü *Extras/Optionen...* 5 im *Interpreter*-Tab einzustellen 6. Damit weiß dann Thonny, wie er mit dem Board sprechen muss.

Das Board sollte sich nun mit *Adafruit CircuitPython 6.2.0 on 2021-04-05; Raspberry Pi Pico with rp2040* im unteren Teil von Thonny melden. Testen Sie nun einmal Thonny und Circuit-Python auf dem Pico: Im unteren Teil des Thonny-Fensters, in der *Kommandozeile*, können Sie Python Befehle eingeben, die direkt auf dem Pico ausgeführt werden. Versuchen Sie doch einmal `1+1` oder auch `print('Hallo Pico!')` (7). Jetzt sollte dem Erfolg nichts mehr im Wege stehen und Sie können nun die Hardware aufbauen.

Tasten anschließen

Es gibt tausende mechanische Bauformen für Taster (8). Welche Sie letztlich verwenden, ist Ihnen überlassen. Die meisten Taster sind Schließer, das heißt, der Stromkreis wird bei Druck auf den Taster geschlossen. Ohne Modifikation am Programm können wir auch nur diese in unserem Projekt gebrauchen. Für die ersten schnellen Experimente empfiehlt sich ein Aufbau auf einem Breadboard, einem Steckbrett, auf dem ein mit Pinleisten versehener Pico und ein paar Taster Platz haben.

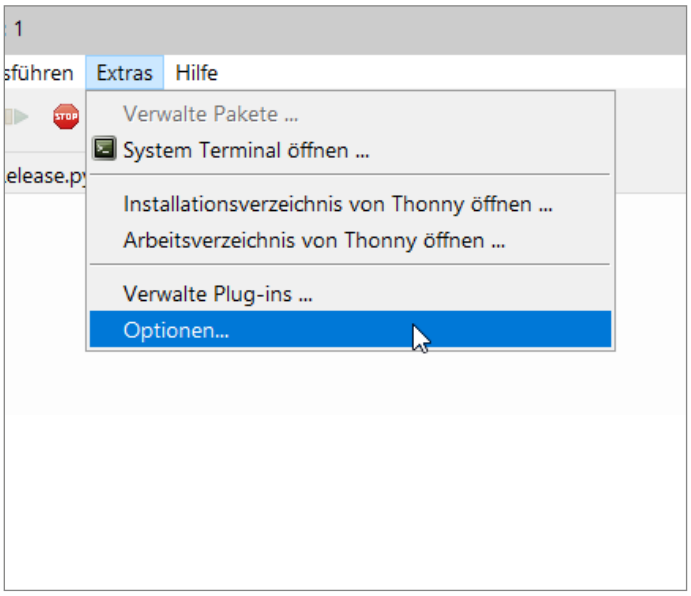
Die Schaltung ist in (9) dargestellt. Jeder Taster ist an einem Anschluss mit der positiven Versorgungsspannung (3,3V) des Mikroprozessors verbunden, der andere Anschluss führt jeweils zu einem Port (GPIO) am Pico. Diese GPIOs können als Aus- und Eingänge per Programmierung definiert werden. Die Pins sind leider nur auf der Rückseite des Pico komplett beschriftet. Bauen Sie die Schaltung am besten zuerst auf einem Breadboard auf. Aber auch Klemmen oder Löten ist natürlich möglich. Achten Sie darauf, keine Pins am Pico versehentlich zu verbinden und arbeiten Sie nur an einem nicht am USB-Port angeschlossenen Pico.

