Raspberry Pi

Teil 1: Programmierung der GPIO-Pin-Leiste

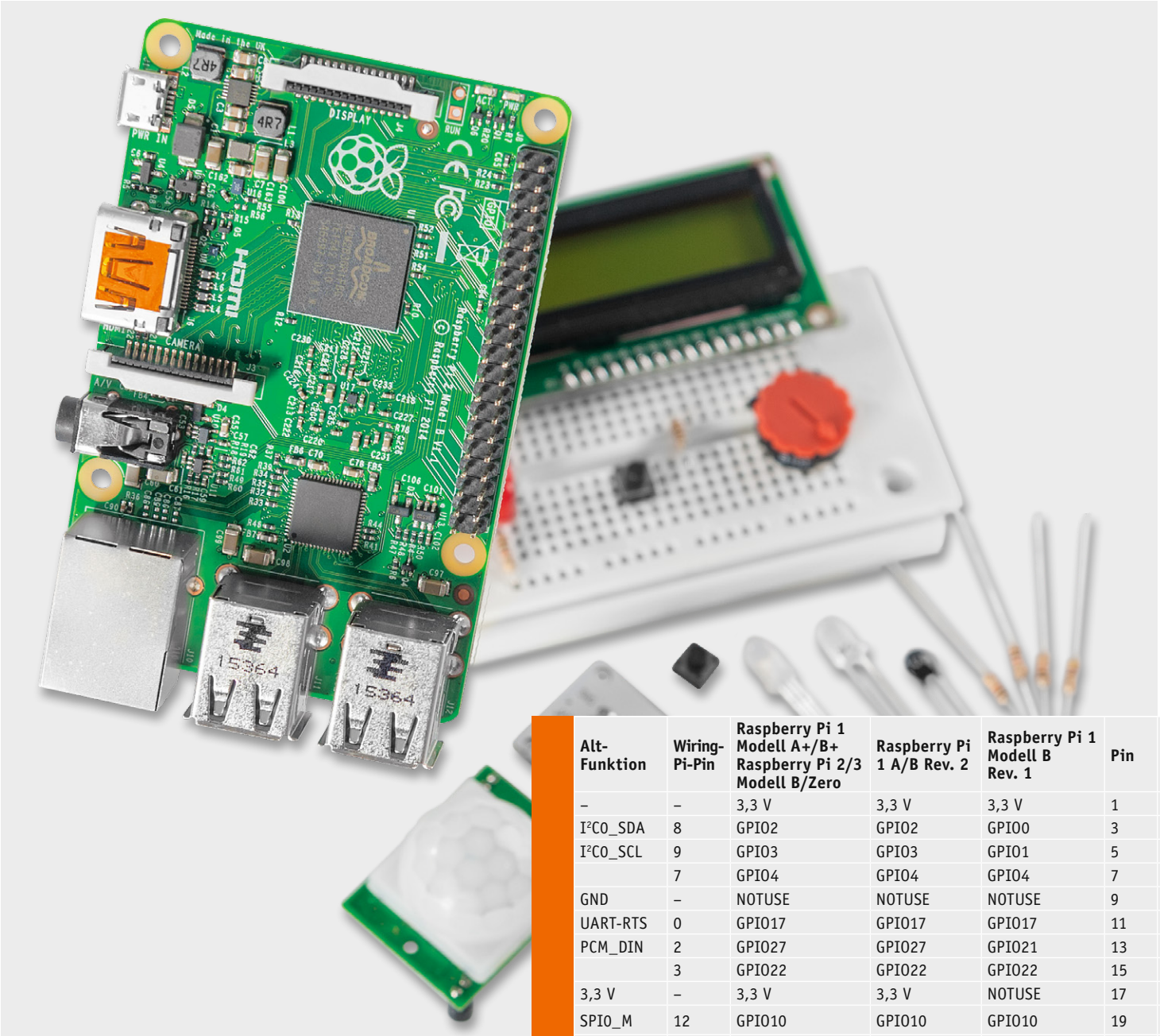


Tabelle 1

Alt-Funktion	Wiring-Pi-Pin	Raspberry Pi 1 Modell A+/B+ Raspberry Pi 2/3 Modell B/Zero	Raspberry Pi 1 A/B Rev. 2	Raspberry Pi 1 Modell B Rev. 1	Pin
–	–	3,3 V	3,3 V	3,3 V	1
I²C0_SDA	8	GPIO2	GPIO2	GPIO0	3
I²C0_SCL	9	GPIO3	GPIO3	GPIO1	5
	7	GPIO4	GPIO4	GPIO4	7
GND	–	NOTUSE	NOTUSE	NOTUSE	9
UART-RTS	0	GPIO17	GPIO17	GPIO17	11
PCM_DIN	2	GPIO27	GPIO27	GPIO21	13
	3	GPIO22	GPIO22	GPIO22	15
3,3 V	–	3,3 V	3,3 V	NOTUSE	17
SPI0_M	12	GPIO10	GPIO10	GPIO10	19
SPI0_MISO	13	GPIO9	GPIO9	GPIO9	21
SPI0_SCLK	14	GPIO11	GPIO11	GPIO11	23
GND		NOTUSE	NOTUSE	NOTUSE	25
SDA_0	30	ID_SD	–	–	27
	21	GPIO5	–	–	29
	22	GPIO6	–	–	31
	23	GPIO13	–	–	33
	24	GPIO19	–	–	35
	25	GPIO26	–	–	37
GND		GND	–	–	39

Egal, ob der kleine Raspberry Pi Zero, der große Raspberry Pi 2/3 oder die betagten Vorgänger: Gerade beim Erstellen von Shell-Skripten in C oder Python ist der Umgang mit den GPIO-Anschlüssen zwar einfach, aber auch relativ umständlich gelöst. Für mehr Möglichkeiten beim Programmieren und vor allem mehr Übersicht sorgt die Auslagerung von Funktionen in eine API-Schnittstelle (Advanced Programming Interface) die für mehrere Programmiersprachen zur Verfügung steht (Bild 1).

Klassisch verwendet man auf dem Raspberry Pi das Terminal und steuert über die Kommandozeile die einzelnen GPIO-Pins in einem Elektronikprojekt. Oder Sie verwenden eine Programmiersprache wie C oder Python und bauen eine Logik in Form eines Programms dazu, die beispielsweise den einen oder anderen GPIO-Pin nur dann ansteuert oder ausliest, wenn Sie das wollen.

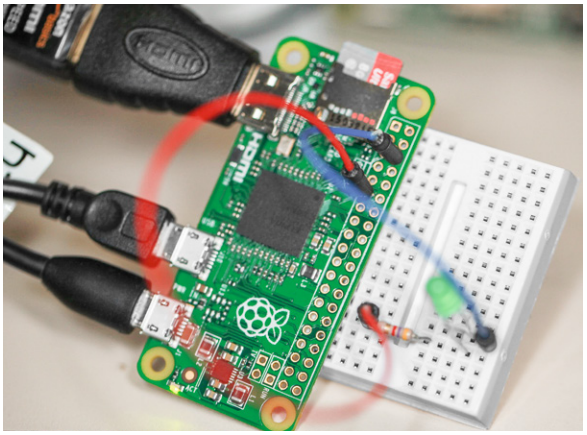
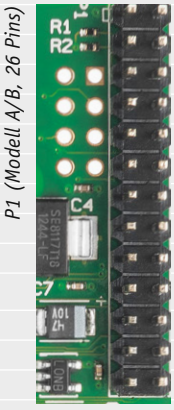
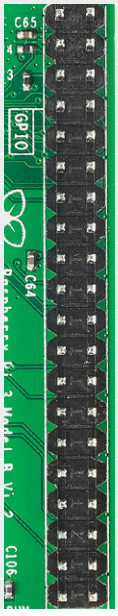


Bild 1: Für kleine Schaltungsexperimente kann der Raspberry Pi Zero auch mal direkt an das Breadboard gesteckt werden.

GPIO-Leiste – Unterschiede zwischen BCM-, WiringPi und Pin-Zählung

Wie auch immer, bei der Anwendung der neuen gpiozero-Bibliothek müssen Sie darauf achten, dass hier die Broadcom-/BCM-Zählung bei der Initialisierung der Pins gilt. Das heißt konkret, dass Sie beispielsweise statt Pin 11 hier den Wert 17 verwenden, falls Sie den GPIO17-Pin in Ihrer Schaltung und dem Python-Code verwenden wollen.

