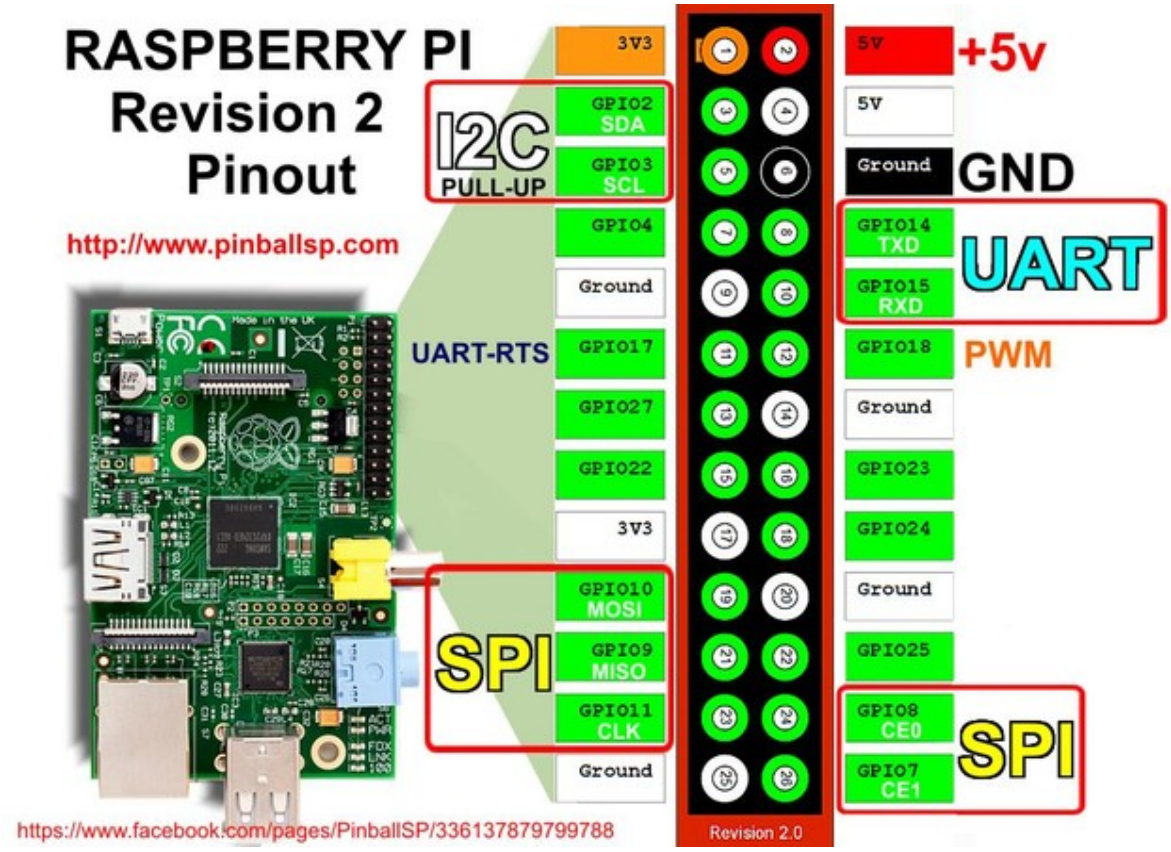
GPIO



LED an GPIO-Ausgang

Beispiel: LED über 470Ω an GPIO25 (Pin 22 am Stecker)

GPIO Out über das Betriebssystem

- Pin in den Userspache exportieren:

  ```
  echo 25 > /sys/class/gpio/export
  ```

  Dadurch wird ein neues Verzeichnis /sys/class/gpio/gpio25 erstellt.
  (Mit ls kontrollieren!)

- In das Verzeichnis gpio25 wechseln.

  In gpio25 gibt es folgende Dateien:

  ```
  active_low direction edge power subsystem uevent
  value
  ```

- Datenrichtung auf Ausgang festgelegen:

  ```
  echo out > direction
  ```

- Zustand des Ausgangs festlegen:

  ```
  echo 1 > value
  echo 0 > value
  ```

Die Dateien export, direction, value, ... sind keine echten Dateien, sondern virtuelle Dateien zum Zugriff auf die Hardware.

Ein GPIO-Anschluss kann je nach Konfiguration 2 ... 16mA liefern (standardmäßig 8mA).
Die Konfiguration geschieht über ein spezielles Register.

GPIO Out über Python

RPi.GPIO Doku:
http://code.google.com/p/raspberry-gpio-python/wiki/Examples

Achtung: Python muß mit Root-Rechten aufgerufen werden wenn auf die Hardware zugegriffen wird.

Test
Eventuell ist die GPIO-Library schon installiert. Dies kann man leicht testen:

```
sudo python
>>> import RPi.GPIO as GPIO
```

Wenn eine Fehlermeldung kommt muß die Library noch installiert werden.