(4) Die Thonny-IDE

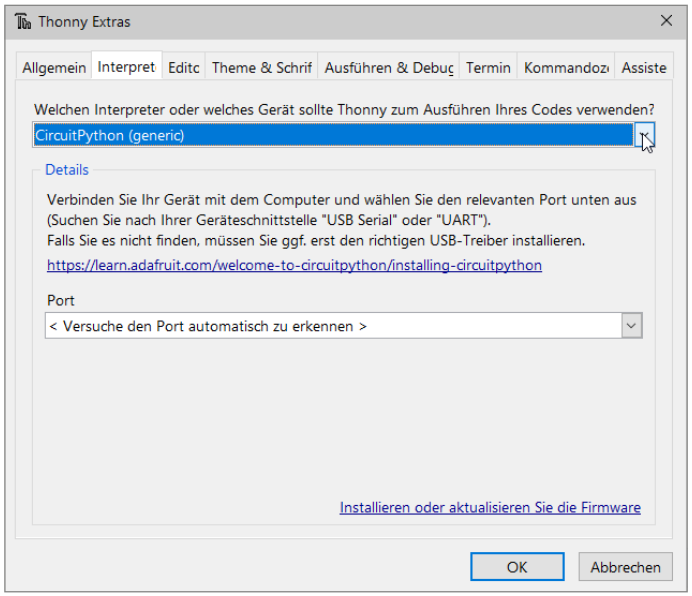
Benutzen Sie das *Lade*-Icon oder *Datei/Öffnen...* und wählen dann *Dieser Computer*, navigieren dann im Dateifenster zum Picodrive und laden *code.py*. Eventuell in *code.py* bereits vorhandenen Code ersetzen Sie mit dem Code aus Listing (10) *Drei_Tasten.py*. Natürlich gibt es den Code auch über die Links in unserer Kurzinfo. Sie können diesen direkt herunterladen und dann in den Editor hineinkopieren. Ich empfehle Einsteigern, die das Programmieren erlernen möchten, kurze Programme auch tatsächlich einzutippen, so lernt man die Befehle und Strukturen einer neuen Programmiersprache schneller kennen und verinnerlicht sie. Die Einrückungen sind wichtig für Python, benutzen Sie bitte die Tab-Taste, um sie gleichmäßig zu erstellen.

Speichern Sie jetzt *code.py* auf dem Picodrive und starten es dann mit dem *Run*-Icon oder F5. Der Pico sendet jetzt beim Drücken der jeweiligen Taste Shortcuts an den Computer. Im Skript sind dies die Tastaturkommandos für *Microsoft Teams*: *Hand heben* (*Strg+Umschalt+K*), *Video umschalten* (*Strg+Umschalt+O*) und *Mikrofon umschalten* (*Strg+Umschalt+M*).

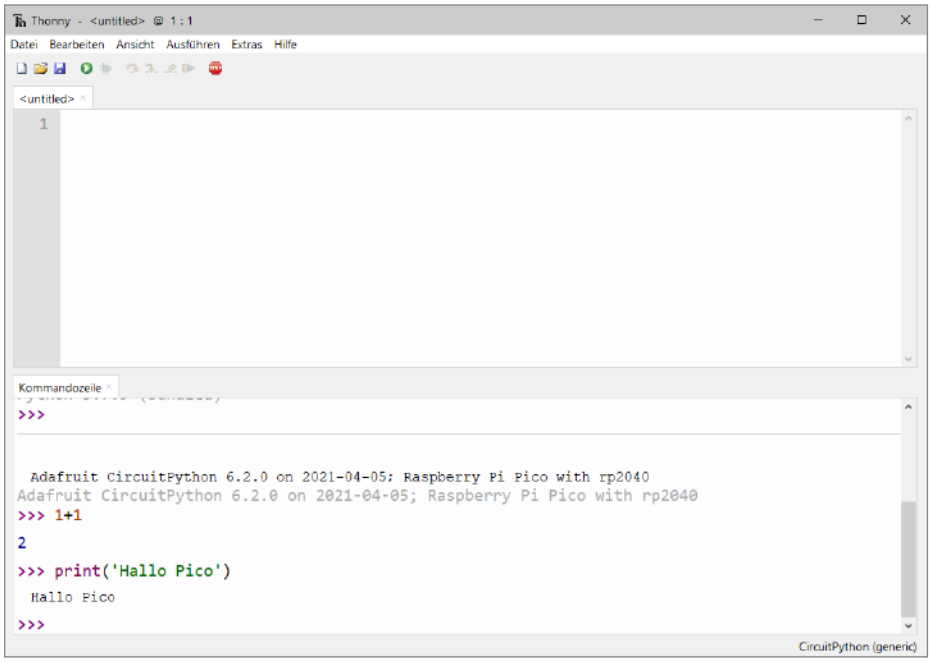
Möchten Sie eine andere Anwendung steuern, muss das Programm geändert und auf dem Pico gespeichert werden. Welche Tastaturkürzel eine Anwendung versteht, bekommt man über deren Anleitung heraus. Oft sind diese Kürzel auch in den Menüs angezeigt. Besonders gut geeignet sind Programme, in denen sich neue Tastenkommandos für alle Befehle und Menüs vergeben lassen. Hier kann man dann lange Kombinationen verwenden, die garantiert nicht mit anderen Funktionen kollidieren. Gut geeignet sind auch die Keycodes der üblicherweise nicht auf Tastaturen vorhandenen Funktionstasten *F13* bis *F24*. Hier ist die Gefahr minimal, mit anderen Programmen in Konflikt zu kommen.