In den jeweils drei Spalten neben der Pin-Nummer in Tabelle 1 finden Sie für jedes Raspberry-Pi-Modell die entsprechende Beschreibung für den vorgesehenen Zweck. Während die 5-V- und 3,3-V-Spannungspins sowie die GND-Masse-Anschlüsse ausschließlich für elektronische Eigenbau-Schaltungen, Sensoren und dergleichen zur Verfügung stehen,

GPIO-Sockel	Pin	Raspberry Pi 1 Modell B Rev 1	Raspberry Pi 1 Modell A/B Rev. 2	Raspberry Pi 1 Modell A+/B+/ Raspberry Pi 2/3 Modell B/Zero	WiringPi-Pin	Alt-Funktion
 *P1 (Modell A/B, 26 Pins)*  *J8 (A+/B+/B2/Zero, 40 Pins)*	2	5 V	5 V	5 V	–	5 V
	4	NOTUSE	5 V	5 V	–	5 V
	6	GND	GND	GND	–	–
	8	GPIO14	GPIO14	GPIO14	15	UART0_TXD
	10	GPIO15	GPIO15	GPIO15	16	UART0_RXD
	12	GPIO18	GPIO18	GPIO18	1	PWM
	14	NOTUSE	NOTUSE	NOTUSE	–	GND
	16	GPIO23	GPIO23	GPIO23	4	
	18	GPIO24	GPIO24	GPIO24	5	
	20	NOTUSE	NOTUSE	NOTUSE	–	GND
	22	GPIO25	GPIO25	GPIO25	6	
	24	GPIO8	GPIO8	GPIO8	10	SPI0_CE0_N
	26	GPIO7	GPIO7	GPIO7	11	SPI0_CE1_N
	28	–	–	ID_SC	31	SCL0
	30	–	–	GND		GND
	32	–	–	GPIO12	26	
	34	–	–	GND	–	GND
	36	–	–	GPIO16	27	
	38	–	–	GPIO20	28	
	40	–	–	GPIO21	29	

sind jene mit einer führenden GPIO-Bezeichnung im Namen für den Programmiereinsatz wichtig. Jeder einzelne GPIO-Pin kann mit jeder auf dem Raspberry Pi verfügbaren Programmier- oder Skriptsprache direkt angesprochen werden.

raspi-gpio auf der Shell

Ist der Befehl *raspi-gpio* auf dem Raspberry Pi noch nicht verfügbar, kann dieser mit dem Kommando:

```
sudo apt-get install raspi-gpio
```

nachträglich installiert werden. Sämtliche GPIO-Anschlüsse mitsamt der jeweiligen Funktionen geben in einem Terminalfenster das Kommando

```
sudo raspi-gpio funcs
```

aus. Damit wird dargestellt, welche alternative Funktionen alle GPIOs liefern. Die Ausgabe lässt sich auch auf einen GPIO einschränken.

```
sudo raspi-gpio funcs <GPIO-NUMMER>
```

Mit dem Befehl *raspi-gpio* stehen nun weitere Möglichkeiten auf der Kommandozeile zur Verfügung, um mit administrativen Root-Rechten die GPIO-Pins zu hacken und zu überwachen. Mit dem Befehl *raspi-gpio help* erhalten Sie auf der Shell die Dokumentation. Das *raspi-gpio*-Kommando lässt sich auch mit dem beliebten *grep*-Befehl kombinieren:

```
sudo raspi-gpio funcs | grep UP
```

Er liefert die GPIO-Anschlüsse, die aktuell mit dem Status UP im System hängen. Übersichtlicher ist die Ausgabe von raspi-gpio mit der *get*-Option:

```
sudo raspi-gpio get
```

Diese stellt sämtliche GPIOs „Bank"-weise sortiert und die Pins gemäß ihrer Funktion wie Input, TX/RX, I²C und SPI übersichtlicher dar (Bild 2).

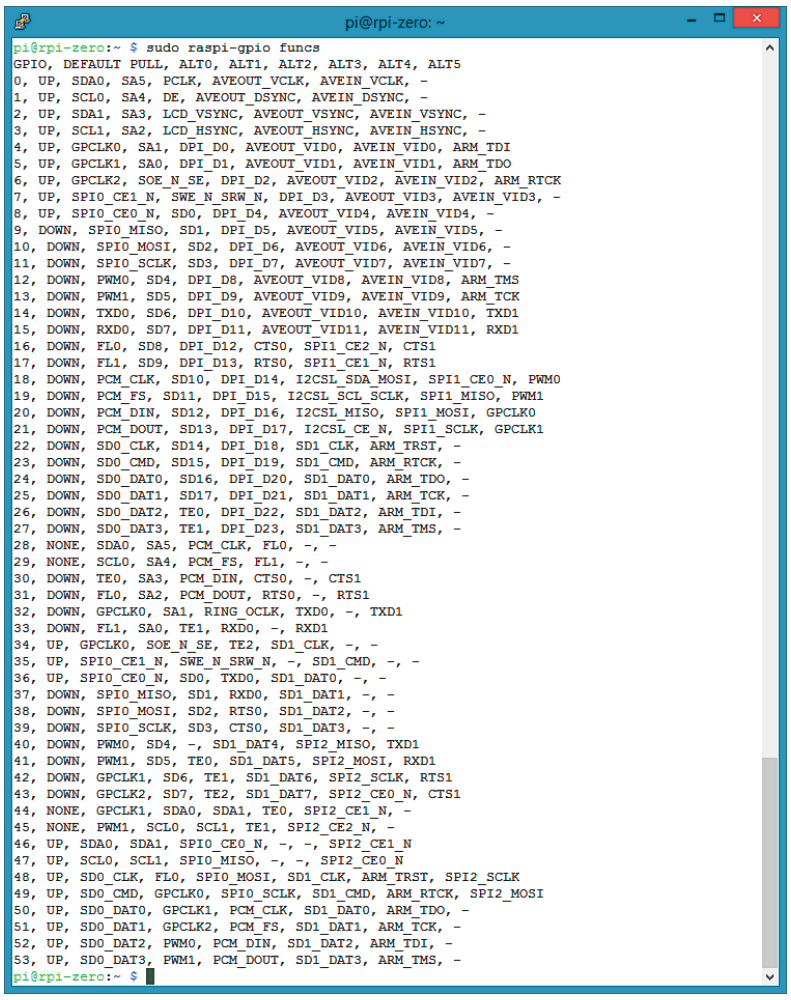


Bild 2: Die gelisteten GPIO-Anschlüsse sind über das gesamte Board verteilt untergebracht – auch deshalb geht die Anzahl über die 40-Pin-Benutzer-GPIO-Pin-Leiste deutlich hinaus. Die aktiven GPIO-Pins für den Raspberry Pi Zero sind hier jene, die mit dem Status UP in dieser Übersicht ausgegeben werden.

Lassen Sie sich hier nicht von der schieren Anzahl der GPIO-Pin-Bezeichnungen erschlagen, denn viele sind für interne Zwecke bereits in Verwendung und stehen dem Benutzer gewöhnlich nicht für Experimente zur Verfügung. Beispielsweise sind GPIO40 und GPIO45 für analoges Audio zuständig und haben eine „echte" PWM-Funktion, GPIO46 dient als Hot-Plug-Pin für ein gestecktes HDMI-Kabel, GPIO47 bis GPIO53 werden für den (micro)SD-Kartenanschluss für die Erkennung und Datenverarbeitung benötigt. Egal, welcher Raspberry Pi im Endeffekt zum Einsatz kommt, beschränken Sie sich auf die GPIO-Pins, die von der 40-poligen GPIO-Leiste (bei dem Zero ist diese mit J8 beschriftet) zur Verfügung gestellt werden (Bild 3).

Neben dem Auslesen der aktuellen Zustände der jeweiligen GPIO-Pins können mit dem *raspi-gpio*-Werkzeug auch auf der Kommandozeile einzelne oder mehrere Pins gesetzt werden. Dafür nutzen Sie raspi-gpio mit der *set*-Option.

```
sudo raspi-gpio set <GPIO-NUMMER>
[GPIO-Option]
```

Somit ist es einfach möglich, beispielsweise einen GPIO-Pin als Ein- oder als Ausgang zu definieren und diesem anschließend einen Wert zuzuweisen.

