Versuchs-Hut

So verdrahten Sie Schalter, Taster und LEDs am Raspi

An den Raspi können Sie Schalter und LEDs direkt anschließen – toll zum Experimentieren. Wir zeigen an einer aufsteckbaren Lochrasterplatine, was Elektronik-Neulinge beim Verdrahten beachten sollten.

Von Pina Merkert

Auf dem Raspberry Pi ragen hinter der Netzwerkbuchse 40 Drahtstifte aus der Platine, auf die man Erweiterungsplatinen wie einen Hut aufstecken oder Jumperkabel direkt anstecken kann. 27 dieser Pins sind sogenannte General-Purpose-Input/Output-Pins (GPIO), also Ein- und Ausgabe-Pins ohne von vornherein festgelegte Aufgabe. Jeden davon kann man als Ausgang konfigurieren und mit eigenen Programmen eine Spannung von 3,3 Volt anlegen oder mit Masse verbinden (GND-Pegel). Man kann sie aber auch jeweils als Eingang benutzen, sodass das eigene Programm erkennt, ob sie mit der 3,3-Volt-Spannungsversorgung oder mit Masse verbunden sind.

Eingänge nicht in die Luft hängen!

Eingänge sollten in einer Schaltung entweder mit Masse (GND-Pin) oder mit 3,3 Volt verbunden sein. Keinesfalls dürfen Sie sie direkt mit den 5-Volt-Pins verbinden, Sie würden sonst den Raspi irreparabel beschädigen. Hängen die Eingänge ohne elektrische Verbindung in der Luft, liest der Raspi zufällig mal einen hohen oder niedrigen Pegel ein, je nachdem, ob sich auf dem Drahtstückchen ein paar Elektronen gesammelt haben oder nicht. Einen so undefinierten Pegel sollte man vermeiden.

Schließt man beispielsweise einen Taster an, stellt dieser im gedrückten Zustand eine Verbindung zu 3,3 Volt her. Ist er allerdings nicht gedrückt, würde eine frei hängende Verbindung entstehen. Deswegen sollte man vom Pin zur Masse eine Verbindung einbauen, über die Ladungen abfließen können. Wäre diese Verbindung aber nur ein Draht, entstünde beim Drücken des Schalters ein Kurzschluss. Man verwendet deswegen sogenannte „Pull-down-Widerstände“, die den Strom begrenzen, wenn der Schalter geschlossen ist. 10 kΩ sind dafür ein gebräuchlicher Wert. Über einen so großen Widerstand können nur ganz kleine Ströme fließen, die geringe Restladung beim Öffnen des Schalters fließt aber verlässlich ab.

Prellen

Beim Schließen des mechanischen Kontakts in einem Schalter kommt es innerhalb einer kurzen Phase von weniger als einer Zehntelsekunde zu einer noch nicht vollständig geschlossenen Verbindung. Meist schwingen federnde Bleche und stellen mehrfach kurz eine Verbindung her, die sie gleich darauf wieder verlieren. Elektroniker bezeichnen dieses Hin und Her als „Prellen“.

Da der Raspi per Interrupt auf steigende und fallende Flanken an einem GPIO reagieren kann, registriert er das Prellen als mehrfaches Schalten. Gegen diese Störung kann man sowohl hardwaremäßig als auch softwareseitig vorgehen.

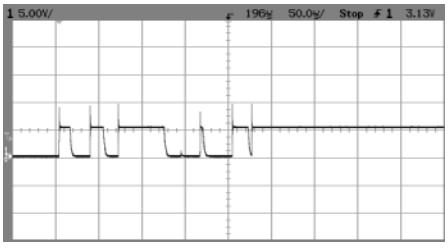
Zum Entprellen per Hardware lötet man beispielsweise einen Kondensator mit 0,1 µF zwischen den Pin und GND. Der muss sich beim Einschalten erst mal aufladen, weshalb er das Potenzial beim Einschalten kurz niederhält. Beim Ausschalten entlädt er sich und hält das Potenzial für einen Moment noch hoch. Eine größere Kapazität entprellt noch sicherer, aber bei sehr großen Kondensatoren reagiert der Schalter mit einer merklichen Verzögerung. Passen Sie den Wert also ruhig an Ihren Schalter und Ihre Bedürfnisse an. Je nach Schalter verhindert das nicht jeden ungewollten Interrupt, reduziert das Hin und Her aber auf wenige Ausnahmen.