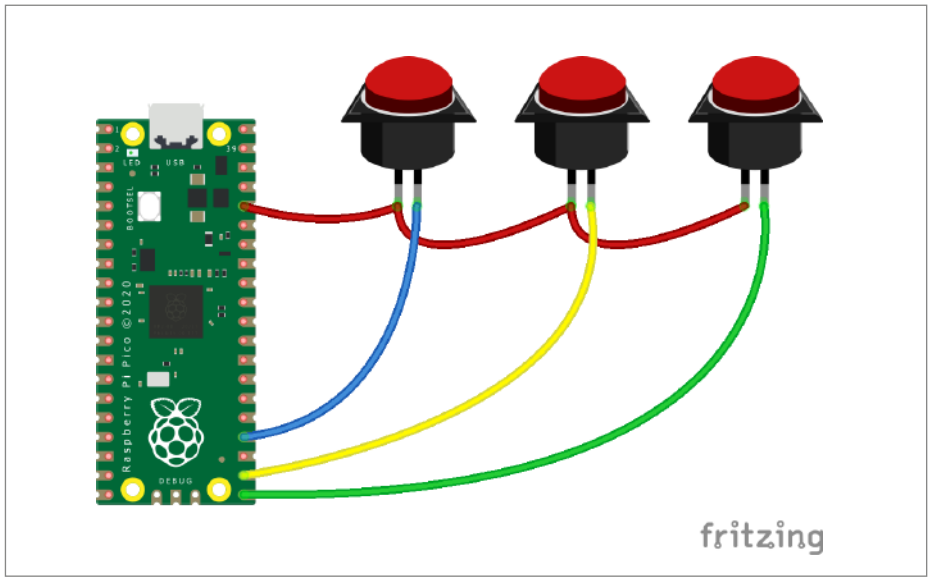
(5) Die wichtigste Option ist …



(6) … welcher Python-Interpreter benutzt werden soll.



7 Erste Lebenszeichen des Pico



9 Prinzipschaltbild für drei Taster

Haben Sie die Informationen gesammelt, suchen Sie die Keycode-Namen aus unserer Tabelle 11 heraus oder schauen online in der Dokumentation des Adafruit-HID-Moduls nach (Link in Kurzinfo). Die relevanten Zeilen im Listing 10 sind 19, 22 und 25 für die drei Tasten. Im Funktionsaufruf `kbd.send()` listen Sie, per Komma getrennt, die gleichzeitig zu drückenden Tasten auf. Möchten Sie zum Beispiel das Tastenkürzel für die erste Taste auf *Videoanruf annehmen* (*Strg+Umschalt+A*) ändern, dann müssen Sie Zeile 19 in:

```
kbd.send(Keycode.CONTROL,Keycode.
SHIFT,Keycode.A)
```

ändern.

Das Skript *code.py* wird nun immer ausgeführt, wenn der Pico angesteckt wird oder neu startet. Der Pico agiert jetzt auf Windows, Mac, Linux und sogar auf Android-Geräten mit OTG-Adapter als Tastatur und kann so Ihre Videokonferenz angenehmer und sicherer machen.

Gehäuse

Als Gehäuse eignet sich eigentlich alles, was der Maker-Schrank so hergibt: Gekaufte Fertiggehäuse 12, Holz- oder Blechkisten, Lego oder Fischertechnik, ausgeschlachtete Geräte mit Tasten, alte Telefone mit Tasten, Gehäuse aus dem Laser-Cutter oder 3D-Drucker – es muss nur der Pico hineinpassen und das USB-Kabel muss herausführen.