GPIO-Option	Bemerkung – GPIO
ip	Eingang (Input)
op	Ausgang (Output)
a0-a5	alternativer Betrieb (Alt Func) alt0 bis alt5
pu	interner Pull-up-Widerstand aktiv
pd	interner Pull-up-Widerstand ausgeschaltet
pn	keine Pull-up/down-Funktion (No Pull)
dh	Digital High, High Level (1)
dl	Digital Low, Low Level (0)

Haben Sie beispielsweise an den Anschluss GPIO17 eine LED angeschlossen, definieren Sie diesen zunächst als Ausgang:

```
sudo raspi-gpio set 17 op
```

Im nächsten Schritt schalten Sie die LED ein (LOW = 0, HIGH = 1):

```
sudo raspi-gpio set 17 dh
```

und wieder aus:

```
sudo raspi-gpio set 17 dl
```

Beide Kommandos funktionieren jedoch nur, falls der verwendete GPIO-Pin auch zuvor entsprechend als Ausgang definiert wurde. Hier ist es auch möglich, mehrere Optionen gemeinsam in einer Befehlszeile zu verwenden, sofern dies sinnvoll ist. Somit lassen sich beide Kommandos aus dem obigen Beispiel mit dem Kommando

```
sudo raspi-gpio set 17 op dh
```

analog abbilden. Haben Sie den Zustand verändert, prüfen Sie mit der bekannten *get*-Option den Zustand:

```
sudo raspi-gpio get 17
```

```
pi@rpi-zero: ~
pi@rpi-zero:~ $ sudo raspi-gpio get
BANK0 (GPIO 0 to 27):
  GPIO 00: level=1 fsel=0 alt=  func=INPUT
  GPIO 01: level=1 fsel=0 alt=  func=INPUT
  GPIO 02: level=1 fsel=0 alt=  func=INPUT
  GPIO 03: level=1 fsel=0 alt=  func=INPUT
  GPIO 04: level=1 fsel=0 alt=  func=INPUT
  GPIO 05: level=1 fsel=0 alt=  func=INPUT
  GPIO 06: level=1 fsel=0 alt=  func=INPUT
  GPIO 07: level=1 fsel=0 alt=  func=INPUT
  GPIO 08: level=1 fsel=0 alt=  func=INPUT
  GPIO 09: level=0 fsel=0 alt=  func=INPUT
  GPIO 10: level=0 fsel=0 alt=  func=INPUT
  GPIO 11: level=0 fsel=0 alt=  func=INPUT
  GPIO 12: level=0 fsel=0 alt=  func=INPUT
  GPIO 13: level=0 fsel=0 alt=  func=INPUT
  GPIO 14: level=1 fsel=4 alt=0 func=TXD0
  GPIO 15: level=1 fsel=4 alt=0 func=RXD0
  GPIO 16: level=0 fsel=0 alt=  func=INPUT
  GPIO 17: level=0 fsel=0 alt=  func=INPUT
  GPIO 18: level=0 fsel=0 alt=  func=INPUT
  GPIO 19: level=0 fsel=0 alt=  func=INPUT
  GPIO 20: level=0 fsel=0 alt=  func=INPUT
  GPIO 21: level=0 fsel=0 alt=  func=INPUT
  GPIO 22: level=0 fsel=0 alt=  func=INPUT
  GPIO 23: level=0 fsel=0 alt=  func=INPUT
  GPIO 24: level=0 fsel=0 alt=  func=INPUT
  GPIO 25: level=0 fsel=0 alt=  func=INPUT
  GPIO 26: level=0 fsel=0 alt=  func=INPUT
  GPIO 27: level=0 fsel=0 alt=  func=INPUT
BANK1 (GPIO 28 to 45):
  GPIO 28: level=1 fsel=4 alt=0 func=SDA0
  GPIO 29: level=1 fsel=4 alt=0 func=SCL0
  GPIO 30: level=0 fsel=0 alt=  func=INPUT
  GPIO 31: level=0 fsel=0 alt=  func=INPUT
  GPIO 32: level=0 fsel=0 alt=  func=INPUT
  GPIO 33: level=0 fsel=0 alt=  func=INPUT
  GPIO 34: level=1 fsel=0 alt=  func=INPUT
  GPIO 35: level=1 fsel=0 alt=  func=INPUT
  GPIO 36: level=1 fsel=0 alt=  func=INPUT
  GPIO 37: level=0 fsel=0 alt=  func=INPUT
  GPIO 38: level=0 fsel=0 alt=  func=INPUT
  GPIO 39: level=0 fsel=0 alt=  func=INPUT
  GPIO 40: level=0 fsel=0 alt=  func=INPUT
  GPIO 41: level=0 fsel=0 alt=  func=INPUT
  GPIO 42: level=0 fsel=0 alt=  func=INPUT
  GPIO 43: level=0 fsel=0 alt=  func=INPUT
  GPIO 44: level=0 fsel=0 alt=  func=INPUT
  GPIO 45: level=0 fsel=0 alt=  func=INPUT
BANK2 (GPIO 46 to 53):
  GPIO 46: level=0 fsel=0 alt=  func=INPUT
  GPIO 47: level=0 fsel=1 alt=  func=OUTPUT
  GPIO 48: level=1 fsel=7 alt=3 func=SD1_CLK
  GPIO 49: level=1 fsel=7 alt=3 func=SD1_CMD
  GPIO 50: level=1 fsel=7 alt=3 func=SD1_DAT0
  GPIO 51: level=1 fsel=7 alt=3 func=SD1_DAT1
  GPIO 52: level=1 fsel=7 alt=3 func=SD1_DAT2
  GPIO 53: level=1 fsel=7 alt=3 func=SD1_DAT3
pi@rpi-zero:~ $
```

Bild 3: Bei dieser übersichtlicheren Bildschirmausgabe werden auch die systeminternen GPIO-Pins ausgegeben – BANK2 beispielsweise ist für die microSD-Karte zuständig.

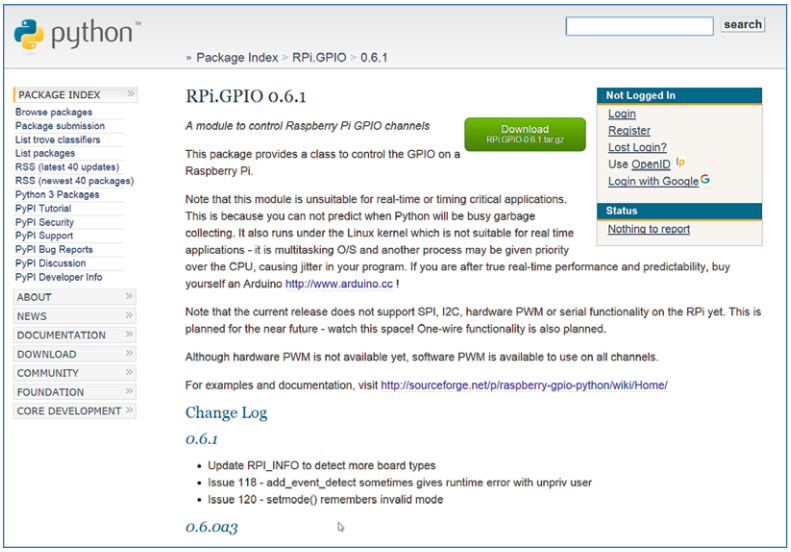


Bild 4: Download der Raspberry-Pi-RPi.GPIO-Bibliothek unter: http://pypi.python.org/pypi/RPi.GPIO

