Physik mit dem Raspberry Pi - GPIO Einführung

23.04.2018, Sebastian Schmidt (sebastian.seb.schmidt@fau.de)

github.com/schmidtseb/RaspberryPiUebung

Initialisierung

Laden des Moduls

Bevor ein Austausch von Daten über die GPIO-Schnittstelle (*G*eneral *P*urpose *I*nput *O*utput) erfolgen kann, muss diese initialisiert werden.

Zuerst muss daher das entsprechende Modul in Python geladen werden:

```
import RPi.GPIO as GPIO
```

Ab diesem Moment stehen die Funktionen des Moduls über den Namensbereich **GPIO** zur Verfügung. Ein Zugriff erfolgt über den **.**-Operator.

Deklaration der Pin-Nummerierung

Die Nummerierung der Pins der GPIO-Schnittstelle kann anhand von zwei System erfolgen:

- `GPIO.BOARD`: Die Nummerierung der Pins ist analog zur Beschriftung auf dem Board
- `GPIO.BCM`: Broadcom Chip spezifische Nummerierung, die der Deklaration der Pins des besagten Chips entspricht

Der entsprechende Modus kann z.B. über

```
GPIO.setmode(GPIO.BOARD)
```

gesetzt werden. Das Setzen des Modus ist nicht optional, sondern muss rfolgen.

Setzen des Pin-Modus

Jeder Pin der Schnitstelle kann als Ein- oder Ausgang eingesetzt werden. Dies muss entsprechend festgelegt werden. Dabei kommen die folgenden Parameter zum Einsatz:

- `GPIO.IN`: Eingang

- `GPIO.OUT`: Ausgang

Ein Beispiel für die Deklaration des Pins mit der Nummer 5 als Eingang, wobei die Nummerierung anhand einer der vorher beschriebenen Schemata erfolgt, geschieht dann über die Funktion

```
GPIO.setup(5, GPIO.IN)
```

Input/Output

Setzen von Ausgangswerten

Ein digitaler Ausgang der GPIO-Schnittstelle hat zwei mögliche Zustände:

- `GPIO.LOW`: 0V
- `GPIO.HIGH`: 3.3V

Wahlweise können auch die gleichwertigen Werte von `True` und `False` und `1` und `0` verwendet werden.

Um beispielsweise den Wert des Pins mit der Nummer 5 auf high zu setzen, kommt die folgende Funktion zum Einsatz

```
GPIO.output(5, GPIO.HIGH)
```

Lesen von Eingangswerten

Wenn ein Pin als Eingang gesetzt wurde, kann dessen Zustand über die Funktion

```
GPIO.input(<Pin-Nr>)
```

ausgelesen werden. Der Rückgabewert ist dabei entweder `GPIO.LOW` oder `GPIO.HIGH`.

SPI

Definition

Die Abkürzung SPI steht für das *S*erial *P*eripheral *I*nterface. Ein serielles Bus-System, das die Verbindung von Komponenten nach dem Master-Slave-Prinzip erlaubt. Hierbei stellt der Raspberry Pi stets den Master dar, die an ihm angeschlossenen Geräte die Slaves. Seriell bedeutet, dass lediglich eine Datenleitung existiert, wobei der Datentransfer über ein Clocksignal gesteuert wird.

Der Bus besteht dabei aus den folgenden Leitungen:

- *SCLK* (Serial Clock): wird vom Master zur Synchronisation ausgegeben
- *MOSI* (Master Output, Slave Input)
- *MISO* (Master Input, Slave Output)
- *SS* (Slave Select): häufig auch *CS* (Chip Select) genannt. Wird genutzt, um einen an das System angeschlossenen Slave zu selektieren und aktivieren

Bevor eine Kommunikation zwischen Geräten stattfinden kann, müssen Pins an der GPIO-Schnittstelle des Raspberry Pis als SPI definiert werden. Dies geschieht über die `GPIO.output()` -Funktion.

Chip Select

Die Selektion des Slaves erfolgt über

```
GPIO.output(CSPin, GPIO.HIGH)
GPIO.output(CSPIN, GPIO.LOW)
```

d.h. durch ein An- und Ausschalten des Ausgangs. Grund hierfür ist, dass der CS-Eingang am ADC negiert ist (zu erkennen an den überstrichenen Buchstaben: $\overline{CS}$). D.h. durch ein Anschalten des Ausgangs ist der ADC nicht aktiv, wird es jedoch, sobald der Ausgang wieder ausgeschaltet wird.

Senden eines Kommandos

Ein Kommando ist eine Bitfolge, die mittels SPI-Schnittstelle von Master zu Slave über die MOSI-Datenleitung gesandt wird.

Im Programmcode findet das Senden eines Kommandos über ein 'Herausschieben' der Daten statt. Der Wert des aktuell zu sendenden Bits wird auf den Pin des MOSI übertragen.

Anschließend wird über

```
GPIO.output(SCLKPin, GPIO.HIGH)
GPIO.output(SCLKPin, GPIO.LOW)
```

ein Clock-Signal generiert, das dem ADC das Senden eines Bits signalisiert. Daraufhin wird das Kommando eine Stelle nach links geschoben und das nächste Bit wird gesendet.

Analog to Digital Converter (ADC)

Verwendet wird der *MCP3208* ADC mit einer Auflösung von 12 Bit, 8 Kanälen und einer SPI-Schnittstelle.

Selektion eines Kanals

Um Daten vom ADC zu lesen, muss dieser erst mit einem Kommando angesprochen werden. Dieses hat hierbei eine Länge von 5 und die folgende Form:

`0b000STKKK`

- **S**: Um den Start eines Kommandos zu signalisieren, beginnt dieses mit einem Startbit.
- **T**: Wird das nachfolgende Bit gesetzt, so wird lediglich eine positive Spannung gemessen. Bei Nicht-Setzen des Bits kann eine Spannung beider Vorzeichen gemessen werden. Hierbei dient muss ein Pin als positiver und ein weiterer als negativer Eingang am ADC genutzt werden. Im Folgenden sei die Nutzung jedoch auf die Verwendung von positiven Spannungen begrenzt.
- **K**: Die folgenden drei Bits selektieren den entsprechenden Kanal des ADCs (Umandlung von binär zu dezimal entspricht Nummer des Kanals).

Im Programmcode wird die Erzeugung des Kommandos wiefolgt gehandhabt. Der anzusprechende Kanal des ADC kann als Integer angegeben werden:

```
sendcmd = adcChannel
```

Durch

```
sendcmd |= 0b00011000
```

werden die Bits **S** und **T** gesetzt.

Eine Schleife über die fünf relevanten Bits des Kommandos wird gestartet. In jedem Schleifendurchgang mittels

```
sendcmd & 0x10
```

stets lediglich das fünfte Bit des Kommandos betrachtet. Die Zahl `0x10` ist hierbei hexadezimal und entspricht `0b0001000`.

Auslesen des ADCs

Zum Auslesen des ADCs wird zuerst eine Variable mit 0 initialisiert. Anschließend wird für 13 Clock-Signale der MISO-Pin der GPIO-Schnittstelle ausgelesen. Ist dieser aktiv, so wird mittels

```
adcValue |= 0x01
```

das entsprechende Bit gesetzt. Daraufhin wird der Wert um eine Position nach links verschoben und das nächste Bit ausgelesen.