Das Gehäuse aus dem Aufmacherbild und 13 finden Sie auf unserem GitHub, es ist für besonders robuste Taster aus Arcade-Joysticks konstruiert. Die transparenten Tastenkappen wurden mit *Prusament-PVB* gedruckt (siehe Kurzvorstellung auf Seite 125). Dann wurden die Tastenkappen mittels Isopropanol geglättet und durchsichtig gemacht. Das Gehäuse ist modular aufgebaut, so dass Sie die Tasten vielfältig kombinieren können, was Farben, Kappen und Einleger angeht. Die Tasten bieten unten Bohrungen für 3mm LEDs, falls Sie die Tasten beleuchten möchten. Weiterhin gibt es auf unserem GitHub eine Blender-Datei, mit der recht einfach das Gehäuse für andere Tastenzahlen abgewandelt werden kann.

8 Einige Taster aus der Bastelkiste

Python-Syntax

Codeblöcke, wie der Inhalt von Schleifen oder Funktionsdeklarationen, werden in Python durch eine Einrückung, bestehend aus Leerzeichen, gekennzeichnet und nicht wie in anderen Sprachen durch Klammern oder ähnliches. Dies dient der Lesbarkeit, hat aber auch seine Tücken. Wenn einem einmal ein Leerzeichen zu viel in die Einrückung rutscht, beschwert sich Python mit einem Syntax-Error. Benutzen Sie am besten die Tab-Taste, sie lässt den Cursor zur nächsten Einrücktiefe springen und erspart umständliches Zählen von Leerzeichen. Mit Tab bzw. Shift-Tab können auch ganze selektierte Bereiche in der Einrücktiefe geändert werden.

Aber auch gängige 3D-Druck-Plattformen bieten unter dem Suchbegriff „Streamdeck" viele Gehäuse zum Ausdruck an.

Sie wollen es genauer wissen?

In diesem Abschnitt gehe ich etwas tiefer in die Materie und erkläre einige Details dieses Projekts genauer. Mit diesem Wissen können Sie dann dieses Projekt erweitern oder abwandeln. Ich werde anhand kleiner Beispiele die Funktion der einzelnen Komponenten genauer erläutern. Die GPIOs bleiben gleich, sodass Sie die oben gebaute Hardware weiter verwenden können. Für die Beispiele in diesem Abschnitt verwenden wir nicht *code.py* (Kasten *Programmstruktur auf dem Picodrive*), sondern speichern die Programme unter einem anderen Namen ab, damit man sie später als Referenz und Grundlage für neue Programme benutzen kann. Damit das *code.py* für die Teams-Tasten nicht dazwischen funkt, sollten Sie es für die weiteren Experimente auf dem Picodrive umbenennen oder den Inhalt löschen.

Die Struktur des Programms für unsere Pico-Tastatur ist simpel. Schauen Sie noch einmal ins Listing ⑩. In den Zeilen 1-6 werden