Bild 5: Installation erfolgreich: Das Nachziehen des Python-Grundpakets python-dev brachte auf dem Raspberry Pi letztendlich die Installation der RPi.GPIO-Bibliothek zustande.

```
pi@rpi-zero: ~/RPi.GPIO-0.6.1
pi@rpi-zero:~ $ cd ~
pi@rpi-zero:~ $ mkdir RPi.GPIO
pi@rpi-zero:~ $ wget --no-check-certificate https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.6.1.tar.gz
converted 'https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.6.1.tar.gz' (ISO-8859-15) -> 'https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.6.1.tar.gz' (UTF-8)
--2016-01-05 23:57:41--  https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.6.1.tar.gz
Auflösen des Hostnamen pypi.python.org (pypi.python.org)... 23.235.43.223
Verbindungsaufbau zu pypi.python.org (pypi.python.org)|23.235.43.223|:443... verbunden.
HTTP-Anforderung gesendet, warte auf Antwort... 200 OK
Länge: 27488 (27K) [application/octet-stream]
In RPi.GPIO-0.6.1.tar.gz speichern.

RPi.GPIO-0.6.1.tar.gz                 100%[=========================================================================>]  26,84K  --.-KB/s   in 0,005s

2016-01-05 23:57:42 (4,89 MB/s) - RPi.GPIO-0.6.1.tar.gz gespeichert [27488/27488]

pi@rpi-zero:~ $ tar xfz RPi.GPIO-0.6.1.tar.gz
pi@rpi-zero:~ $ cd RPi.GPIO-0.6.1
pi@rpi-zero:~/RPi.GPIO-0.6.1 $ sudo apt-get install python-dev
Paketlisten werden gelesen... Fertig
Abhängigkeitsbaum wird aufgebaut.
Statusinformationen werden eingelesen.... Fertig
Die folgenden zusätzlichen Pakete werden installiert:
  libpython-dev libpython2.7-dev python2.7-dev
Die folgenden NEUEN Pakete werden installiert:
  libpython-dev libpython2.7-dev python-dev python2.7-dev
0 aktualisiert, 4 neu installiert, 0 zu entfernen und 0 nicht aktualisiert.
Es müssen 18,2 MB an Archiven heruntergeladen werden.
Nach dieser Operation werden 25,7 MB Plattenplatz zusätzlich benutzt.
Möchten Sie fortfahren? [J/n]
Holen: 1 http://mirrordirector.raspbian.org/raspbian/ jessie/main python-dev armhf 2.7.9-1 [1.188 B]
Holen: 2 http://mirrordirector.raspbian.org/raspbian/ jessie/main libpython2.7-dev armhf 2.7.9-2 [17,9 MB]
Holen: 3 http://mirrordirector.raspbian.org/raspbian/ jessie/main libpython-dev armhf 2.7.9-1 [19,6 kB]
Holen: 4 http://mirrordirector.raspbian.org/raspbian/ jessie/main python2.7-dev armhf 2.7.9-2 [281 kB]
Es wurden 18,2 MB in 10 s geholt (1.727 kB/s).
Vormals nicht ausgewähltes Paket libpython2.7-dev:armhf wird gewählt.
(Lese Datenbank ... 134681 Dateien und Verzeichnisse sind derzeit installiert.)
Vorbereitung zum Entpacken von .../libpython2.7-dev_2.7.9-2_armhf.deb ...
Entpacken von libpython2.7-dev:armhf (2.7.9-2) ...
Vormals nicht ausgewähltes Paket libpython-dev:armhf wird gewählt.
Vorbereitung zum Entpacken von .../libpython-dev_2.7.9-1_armhf.deb ...
Entpacken von libpython-dev:armhf (2.7.9-1) ...
Vormals nicht ausgewähltes Paket python2.7-dev wird gewählt.
Vorbereitung zum Entpacken von .../python2.7-dev_2.7.9-2_armhf.deb ...
Entpacken von python2.7-dev (2.7.9-2) ...
Vormals nicht ausgewähltes Paket python-dev wird gewählt.
Vorbereitung zum Entpacken von .../python-dev_2.7.9-1_armhf.deb ...
```

Wie bei Linux gewohnt, haben Sie die Möglichkeit, auf der Shell die gewohnten Linux-Basis-Werkzeuge wie *grep, cut* und Konsorten auf die Bildschirmausgabe anzuwenden und die Rückgabe weiter auszuwerten.

Python-Zugriff mit der RPi.GPIO-API

Python ist standardmäßig bei jedem Raspberry-Pi-Image mit an Bord. Für den einfachen Zugriff auf die GPIO-Pin-Reihe sowie die Steuerung und Überwachung der einzelnen Pins erleichtert eine passende Bibliothek vieles, da diese viele grundsätzliche Vorarbeiten abnimmt. Seit jeher kam dafür im Python-Umfeld die Raspberry-Pi-RPi.GPIO-Bibliothek zum Einsatz, die in der aktuellsten Version kostenlos auf http://pypi.python.org/pypi/RPi.GPIO zum Download bereitsteht (Bild 4), sowie ggf. python-dev, welches grundlegende Pakete für die Python-Programmierung mitbringt.

In diesem Beispiel wird exemplarisch Version 0.6.1 installiert. Möglicherweise steht aktuell im Internet bereits eine neuere Version zur Verfügung. In dem Fall passen Sie die nachfolgenden Kommandos einfach entsprechend der Versionsnummer an. Erscheint eine Zertifikatsfehlermeldung, nutzen Sie zusätzlich die Option *--no-check-certificate,* um die tar.gz dennoch auf den Raspberry Pi zu laden.

```
cd ~
mkdir RPi.GPIO
cd RPi.GPIO
wget --no-check-certificate https://pypi.python.org/packages/
source/R/RPi.GPIO/RPi.GPIO-0.6.1.tar.gz
tar xfz RPi.GPIO-0.6.1.tar.gz
cd RPi.GPIO-0.6.1
```

Nach dem Download entpacken Sie die *gz-tarball*-Datei ins /home/pi-Verzeichnis und navigieren per *cd*-Kommando in das Verzeichnis mit dem Inhalt. Anschließend starten Sie mit dem Kommando

```
sudo python setup.py install
```

die Installation von RPi.GPIO. Das Installationsskript setup.py sorgt für sämtliche Installationsarbeiten. Hier ist allerdings ein installiertes python-dev-Kernpaket Voraussetzung. Erscheint also beispielsweise eine Fehlermeldung mit dem Text „fatal error: Python.h: No such file or directory", sorgt das nächste Kommando für die vorherige Grundinstallation von Python auf dem Raspberry Pi:

```
sudo apt-get install python-dev
```

(Bild 5)

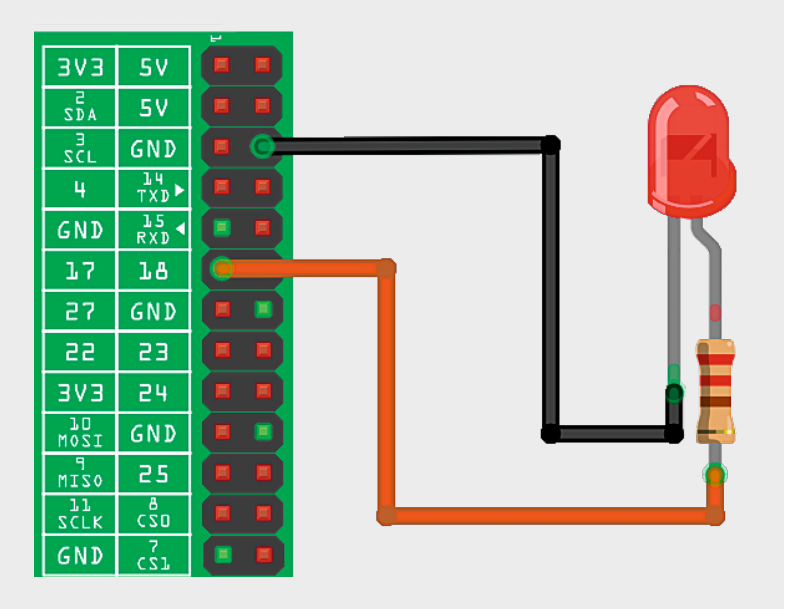


Bild 6: Die Schaltung für die LED ist auf dem Steckboard schnell umgesetzt.

Nun können Sie die RPi.GPIO-Bibliothek mit Python verwenden. Für den Start empfiehlt sich zunächst eine einfache Schaltung, die sich in wenigen Minuten auf einem Steckboard umsetzen lässt. Die klassische LED (passenden Vorwiderstand nicht vergessen) wird über einen GPIO-Pin ein- und dann wieder ausgeschaltet (Bild 6).