Denen kann man auch in der Software begegnen: Beispielsweise kann das Programm einen GPIO einfach nur in Zehntelsekundenabständen abfragen. Oder man fügt ein `time.sleep(0.1)` hinter der Reaktion auf ein Schaltereignis ein, um erst nach einer Zehntelsekunde wieder auf einen weiteren Interrupt zu reagieren. Im Code unter ct.de/y1rk haben wir ein Beispiel für ein Software-Entprellen eingefügt.

Bauteile wie LEDs haben einen recht kleinen Widerstand, weshalb auch bei 3,3 Volt vergleichsweise große Ströme fließen können. Im schlimmsten Fall geht eine LED kaputt, wenn sie direkt am GPIO hängt. Es empfiehlt sich daher, die Spannung an der LED zu reduzieren, was laut Ohm’schem Gesetz auch den Strom begrenzt. Dafür fügt man einen Vorwiderstand in den Stromkreis ein. Den perfekten Wert für den Vorwiderstand einer LED können Sie mit den Werten aus dem Datenblatt der LED berechnen. Falls Ihnen das zu mühevoll ist: 80 Ω ist ein passender Wert für alle üblichen LEDs. Finden sich keine 80 Ω in der Bastelkiste, tun es auch 100 Ω.

Lochrasterplatine

Damit die Bastelei nicht zum Kabelverhau wird, lohnen sich schon ab wenigen Bauteilen Lochrasterplatinen. Sie halten die Bauteile an Ort und Stelle und sorgen für Übersicht. Um möglichst billig wegzukommen, reicht es, von einer größeren



Das Speicheroszilloskop zeigt, wie oft der Pegel eines prellenden Schalters wechselt. Das Hin und Her dauert bei diesem Schalter etwa 250 µs.

Platine für weniger als einen Euro ein Raspi-großes Stück zum Experimentieren abzusägen.

Wer richtig viel Platz braucht, kann auch ein altes IDE-Flachkabel als Verbindung zu einer großen Platine nutzen. Die 40-poligen Stecker passen genau auf die Pinleiste vom Raspi. Die Litzen jüngerer UDMA-Kabel führen zwischen allen Signalkabeln allerdings eine zusätzliche Masseverbindung, sodass die Litze zu fein wird, um sie mit heimischen Mitteln zu verarbeiten. Aus einem alten Kabel kann man dafür aber mindestens drei Stecker ernten.

Wer sich mehr Bequemlichkeit erkaufen will, erwirbt eine an den Raspi angepasste Platine für 5 bis 10 Euro. Diverse Hersteller bieten leicht unterschiedliche Modelle unter Bezeichnungen wie „Raspberry Pi Prototyping HAT“ an. Sie verbinden die GPIO-Pins zu Löchern, die nicht direkt nebeneinander liegen, sodass man Bauteile dazwischen löten kann. Außerdem führen sie GND und 3,3 Volt als Streifen über die Platine, was kürzere Wege beim Verkabeln erlaubt.

Ansteuerung mit Python

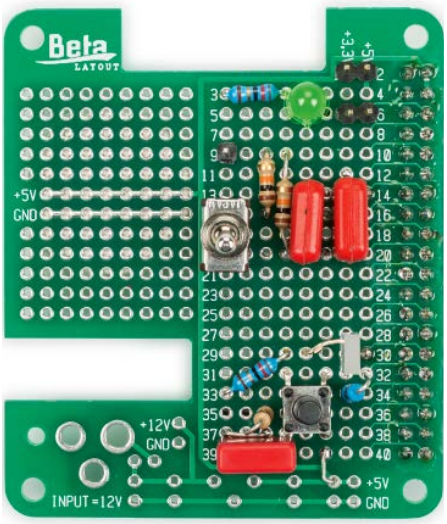
Der Bequemlichkeit zuliebe nutzen die folgenden Beispiele für die Ansteuerung Python. Dort ist die Schnittstelle zu den Pins die `GPIO`-Bibliothek. Ihr teilt man zunächst mit, welches Nummerierungsschema man verwendet (`GPIO.BCM` sind die Pin-Nummern des SoC) und welche Pins als Ein- oder Ausgänge dienen:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(2, GPIO.OUT,
           initial=GPIO.LOW)
GPIO.setup(22, GPIO.IN)
```

Eine Endlosschleife sorgt danach beispielsweise für eine blinkende LED:

```
import time
while True:
  GPIO.output(2, GPIO.HIGH)
  time.sleep(0.5)
  GPIO.output(2, GPIO.LOW)
  time.sleep(0.5)
```

LEDs einfach nur blinken zu lassen, sieht bei weitem nicht so schön aus wie ein weicher Wechsel der Helligkeit. Das geht erstaunlich leicht mit Pulsweitenmodulation (PWM). Der Raspi beherrscht PWM in Hardware an den Pins 12, 13, 18 und 19. An allen anderen Pins emuliert er den Effekt durch schnelles Umschalten softwareseitig. Das Beispiel moduliert die Pulsweiten mit 50 Hz:



Unser Raspi-GPIO-Testaufbau entstand auf einem Werbegeschenk eines Platinenfertigers, das dieser auf der Maker Faire in Hannover verteilte. Vergleichbare Platinen verkaufen Pimoroni und andere für circa 5 Euro.

```
GPIO.setup(12, GPIO.OUT)
led = GPIO.PWM(12, 50)  # 50Hz
while True:
  for dc in range(0, 101, 5):
    led.ChangeDutyCycle(dc)
    time.sleep(0.1)
  for dc in range(100, -1, -5):
    led.ChangeDutyCycle(dc)
    time.sleep(0.1)
```

Glättet man ein PWM-Signal mit einem Kondensator, lassen sich damit feingliedrig Spannungen zwischen 0 und 3,3 Volt einstellen. Die optimale Kapazität hängt von der Last ab, da große Ströme große Kondensatoren trotzdem schnell entladen. Generell sorgt ein größerer Kondensator für eine glattere Spannung. Falls Sie das geglättete Signal modulieren wollen, sollten Sie den Kondensator allerdings nicht zu groß wählen, weil er dann auch die Modulation glättet. Probieren Sie für Ihr Projekt einfach verschiedene Kapazitäten aus.

Leseoptionen

Auslesen von GPIOs geht mit `GPIO.input(pinNo)`. Falls der Raspi aber noch mit anderen Berechnungen beschäftigt ist, bekommt er mit dieser Funktion nicht sofort mit, wenn sich das Potenzial am Pin ändert. Abhilfe schafft die Funktion `GPIO.wait_for_edge(pinNo, GPIO.RISING)`, die blockiert, bis der Raspi einen Interrupt wegen einer steigenden Flanke verarbeitet. Da das Programm dabei blockiert ist, empfiehlt es sich, eine solche Abfrage in einen eigenen Thread auszulagern (das `time.sleep(0.1)` dient dem Entprellen):

```
from threading import Thread
def ck_btn_forever():
  while True:
    GPIO.wait_for_edge(26,GPIO.RISING)
    print("Button pressed")
    time.sleep(0.1)
Thread(target=ck_btn_forever).start()
```

Stattdessen kann man die Überwachung der Bibliothek überlassen und bei einem Ereignis ein Callable ausführen, im hier folgenden Code-Ausschnitt eine `lambda`-Funktion:

```
GPIO.add_event_detect(
  22, GPIO.RISING,
  callback=lambda _: print(
           "<- Switch to the left."))
GPIO.add_event_detect(
  10, GPIO.RISING,
  callback=lambda _: print(
           "-> Switch to the right."))
```

Serielle Schnittstellen

Über die PIN-Leiste des Raspi stellt der Minirechner auch zwei SPI-Schnittstellen, einen I2C-Bus und eine serielle Verbindung bereit. Komplexere Sensoren mit eigener Intelligenz bindet man über diese Schnittstellen an. An einem Bus können dabei meist Dutzende Sensoren hängen. Über I2C unterstützt der Raspi auch Chips, die noch mehr GPIO-Pins bereitstellen [1].

All diese Erweiterungen belegen nur einen Teil der GPIOs, sodass bei jedem Projekt noch Pins frei bleiben, um Modi bequem per Kippschalter umzuschalten oder mit einer Status-LED anzuzeigen, was der Raspi gerade macht. *(pmk@ct.de)* c't

Literatur

[1] Pina Merkert, Mehr I/O zum Basteln, Sehr viele GPIO-Pins an Arduino, Raspi und ESP, c't 15/2020, S. 146

Beispielcode: ***ct.de/y1rk***