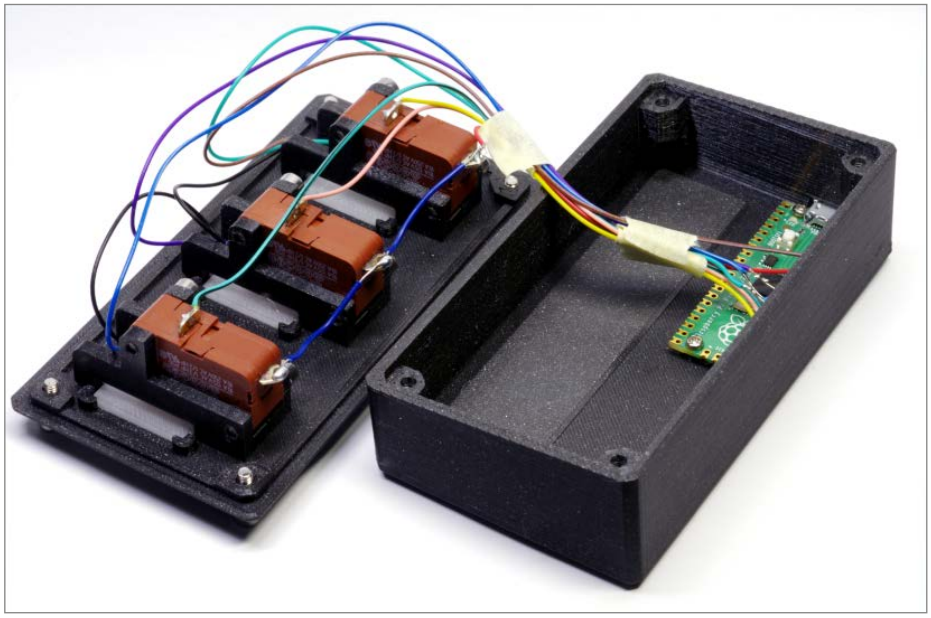
⑩ Drei_Tasten.py

```
01 import board
02 import digitalio
03 import time
04 import usb_hid
05 from adafruit_hid.keyboard import Keyboard
06 from adafruit_hid.keycode import Keycode
07
08 kbd = Keyboard(usb_hid.devices)
09
10 taste1=digitalio.DigitalInOut(board.GP16)
11 taste1.switch_to_input(pull=digitalio.Pull.DOWN)
12 taste2=digitalio.DigitalInOut(board.GP17)
13 taste2.switch_to_input(pull=digitalio.Pull.DOWN)
14 taste3=digitalio.DigitalInOut(board.GP18)
15 taste3.switch_to_input(pull=digitalio.Pull.DOWN)
16
17 while True:
18     if taste1.value:
19         kbd.send(Keycode.CONTROL,Keycode.SHIFT,Keycode.K)
20         time.sleep(0.2)
21     if taste2.value:
22         kbd.send(Keycode.CONTROL,Keycode.SHIFT,Keycode.O)
23         time.sleep(0.2)
24     if taste3.value:
25         kbd.send(Keycode.CONTROL,Keycode.SHIFT,Keycode.M)
26         time.sleep(0.2)
```

⓫ Beispiele für Tastencodes

Keycode	Taste
A,B,C…	Entsprechende Buchstabentaste, X und Y vertauscht!
ONE, TWO, THREE...	Ziffern
END	Ende-Taste
CAPS_LOCK	Die ungeliebte Shift-Lock-Taste
COMMAND	Mac-Command-Taste
LEFT_CONTROL, CONTROL	Strg-Taste
ESCAPE	Esc-Taste
F1-F24	Funktionstasten
SHIFT	Shift-Taste, Umschalt-Taste
GUI	Windows-Taste, Command (Mac), Meta
PRINT_SCREEN	Druck-Taste
SPACEBAR	Leertaste

⓬ Zusatztastatur in Fertiggehäuse aus dem Elektronikbedarf



⓭ Gehäuse in 3D-Druck

die benötigten Module (ähnlich wie Bibliotheken in anderen Sprachen) in das Programm geladen (*import*). Die Module *board* und *digitalio* stellen die Funktionen zur Behandlung der Tasten an den GPIO-Anschlüssen des Pico. Das *time*-Modul liefert eine Schlaffunktion, in der der Prozessor wartet oder andere Dinge machen kann. Die Module *usb_hid* und die *adafruit_hid* Module stellen die USB-Tastatur-Funktionen bereit.

In Zeile 8 wird das `Keyboard`-Objekt (Objekte in Python sind mehr als einfache Variablen) erstellt und in der Variablen `kbd` für den späteren Zugriff gespeichert.

In den Zeilen 10-11 wird eines der drei Tasten-Objekte mithilfe der `digitalio.DigitalInOut()` Funktion auf den GPIO 16 eingestellt und in dem Objekt `taste1` gespeichert. Dieses Objekt stellt nun eine Funktion `switch_to_input()` bereit, mit der der Port auf Eingang geschaltet wird. Der Parameter `pull=digitalio.Pull.DOWN` sorgt für eine ordnungsgemäße Funktion in unserer Schaltung, bei der 3.3V über den gedrückten Schalter mit dem GPIO verbunden werden. Dazu wird ein interner Pulldown-Widerstand (siehe Kasten *Widerstände, Pulldown, Pullup?*) aktiviert. In den Zeilen 12-15 geschieht das gleiche für die anderen zwei Tasten.

In Zeile 17 wird eine `while`-Schleife begonnen, die durch die Bedingung `True` eine Endlosschleife ist. Solch eine Schleife oder jedes andere Programm können Sie übrigens beenden, indem Sie das *Stop*-Icon in Thonny benutzen oder *Strg-C* in der Kommandozeile von Thonny drücken. Eine Besonderheit im Zusammenhang mit dem Stop-Icon ist, das dann auch wieder *code.py* ausgeführt wird, was bei der Entwicklung zu unerwünschten Phänomenen führen kann (s.o.).