LED-Praxis mit der RPi.GPIO-Bibliothek

Um die Funktion der RPi.GPIO-Bibliothek zu prüfen, werden nur wenige Zeilen Code für die Konfiguration, Initialisierung und Verwendung eines GPIO-Pins benötigt. Mit den systemnahen Dingen müssen Sie sich dann nicht mehr beschäftigen, das übernimmt die RPi.GPIO-Bibliothek, die am Anfang des Python-Codes mit dem *import*-Befehl in Ihr Python-Programm eingebunden wird:

```
#!/usr/bin/python
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM) # GPIO Mode
GPIO_LED = 17 # GPIO17 (Ausgang) / Pin 11 /
wiring Pi: 0
import time
import RPi.GPIO as GPIO
waitLED = 10
# RPi.GPIO BCM Layout verwenden
GPIO.setmode(GPIO.BCM)
# Pin 11 (GPIO 17) auf Output setzen
GPIO.setup(GPIO_LED, GPIO.OUT)
GPIO.output(GPIO_LED,GPIO.HIGH)
time.sleep(waitLED)
GPIO.output(GPIO_LED,GPIO.LOW)
```

Speichern Sie den Quellcode in eine Datei mit der Bezeichnung rpigpioled.py im Home-Verzeichnis (/home/pi) des Benutzers pi. Dafür nutzen Sie entweder auf der Kommandozeile den vi- oder – besser für Einsteiger – den Nano-Editor mit dem Kommando:

```
nano rpigpioled.py
```

Natürlich können Sie auch auf der GUI den grafischen Editor gedit verwenden – das Ergebnis ist dasselbe. Ist die Datei mit dem obigen Quellcode befüllt und auf der Speicherkarte gesichert, starten Sie das Programm mit dem Python-Interpreter:

```
sudo python rpigpioled.py
```

Damit führen Sie das obige Python-Beispiel mit der RPi.GPIO-Bibliothek auf dem Terminal aus – die auf dem Steckboard angeschlossene LED sollte nun leuchten und nach einem kurzen Augenblick dann wieder erlöschen.

PIR-Praxis mit der RPi.GPIO-Bibliothek

Analog zum obigen Shell-Skript mit der LED lässt sich der PIR-Sensor auch mit Python mittels der RPi.GPIO-Bibliothek unproblematisch einsetzen. Die Schaltung ist dieselbe – der Datenpin bleibt an GPIO17 gesteckt.

Ähnlich wie eine LED lässt sich auch ein Sensor mit der RPi.GPIO-Bibliothek verwenden. Da der Sen-

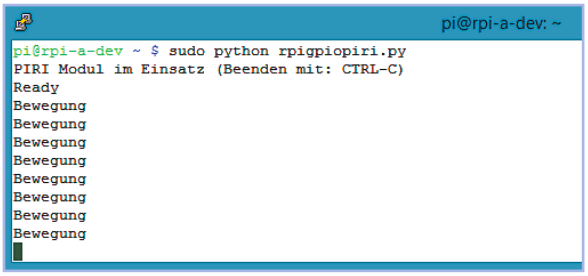


Bild 7: PIR-Sensor-Überwachung mit der RPi.GPIO-Bibliothek

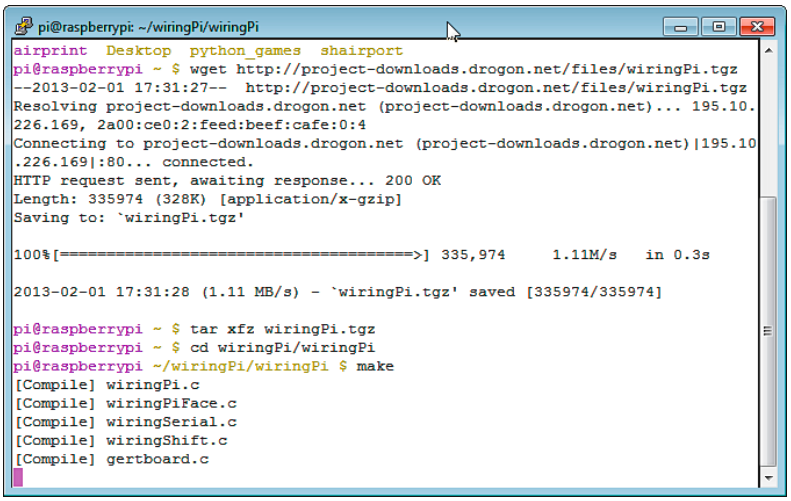


Bild 8: Traditionell: WiringPi erhalten Sie nach wie vor auch im tgz-Paket, das Sie allerdings zunächst noch per tar-Befehl entpacken und per make-Befehl kompilieren müssen.

sor vom Raspberry Pi überwacht werden soll, muss demnach der GPIO-Pin als Eingang (vom Sensor kommend) definiert werden, was in der nachstehenden Zeile erledigt wird:

```
GPIO.setup(piri, GPIO.IN)
```

Im nächsten Schritt läuft das Python-Programm in eine Endlos-Schleife, die mit der Tastenkombination [CTRL] und [C] unterbrochen werden kann. In dieser Schleife prüft das If-Konstrukt, ob am piri-Datenpin ein Signal eingetroffen ist oder nicht.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
piri = 7
waitpiri = 10

GPIO.setup(piri, GPIO.IN)

try:
    print „PIRI Modul im Einsatz (Beenden
mit: CTRL-C)“
    time.sleep(waitpiri)
    print „Ready“
    while True:
        if GPIO.input(piri):
            print „Bewegung“
        time.sleep(1)
except KeyboardInterrupt:
    print „Exit!“
    GPIO.cleanup()
```

Wird eine Bewegung erkannt, wird auf dem Terminal der schlichte Text „Bewegung“ ausgegeben – als Standardausgabe erscheint anderenfalls alle 10 s der Ready-Hinweis (Bild 7).

Nicht nur über Python, auch direkt auf der Shell lassen sich solche Prüfungen und kleinere Schaltungen einfach überwachen und steuern. Ein sehr beliebtes Werkzeug in der Praxis mit dem Raspberry Pi ist die WiringPi-API, die verschiedene und oftmals notwendige Dinge beim Umgang mit den GPIO-Anschlüssen auf der Shell in einfachen Kommandos zusammenfasst.

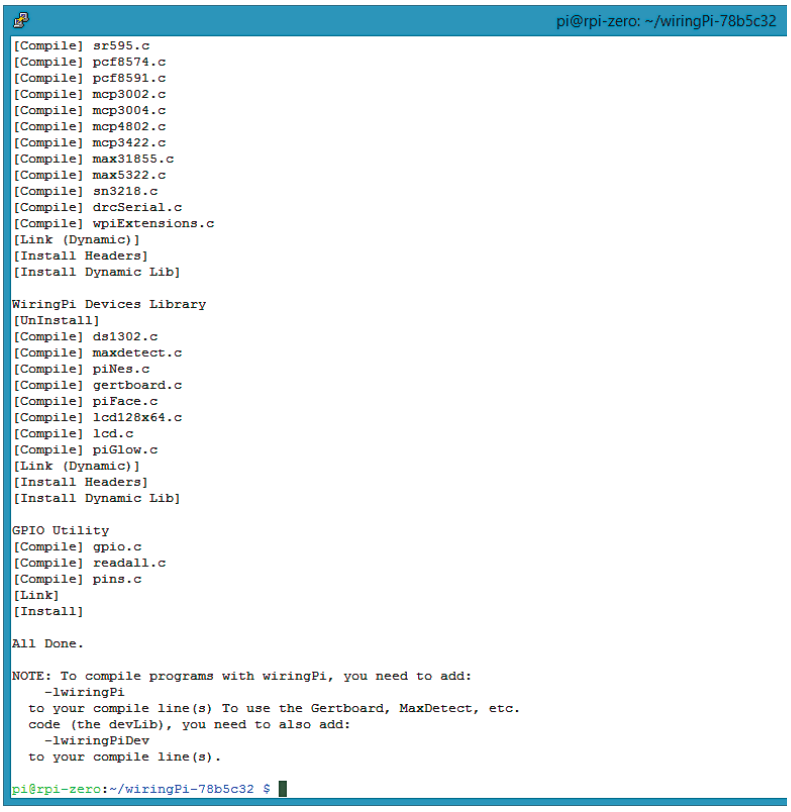


Bild 9: Nach dem Durchlauf des build-Skripts steht die API auf der Kommandozeile bereit.

WiringPi-API: Schnell auf der Shell

Wer jenseits der Programmiersprachen auf der Shell mal schnell einen oder mehrere GPIO-Pins konfigurieren und verwenden möchte, der musste sich jenseits des *raspi-gpio*-Kommandos bisher umständlich mit den Linux-Basis-Werkzeugen behelfen. Auch aus diesem Grund hat sich auf der Kommandozeile die äußerst praktische WiringPi-API des Autors Gordon Drogon (https://projects.drogon.net/raspberry-pi/wiringpi/download-and-install/) etabliert. Doch bevor Sie diese API installieren, sollten Sie darauf achten, den Raspberry Pi auf den aktuellsten Stand zu bringen und die GIT-Versionsverwaltung zu installieren. Wie gewohnt, nutzen Sie dafür das entsprechende *update-*, *upgrade-* bzw. *install*-Programm von Raspbian:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install git
```