Installation der GPIO-Library:

Am einfachsten über den Paketmanager:

```
sudo apt-get update
sudo apt-get install python-rpi.gpio
```

oder

```
sudo apt-get install python-dev
wget
http://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-
0.5.0a.tar.gz
tar -zxf RPi.GPIO-0.5.0a.tar.gz
cd RPi.GPIO-0.5.0a
sudo python setup.py install
```

Ausgang schalten

Es gibt zwei Möglichkeiten der Pin-Nummerierung:

GPIO.setmode(GPIO.BOARD) benutzt die Nummerierung des Steckers auf dem Board
GPIO.setmode(GPIO.BCM) benutzt die Nummerierung der GPIO-Pins

```
sudo python
>>>import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)    #"normale" Nummerierung
>>> GPIO.setup(25,GPIO.OUT)   # GPIO25 als Ausgang

>>> GPIO.output(25,0)         # GPIO25 auf 0
>>> GPIO.output(25,1)         # GPIO25 auf 1
```

Blinken

Mit dem Editor (z.B. nano im Textmodus, oder besser Geany) ein Programm erstellen und als blink.py speichern:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(25,GPIO.OUT)

while True:
        GPIO.output(25,1)
        time.sleep(1)
        GPIO.output(25,0)
        time.sleep(1)
```

Dieses mit Rootrechten unter Python ausführen:

```
sudo python blink.py
```

Abbruch der Endlosschleife mit <Ctrl>C

Alternativ zum Aufruf mit Python kann das Skript auch eigenständig gestartet werden:

- Zu Beginn des Programms die magische Zeile
  `#!/usr/bin/python`
  einfügen, damit der Interpreter automatisch gefunden wird

- Die Datei blink.py muß ausführbar gemacht werden:
  `sudo chmod +x blink.py`

- Aufruf:
  `sudo ./blink.py`

Achtung:

Dieses einfache Programm macht keine Aufräumarbeiten wenn es mit <Ctrl>-C unterbrochen wird. Dies kann dazu führen, dass der benutzte Pin in einer alternativen Anwendung z.B. SPI anschließend nicht funktioniert.

Es istr also eine gute Idee, beim Unterbrechen des Programms GPIO.cleanup() auszuführen. Hierzu wird der KeyboardInterrupt genutzt, um das Programm ordnungsgemäß (mit Aufräumen) zu beenden:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(11,GPIO.OUT)

while True:
    try:
```

```
        GPIO.output(11,1)
        time.sleep(1)
        GPIO.output(11,0)
        time.sleep(1)
    except KeyboardInterrupt:
        GPIO.cleanup()
```

Taster am GPIO-Eingang

Beispiel:
Taster nach +3.3V (nicht 5V!) mit Pulldown 10k am Eingang GPIO24 (Pin 18 des Steckers).

In Python kann auch ein interner Pulldown eingeschaltet werden mit

```
GPIO.setup(24, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
```

GPIO Input über das Betriebssystem

- Alle Befehle mit Root-Rechten:
  ```
  sudo su
  ```
- Pin in den Userspache exportieren:
  ```
  echo 24 > /sys/class/gpio/export
  ```
- In das Verzeichnis gpio24 wechseln und Datenrichtung auf Eingang festgelegen:
  ```
  cd /sys/class/gpio/gpio24
  echo in > direction
  ```
- Zustand des Eingangs anzeigen:
  ```
  cat value
  ```

GPIO Input mit Python

Programm taster.py mit Polling:

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(24,GPIO.IN)

while True:
        taster = GPIO.input(24)
        print taster                # 1/0 ausgeben je nach Wert
        time.sleep(0.02)
```

```
sudo python ./taster.py
```

Der sleep – Befehl gibt Resourcen für andere Programme frei die nebenher laufen.

GPIO-Interrupts

Die Versionen ab 0.5.1 von RPi.GPIO erlauben Interrupts.
Ein Update des Raspi vom 5. Oktober 2013 enthält die Version 0.5.3 (von aptitude angezeigt).

In Python scheint mit GPIO.VERSION die Version falsch angezeigt zu werden.

Beispiel:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(24, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

# ISR
def myInterrupt():
        print "Interrupt!"

# Add interrupt event for Pin 24, rising edge
GPIO.add_event_detect(24, GPIO.RISING, callback = myInterrupt)

# Infinite loop or other activities
while True:
        time.sleep(1)
```

Zunächst wird die Callback-Funktion myInterrupt definiert. Hier wird festgelegt, was beim Interrupt passieren soll, im Beispiel einfach eine Textmeldung.

Anschließend wird mit GPIO.add_event_detect festgelegt, daß beim hardwaremäßigen Interruptereignis die oben definierte Interruptfunktion aufgerufen wird.

Achtung: in vielen Internetbeispielen wird der Funktion myInterrupt eine Variable "channel" übergeben. Dies führte bei mir zu Fehlermeldungen, außerdem ist die Bedeutung dieser Variablen nicht klar. Laut Wiki soll sie die Pinnummer sein, aber die wird ja schon in add_event_detect festgelegt. Das obige Programm funktionierte korrekt (Okt. 2013).