Die Einrückung im Schleifenkörper sagt dem Python-Programm, dass dies als ein Block von Befehlen anzusehen ist und zusammen gehört. Die Einrückung nach einer `if`-Anweisung ist nochmals eine Ebene tiefer.

Ab Zeile 18 folgen dann drei `if`-Abfragen, die die jeweilige Taste auf einen Tastendruck prüfen und in diesem Falle den eingerückten Block ausführen. Dazu wird das Tasten-Objekt, `taste1.value` abgefragt, das bei gedrückter Taste `True` ist und so den Block in der `if`-Anweisung ausführt. `kbd.send()` sendet dann, wie vorhin im Projekt gezeigt, die entsprechenden Tastencodes. Danach wartet das Programm 0,2 Sekunden.

Die restlichen Zeilen erledigen die Abfrage für die übrigen Tasten und dann wird die Schleife ab Zeile 17 erneut ausgeführt.

Der Ansatz aus Listing ⓾ wird spätestens bei mehr als drei Tasten unübersichtlich. Aber einmal erstellt und dann in die Pico-Tastatur eingebaut, ist das eventuell die am schnellsten zum Ziel führende, wenn auch nicht die eleganteste Methode. Im GitHub finden Sie eine elegantere Version für beliebig viele Tasten.

Mediensteuerung und Maus

Neben dem Senden von Tastendrücken an den Host-Computer bietet das HID-Modul auch eine *Consumer-Control*- und eine Mausteuerung an. *Consumer Control* ist eine HID-Methode, um Multimedia-Programme per USB zu steuern. Diese beinhaltet Befehle, um die Lautstärke zu ändern, den Ton auszuschalten (Mute), zwischen Titeln zu springen und verschiedenes mehr. Dabei muss die abspielende Anwendung auf dem Host-Computer übrigens nicht im Vordergrund laufen.

Die Befehle und Codes sind nicht in *adafruit_hid.keyboard* gekapselt, sondern in einer eigenen Klasse (siehe Listing 14), aber der Aufbau ist analog wie bei den Keycodes. Wird dieses Programm ausgeführt, wird die Lautstärke auf null gesetzt. Bei erneutem Aufruf des Programms wird die alte Lautstärke wieder hergestellt 15.

Bisher haben wir nur mit kurzen Tastendrücken gearbeitet. Möchten Sie eine oder mehrere Tasten halten, so gibt es dafür die `press()`- und `release()`-Funktionen. Werden `kbd.press(Keycode.A)` und `kbd.release(Keycode.A)` direkt hintereinander ausgeführt, sind sie das Äquivalent zu `kbd.send(Keycode.A)`. Interessant wird es, wenn Sie zwischen den Befehlen eine Zeitverzögerung einbauen. Der Code in Listing 16 drückt die A-Taste zwei Sekunden lang und dies führt zu einer Tastenwiederholung und vielen „aaaaaaa" in der gerade aktiven Anwendung. Hier lauert auch eine Gefahr, denn wenn durch einen Programmfehler (zum Beispiel ein Tippfehler in der `time.sleep()`-Zeile) die Taste nicht mit `release()` freigegeben wird, sendet das HID-Modul den Tastencode unendlich. In so einem Fall kann man nur noch schnell den Pico abstöpseln.

Entsprechende Funktionen gibt es auch für die Mediensteuerung mit Consumer-Con-

Programmstruktur auf dem Picodrive

Das im Projekt oben verwendete *code.py* wird bei jeder Inbetriebnahme und jedem Neustart des Pico ausgeführt, ideal für Anwendungen die nicht mehr geändert werden. Für Experimente sollten Sie *code.py* nicht direkt ändern. Eine ungewollte, endlose Schleife beim Ausführen des Programms kann es schwierig machen, das Programm nach Einstecken des Pico zu beenden. Auch möchte man vielleicht unterschiedliche Aufgaben und Tests erledigen und nicht jedes Mal dazu *code.py* ändern. Daher ruft man, nachdem man sicher ist, dass der Code funktioniert, aus *code.py* nur das eigentliche Programm auf. Für ein Programm mit dem Namen *anwendung.py* geschieht das mit `import anwendung` (auch ohne .py). In *code.py* kann man aus Sicherheitsgründen auch noch ein paar Sekunden warten, bis es weiter geht, dann hat man im Falle eines Falles noch Zeit, *code.py* zu unterbrechen. Auch wenn wir hier mit Python arbeiten, sind wir doch auf einem Mikroprozessor aktiv, den ein wirres Programm durchaus crashen kann.