Ist die GIT-Versionsverwaltung auf dem Raspberry Pi installiert, klonen Sie das WiringPi-Paket im ersten Schritt lokal auf den Raspberry Pi. Dies erledigen Sie mit dem Kommando:

```
cd ~
git clone git://git.drogon.net/wiringPi
```

Ist die GIT-Versionsverwaltung noch nicht installiert, wird dies entweder per Kommando

```
sudo apt-get install git-core
```

nachgeholt oder WiringPi gelangt per *wget*-Befehl in das aktuelle Verzeichnis auf der Speicherkarte des Raspberry Pi. Dafür ist schließlich eine händische Installation mit nachfolgenden Befehlen nötig:

```
wget http://project-downloads.drogon.net/
files/wiringPi.tgz
tar xfz wiringPi.tgz
cd wiringPi/wiringPi
make
sudo make install (Bild 8)
```

Im Fall der GIT-Versionsverwaltung ist der Quellcode bereits per Klon im aktuellen Verzeichnis. Mit dem *cd*-Kommando wechseln Sie dorthin:

```
cd wiringPi
git pull origin
```

Mit dem *git pull origin*-Befehl ist sichergestellt, dass die aktuellste WiringPi-Version verwendet wird – diese Methode empfiehlt sich auch für die Zukunft, um die WiringPi-Installation auf dem Raspberry Pi aktuell zu halten. Anschließend erstellen bzw. installieren Sie die WiringPi-API auf dem Raspberry Pi. Dafür verwenden Sie das *build*-Skript und starten es:

```
./build
```

Das *build*-Skript konfiguriert die API so, dass die bereitgestellten Funktionen umgehend verwendet werden können (Bild 9).

Nach der Installation stehen die nötigen Tools wie beispielsweise *gpio readall* auf der Kommandozeile zur Verfügung – im nächsten Abschnitt verwenden Sie einen einfachen Bewegungssensor am Raspberry Pi Zero, der auf der GPIO-Pin-Leiste nur drei Anschlüsse benötigt: Spannung, Masse und Datensignal.

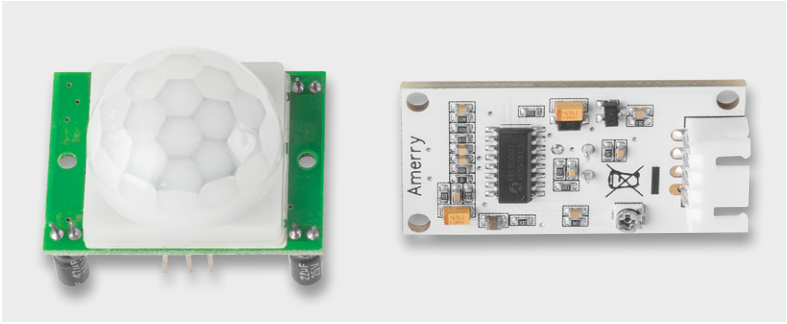


Bild 10: Groß und klein: Das linke PIR-Modul hat in etwa den Durchmesser einer 1-Euro-Münze. Das PIR-Modul rechts ist eine Linker-Kit-Platine mit PIR (CJ-12 50 75).

Bild 11: Sehr praktisch: Die 5-V-, Out- und GND-Anschlüsse können direkt an der Pin-Leiste des Raspberry Pi oder über ein Linker-Kit-Baseboard (CJ-12 21 37) angeschlossen werden.

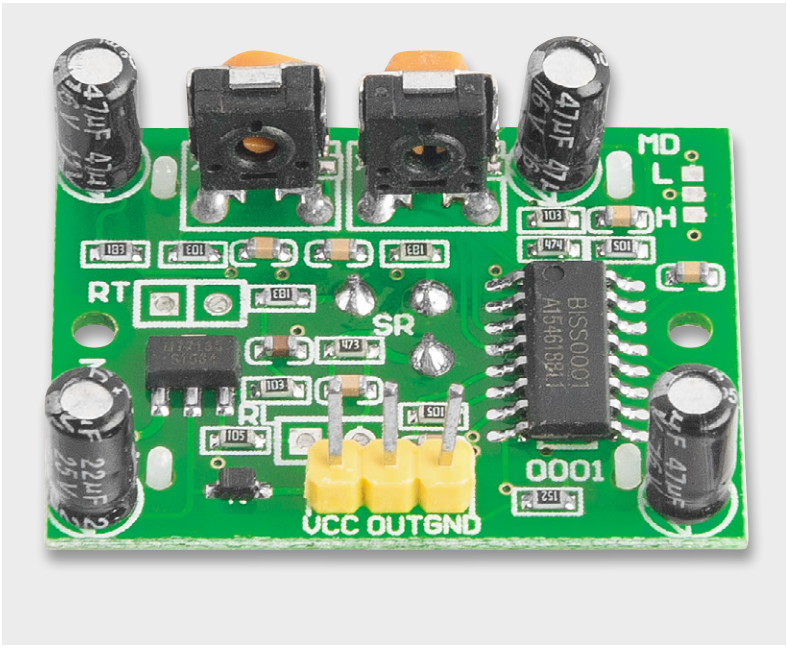


Bild 12: Pin-Belegung von links nach rechts: 5 V/Out/GND. Damit können Sie sich direkt mit dem Raspberry Pi verbinden.

PIR-Modul am Raspberry Pi

Ein für den Raspberry Pi geeigneter Bewegungssensor bzw. das PIR-Modul (PIR steht für passiver Infrarot-Bewegungsmelder) ist auf den einschlägigen Auktionsseiten im Netz für kleines Geld zu finden. Die Lieferanten der Billigmodule sitzen jedoch meist in China/Hongkong und die Lieferung dauert hier manchmal bis zu drei Wochen, doch der Preis von 2 Euro für 10 PIR-Module entschädigt für die Wartezeit. Vergessen Sie dabei die eventuell anfallenden Zollgebühren nicht! Die kleineren PIR-Module sind im Vergleich teurer, sie kosten pro Stück je nach Händler und Wartezeit 1 bis 2 Euro (Bild 10).

Egal, ob Sie das kleine oder das große PIR-Modul verwenden: Beide Module benötigen eine Versorgungsspannung von 5 V. Die Signalleitung liefert passende 3 V, sodass der Anschluss am Raspberry Pi direkt ohne irgendwelche Wandler erfolgen kann. Lediglich beim kleineren Modul sind zuvor die Kabel für den Anschluss an den Raspberry Pi zu verlöten. Hier nutzen Sie am besten Jumper-Kabel, die sich nach dem Löten einfach an die entsprechenden Pins am Raspberry Pi stecken lassen (Bild 11).

Die drei Pins des PIR-Moduls werden folgendermaßen mit dem Raspberry Pi verbunden:

PIR-Pin	Raspberry-Pi-Pin-Nr.	GPIO-Bezeichnung	WiringPi-Pin-Nr.
5 V	2	–	–
Out	26	GPIO7	11
GND	6	–	–

Bei den größeren Modulen sind dieselben Pins mit einer 3fach-Steckerleiste einfach zu erreichen. Nutzen Sie einfach die Jumper-Kabel für den Anschluss des PIR-Moduls am Raspberry Pi (Bild 12).

So kommen die Anschlüsse GND an Pin 6 (Masse) und 5 V an den 5-V-Anschlusspin 2, und der Zustand OUT wird in diesem Beispiel an GPIO7 (Pin 26) beim Raspberry Pi gesteckt. Ist die WiringPi-API installiert, prüfen Sie zunächst die Pin-Belegung (Bild 13).

Im nächsten Schritt erstellen Sie für die erstmalige Inbetriebnahme des PIR-Moduls ein einfaches Skript, was die Zustandsänderung des GPIO-Eingangs anzeigt.