Weitere Ideen

Eine Eigenbau-Tastatur kann auch gut eingesetzt werden bei Anwendungen wie öffentlichen Terminals oder Zugangssystemen, bei denen eine komplette Tastatur nicht nötig, erwünscht oder zu empfindlich ist. Dann können spezielle Tasten eingesetzt werden, die wasserfest oder vandalensicher sind. Gehen einem bei einem Raspberry-Pi-Projekt die GPIOs aus, so kann solch eine Pico-Tastatur noch viele weitere Funktionen hinzufügen, auch parallel zu der normalen Tastatur.

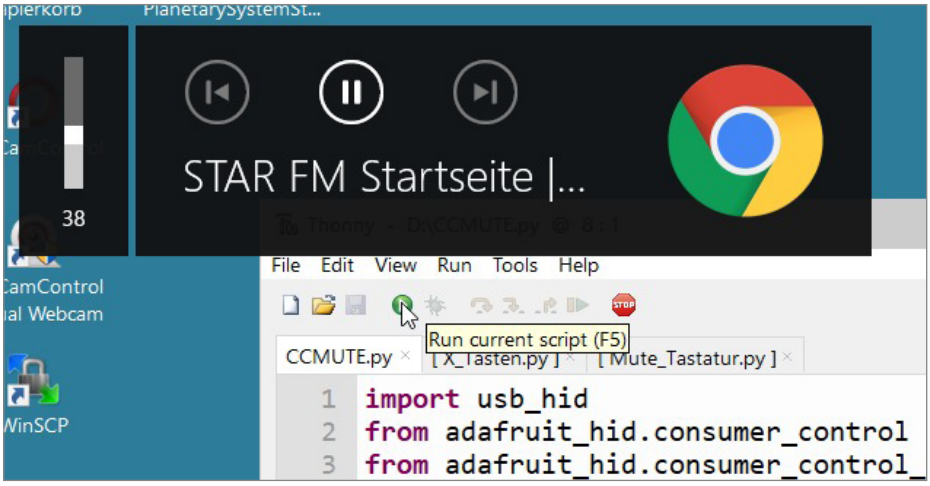
Geht man einen Elektronikkatalog durch und sieht Neigungsschalter, Reedkontakte (durch Magnetfelder ausgelöst), Schlüsselschalter, Näherungsschalter oder Lichtschranken mit Relaisausgang, fallen einem die Ideen geradezu in den Schoß.

Wer mit dem PC Musik macht, wünscht sich vielleicht ein Sustain- bzw. Haltepedal für Software-Synthesizer. Näherungsschalter können kontaktlose Eingaben an einem zu steuernden Gerät ermöglichen. Es sind auch Anwendungen denkbar, um Geräte auf Herz und Nieren zu testen. Entwickeln Sie neuartige, praktische oder behindertengerechte Eingabegeräte, bis hin zu funktionalen Simulator-Cockpits. Seien Sie kreativ!

⓮ CCMUTE.py

```
# Consumer Control Mediensteuerung
import time
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

cc = ConsumerControl(usb_hid.devices)
cc.send(ConsumerControlCode.MUTE)
```



⓯ Der Pico hat gerade meinen Lieblingssender wieder laut geschaltet

Widerstände, Pulldown, Pullup?

Pullup- oder *Pulldown*-Widerstände sorgen an Mikrocontrollern für definierte elektrische Zustände an den Eingängen. Ohne diese Maßnahme könnten selbst kleine elektrische Störungen den Eingang ungewollt schalten. Offene Eingänge zeigen dann oft völlig zufällige Werte an. Man kann als Abhilfe externe Widerstände benutzen oder auch die im Chip vorhandenen internen Widerstände. Bei einigen Mikrocontrollern gibt es intern nur *Pullups*, dort hat man keine Wahl, wenn man keine externen *Pulldowns* verwenden möchte.