Shell-Skript für PIR-Bewegungsmelder

Mit dem Nano-Editor erstellen Sie die Beispiel-Datei piri.sh und definieren dort den genutzten Pin-Anschluss. In diesem Beispiel ist der Out-Ausgang des Sensors auf Eingang GPIO7 (Pin 26) des Raspberry Pi gesteckt, der gleichbedeutend mit der WiringPi-Nummer 11 ist (Bild 14).

Anschließend legen Sie mit dem *gpio mode*-Kommando die Richtung des GPIO-Pins fest. Dann geht es in die While-Schleife, in der der Zustand des Eingangs per *gpio read*-Befehl permanent abgefragt wird.

```
#!/bin/bash
# Skript: piri.sh
# GPIO7 = Pin 26 = Wiring Pi-Pin 11
piri="11"
gpio mode $piri in
PiriBin=/usr/local/bin/piri.sh
LockFile=/var/lock/subsys/piri   ## Define
Lock Datei
echo „PIRI Modul im Einsatz
(Beenden mit: CTRL-C)"
while true; do
sleep 2
if [ $(gpio read $piri) -eq  0 ]; then
    echo „Ready";
   else
    echo „Bewegung"
   fi
done
exit 0
```

```
pi@rpi-zero:~/wiringPi-78b5c32 $ cd ..
pi@rpi-zero:~ $ gpio readall
 +-----+-----+---------+------+---+--Pi Zero--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     |  15 |  14 |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     |  16 |  15 |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 |   1 |  18 |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 |   4 |  23 |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 |   5 |  24 |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 |   6 |  25 |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     |  10 |   8 |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     |  11 |   7 |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   |  31 |   1 |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 |  26 |  12 |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 |  27 |  16 |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 |  28 |  20 |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 |  29 |  21 |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+--Pi Zero--+---+------+---------+-----+-----+
pi@rpi-zero:~ $
```

Bild 13: WiringPi-API im Einsatz: Mit dem Kommando gpio readall liefert der Raspberry den Status sämtlicher Pin-Einstellungen der GPIO-Stiftleiste zurück.

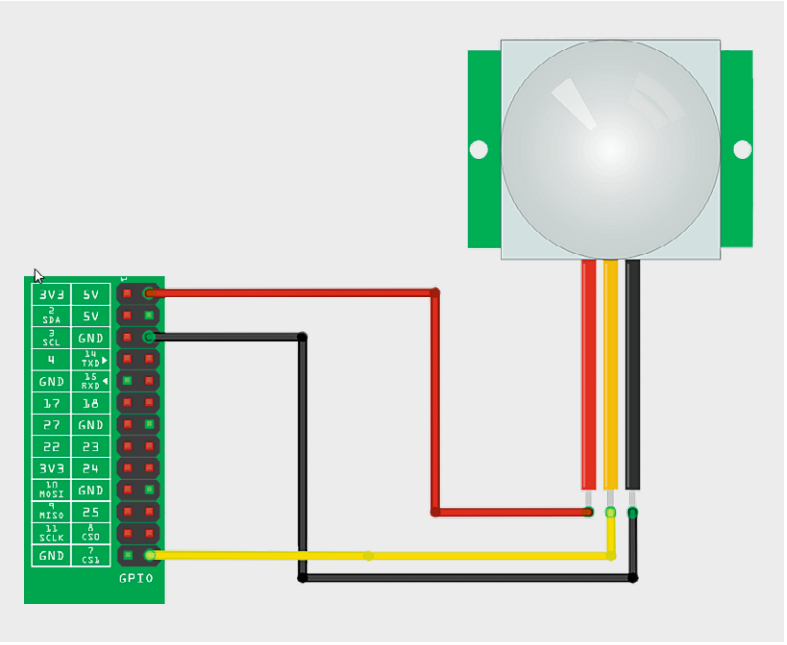


Bild 14: Neben Spannung und Masse benötigt der PIR-Sensor einen Dateneingang auf der GPIO-Leiste – hier wird GPIO7 genutzt.

```
pi@rpi-a-dev: ~
pi@rpi-a-dev ~ $ bash piri.sh
PIRI Modul im Einsatz (Beenden mit: CTRL-C)
Bewegung
Ready
Ready
Ready
Ready
Ready
Bewegung
Ready
Ready
Bewegung
Ready
Ready
Ready
Ready
Ready
Ready
Ready
Ready
Bewegung
Bewegung
Ready
Ready
Ready
Bewegung
Ready
Ready
Bewegung
Ready
Ready
Ready
Ready
```

Bild 15: Das Beispielskript bei der Ausführung: Wird eine Bewegung erkannt, wird der Output-Pin auf der PIR-Platine auf den Zustand HIGH gesetzt, was wiederum beim festgelegten GPIO-Eingang des Raspberry Pi für eine Zustandsänderung sorgt.

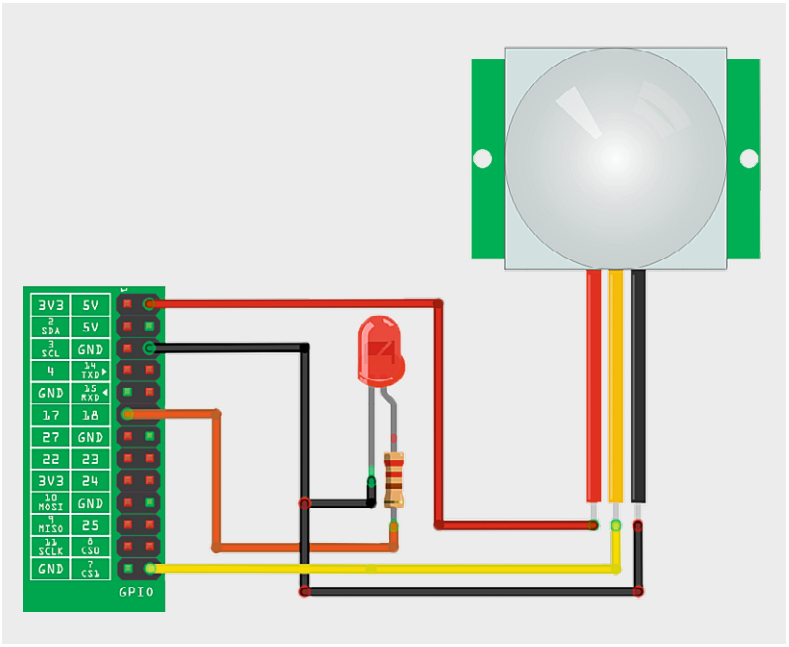


Bild 16: All inclusive: Neben dem PIR-Sensor ist nun auch die LED am GPIO-Anschluss im Einsatz.

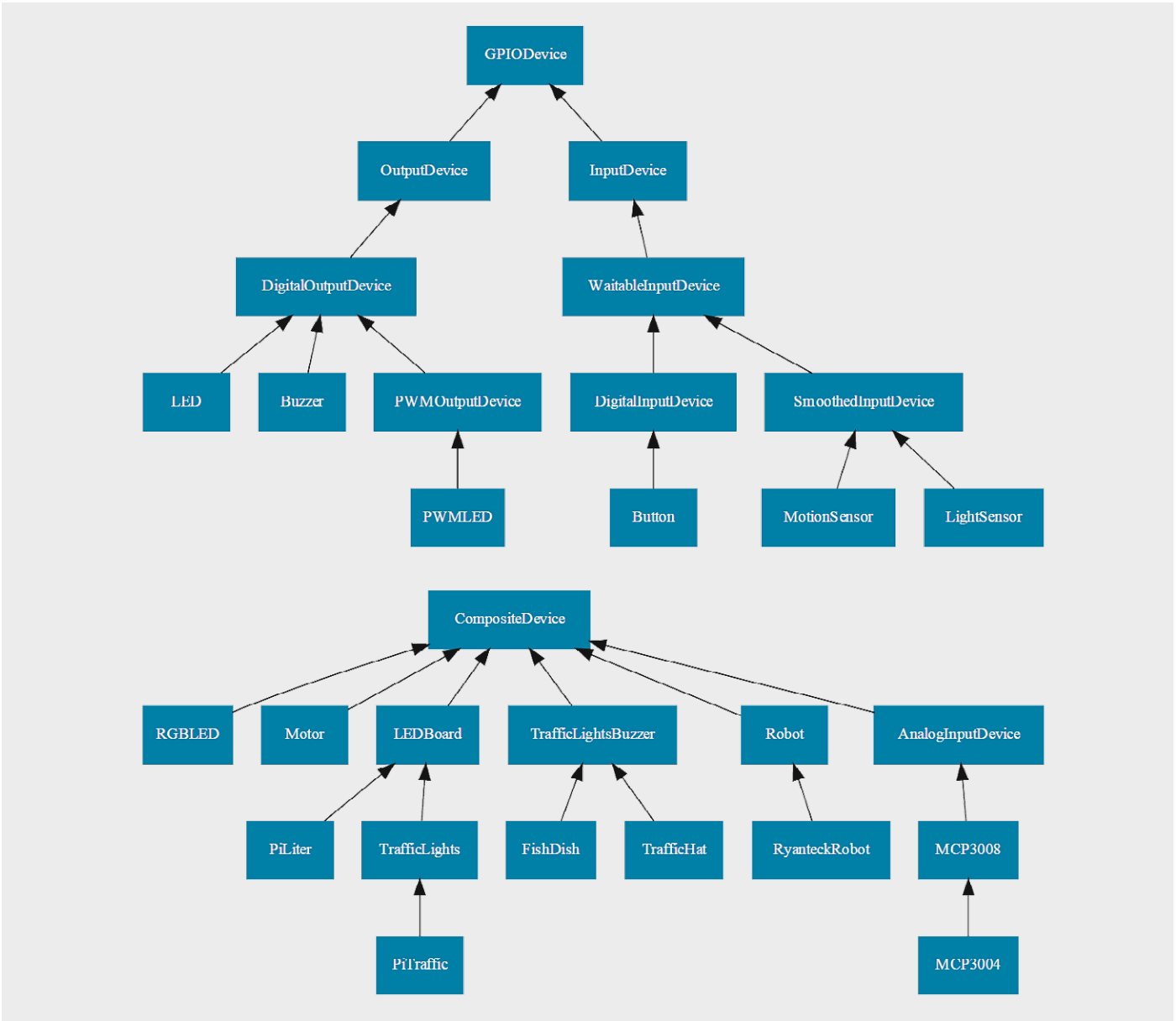


Bild 17: Dokumentation der gpiozero-Bibliothek (https://gpiozero.readthedocs.org/en/rest-docs/api_generic.html): Neben den in den Beispielen verwendeten LED- und MotionSensor-Klassen sind noch zahlreiche weitere Klassen in der API definiert.

Das Skript lässt sich starten, falls es zuvor mit dem Kommando *chmod a+x* auf ausführbar gesetzt worden ist. Bei der Ausführung befindet sich das Skript sozusagen in der Dauerschleife (konkret der While-Schleife) und prüft permanent den Status des Eingangs am GPIO7 ab. Meldet der Bewegungsmelder eine Zustandsänderung, wird diese auf der Konsole angezeigt, ansonsten wird eine Ready-Meldung ausgegeben (Bild 15).

Erfolgt hingegen keine Bewegung am PIR-Modul, ist der Output-Pin auf der PIR-Platine auf 0 V gesetzt und der GPIO-Eingang befindet sich im Leerlauf.

gpiozero-Bibliothek im Einsatz

Mit dem Erscheinen des Raspberry Pi Zero Ende 2015 wurde von der Raspberry Pi Foundation für den Python-Einsatz auch eine neue Bibliothek gpiozero (https://gpiozero.readthedocs.org/en/rest-docs/api_generic.html) vorgestellt, die ähnlich wie die bekannte WiringPi-Bibliothek bzw. die bekannte RPi.GPIO-Bibliothek einen verbesserten und vereinfachten Zugriff auf die einzelnen GPIO-Pins ermöglicht. Mit dem aktuellen Raspbian-Jessie-Image für den Raspberry Pi Zero ist auch die neue gpiozero-Bibliothek für Python 2.7 und Python 3 mit an Bord. Mit einem schlichten

```
sudo apt-get update && sudo apt-get upgrade
```

landet das Paket auf der Speicherkarte. Alternativ lassen sich beide Python-Pakete auch mit dem *apt-get*-Kommando für Python 2.7 und Python 3 auf die Speicherkarte des Raspberry Pi holen:

```
sudo apt-get install python-gpiozero
python3-gpiozero
```

Wie die großen Geschwister besitzt der Raspberry Pi Zero auch eine voll funktionsfähige GPIO-Leiste mit 40 Pins – die Stiftleiste zum Einstecken der kleinen Anschlusskabel ist aus Platzgründen jedoch nicht im Lieferumfang. Hier haben Sie zwei Möglichkeiten: Entweder Sie löten eine Stiftleiste (oder wie bei einem Arduino eine Buchsenleiste) nachträglich an oder Sie verbinden das gewünschte Anschlusskabel mit dem Lötkolben platzsparend direkt auf den gewünschten Pin auf der GPIO-Leiste des Raspberry Pi Zero. Wer keinen Raspberry Pi Zero im Einsatz hat, benötigt diese Lötarbeiten und Vorbereitungen nicht. Bei den „großen" Raspberry-Pi-Modellen 1/2/3 ist die Pin-Leiste bereits auf der Platine verlötet und die gpiozero-Bibliothek lässt sich naturgemäß auch mit diesen Platinen verwenden.

LED-Praxis mit der gpiozero-Bibliothek

Ist beispielsweise an Pin 11 eine LED angeschlossen, verwenden Sie die neue gpiozero-Bibliothek wie folgt:

```
#!/usr/bin/python
from gpiozero import LED
from time import sleep
waitLED = 10
myled = LED(17)  # GPIO17 = Pin 11
myled.on()
sleep(waitLED)
myled.off()
```

Das einfache Beispiel können Sie entweder als eigene Datei – beispielsweise mit dem Namen gpiozeroled.py – auf dem Raspberry Pi speichern und anschließend mit dem Kommando

```
sudo python gpiozeroled.py
```

ausführen.

Für schnelle Experimente lässt sich auch die Python-Konsole nutzen, die einfach mit dem Kommando *python* (oder *python3* für Python 3) im Terminal gestartet wird. Nun tippen Sie vom obigen Quellcode beginnend bei der From-Anweisung Zeile für Zeile in den Interpreter ein. Mit der [Eingabe]-Taste bestätigen Sie jede Zeile. Eine Rückmeldung erfolgt in der Regel nur, wenn ein Fehler auftritt. Die LED des obigen Testaufbaus sollte nun wie gewünscht leuchten. Beenden lässt sich die Python-Konsole mit dem *exit()*-Befehl.

PIR-Bewegungsmelder mit der gpiozero-Bibliothek

Soll zusätzlich noch ein Bewegungssensor zum Einsatz kommen (Bild 16), steht wie bei einer LED eine eigene Klasse von der gpiozero-Bibliothek dafür zur Verfügung und wird eingangs analog definiert.

Schließlich brauchen Sie nur noch den Anschluss-Pin für den Datenanschluss des Bewegungssensors anzugeben (hier GPIO7/Pin26).

```
from gpiozero import MotionSensor, LED
pir = MotionSensor(7)
myled = LED(4)
while True:
    pir.when_motion = myled.on
    pir.when_no_motion = myled.off
```

Wird vom Bewegungssensor eine Bewegung erkannt, wird umgehend über den *myled.on*-Aufruf die LED zum Leuchten gebracht, anderenfalls bleibt die LED aus.

Viele Klassen und Funktionen

Die gpiozero-Bibliothek bringt neben dem einfachen LED-Beispiel auch eine direkte Unterstützung von weiteren Geräten wie Schalter, Licht-Sensoren, Motoren, dem bekannten AD-Wandler MCP3008, PIR-Sensoren usw. mit (Bild 17).

Bei bestehenden und funktionierenden Projekten lohnt sich der Aufwand des Umstiegs und Neuprogrammierung derzeit eher nicht – doch bei neuen Projekten verspricht der Ansatz einen schlankeren und somit performanteren Python-Code. Ob der Einsatz der Python-Beispiele von einem Original-Raspberry Pi später auch auf Raspberry-Pi-Klons wie dem Banana Pi möglich sein wird, hängt dort von der Portierung der gpiozero-Bibliothek ab. ELV

Weitere Infos:

www.elv.de/raspberry-pi-minicomputer.html

www.elv.de/linker-kit-system.html