Pullup-Widerstände ziehen einen Eingang auf die Arbeitsspannung des Prozessors. Damit ist dieser Eingang dann *HIGH*, was im Code ein `True` (wahr) bei der Abfrage ergibt. Bei gedrücktem Taster wird der Eingang mit *GND* (Masse) verbunden und ist dann *LOW*, also `False` (unwahr) im Code. Das liest sich im Code etwas verwirrend, wird aber häufig so verwendet. Bei einem Platinenlayout kann dies Vorteile haben, da GND praktisch überall auf der Platine vorhanden ist und so das Layout um die Taster herum einfacher bleibt.

Pulldown-Widerstände legen den Eingang auf GND-Pegel. Bei gedrücktem Taster wird der Eingang mit *VCC* (Prozessor-Versorgungsspannung) verbunden und ist dann *HIGH*. Dies liest sich im Code einfacher, `True` bedeutet so „Taster gedrückt". Pulldown-Widerstände wurden traditionell bei der veralteten TTL-Technik verwendet, damit konnte der Stromverbrauch gesenkt werden. Bei modernen Chips ist dies aber nicht mehr relevant.

trol-Befehlen. So können Sie dann die Lautstärke Ihres Computers vom Pico aus steuern. Im Listing ⓱ wird auf diese Weise die Lautstärke eine Sekunde erhöht, dann wird zwei Sekunden gewartet und die Lautstärke wieder gesenkt. So können Sie eine Tastatur mit Tasten für Ihren Mediacenter-PC bauen. Die CC-Codes sind bei Adafruit dokumentiert.

Auch die Maus lässt sich steuern, was durchaus bei Tests auf Hardware, auf denen man keine Anwendungen installieren darf oder kann, verwendbar ist. Bei der Recherche zu diesem Artikel habe ich auch gelernt, dass es sogenannte „Mouse Wiggler" gibt, die auf einem von der Firma überwachten Computer eine Aktivität des Benutzers vorgaukeln. Ich warte noch mit dem Kauf, bis diese Geräte auch sonst meine Arbeit abnehmen können. Ein ganz simples Programm, das die Maus bewegt, sehen Sie in Listing ⓲.

Wird der Taster betätigt, verschiebt sich der Mauscursor etwas nach links oben. Wahrscheinlich kann man mit einem ähnlichen Programm, dass zufallsgesteuert die Maus bewegt, jemanden zum Wahnsinn treiben. Wenn es konstruktiver sein soll, so können Sie hiermit auch einen barrierefreien Mausersatz bauen.

Egal welches Maker-Problem Sie mit einer oder mehreren Zusatztastaturen auf Basis des Raspberry Pico oder auch anderer Mikrocontroller Boards angehen wollen, fangen Sie an zu basteln! Gerne würden wir von Ihren Lösungen und Erkenntnissen hören. Eventuell möchten Sie es auch auf *Make: Projects* veröffentlichen und so der Community bekannt machen. —*caw*

⑯ Keyb_PressRelease.py

```
# Keyboard send() und send()/release()
import time
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode

kbd = Keyboard(usb_hid.devices)

kbd.send(Keycode.A) # send

# press/release
kbd.press(Keycode.A)
time.sleep(2)
kbd.release(Keycode.A)
```

⑰ CC_PressRelease.py

```
# Consumer Control Mediensteuerung
import time
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

cc = ConsumerControl(usb_hid.devices)

# 2 Sekunden Lautstärke erhöhen
cc.press(ConsumerControlCode.VOLUME_INCREMENT)
time.sleep(1)
cc.release()

time.sleep(2)

# 2 Sekunden Lautstärke verringern
cc.press(ConsumerControlCode.VOLUME_DECREMENT)
time.sleep(1)
cc.release()
```

⑱ Maus.py

```
# Mausbewegung bei Tastendruck
import usb_hid
from adafruit_hid.mouse import Mouse

import board
import digitalio
import time

m = Mouse(usb_hid.devices)

button = digitalio.DigitalInOut(board.GP16)
button.switch_to_input(pull=digitalio.Pull.DOWN)

while True:
    if button.value:
# 3. Parameter ist Scrollrad
   m.move(-10, -10, 0)
time.sleep(0.2)
```