Karlsruher Institut für Technologie

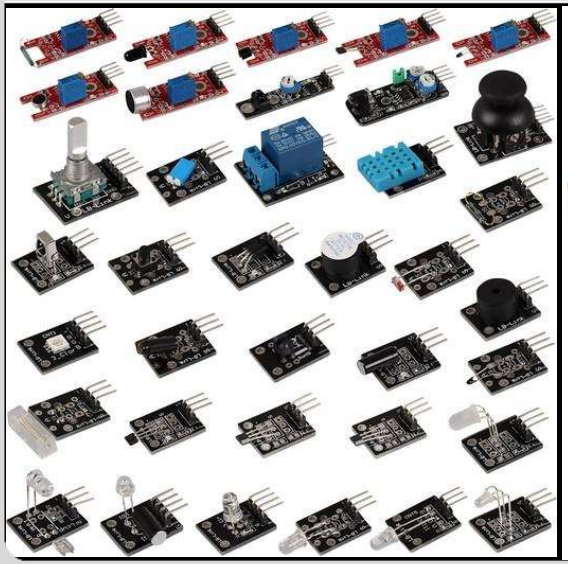
Digitales Messen im Physikunterricht
mit dem Raspberry Pi

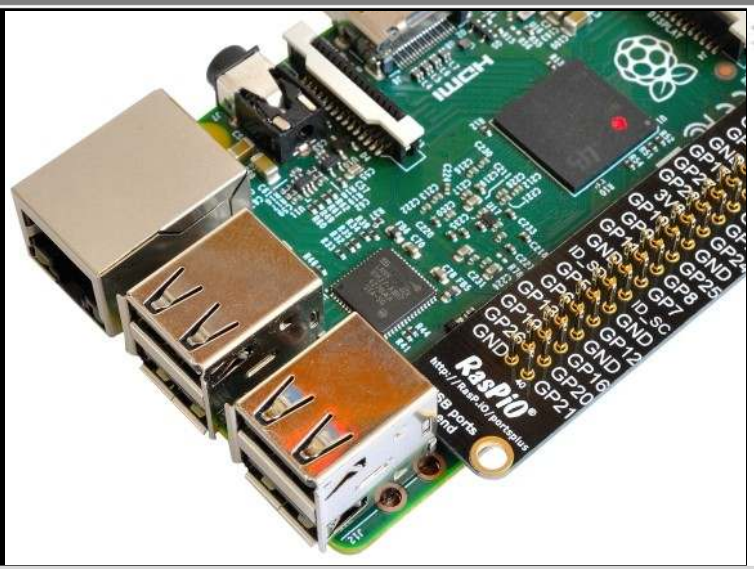
Günter Quast

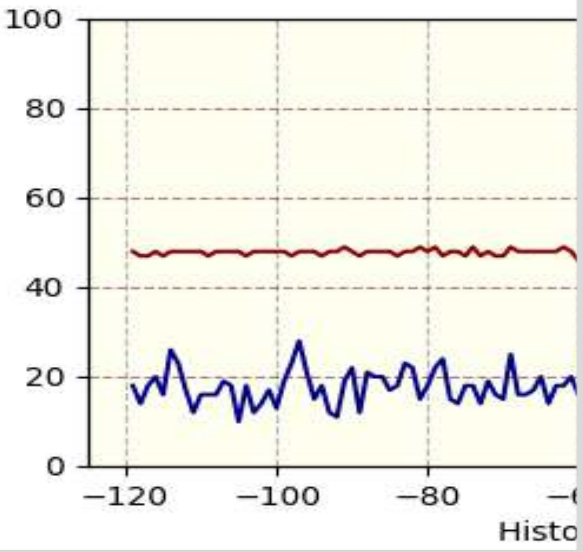
Fakultät für Physik
Institut für Experimentelle Teilchenphysik

Pforzheim 2019

13. Februar 2019







Einleitung:

Digitale Messtechnik und PhyPiDAQ

Digitalisierung der Messtechnik

Digitale Technologien prägen zunehmend unseren Alltag

- Sensoren zur Messwerterfassung
  in praktisch allen Alltagsgeräten

- Zusammenführung und Analyse der Daten
  - zur Geräte- und Systemdigagnose
  - zur Anwenderinformation
  - als Grundlage für Steuer-und Regelungsaufgaben
  - !!! zur Analyse des Nutzerverhaltens und Werbung

- Messverfahren in Technik und Wissenschaft heute
  ausnahmslos digital

- !!! Datenaufzeichnung und Zusammenführung
  in Internet-Datenbanken
  auch Sensordaten sind eine wichtige Komponente
  im Umfeld von „Big Data Analytics“

- Das „Internet der Dinge“ und „Industrie 4.0“
  stellen besondere Herausforderungen an die
  digitale Kompetenz von Schulabgängern

Digitalisierung im schulischen Kontext

Digitale Medien in regulären Lehr- und Lernprozessen einsetzen

Fachspezifische Beiträge der Physik:

- **Algorithmen erkennen und formulieren**
  - Funktionsweise und grundlegende Prinzipien der digitalen Welt kennen und verstehen
  - Technische Probleme identifizieren
  - Bedarfe für Lösungen ermitteln und Lösungen finden bzw. Lösungsstrategien entwickeln

- **Problemlösen und Handeln**
  - Eine Vielzahl von digitalen Werkzeugen kennen und kreativ anwenden
  - Anforderungen an digitale Werkzeuge formulieren
  - Passende Werkzeuge zur Lösung identifizieren

  - Digitale Umgebungen und Werkzeuge zum persönlichen Gebrauch anpassen

KMK vom 8.12.2016

https://www.kmk.org/fileadmin/Dateien/pdf/PresseUndAktuelles/2018/Digitalstrategie_2017_mit_Weiterbildung.pdf

Digitale Messtechnik in Schülerhände !

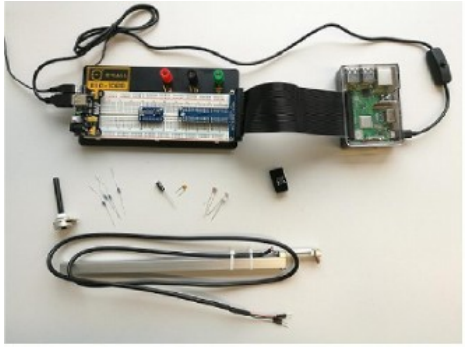
Digitale Messtechnik gibt es seit langem im Physikunterricht

- wegen des hohen Preises fast ausschließlich in Demonstrationsexperimenten

- relativ leicht aufzubauen, Softwareunterstützung für fast jede Messaufgabe

- wird von Schülern nicht all „alltagsnah“ empfunden

Die Alternative:
analoge und digitale Sensoren
und Einplatinencomputer
zur Datenerfassung und Auswertung



- kostengünstig, alltagsnah

- in Demonstrationsexperimenten aufwändig

Arduino oder Raspberry Pi ?

Arduino: Mikrocontroller auf Basis eines ATmega328 oder neuer, kein eigenes Betriebssystem, Programmierung in C, nicht Multi-Tasking-fähig

Eignet sich hervorragend für einzelne Anwendungen

Raspberry Pi: Einplatinen-Computer auf Basis von Handy-Technologie, vollständiges Betriebssystem, Programmierung vorzugsweise in Python oder mit grafischer Oberfläche *(Scratch)*, multi-tasking und netzwerkfähig

Vollständiger „PC“ zur Entwicklung von Projekten, zur Datenaufnahme, Visualisierung und Auswertung

Für beide Plattformen gibt es eine große Anzahl an Sensoren und Unterstützung im Internet

Als Universalsystem zur digitalen Datenerfassung und Auswertung viel unsere Wahl auf den Raspberry Pi

Sensoren

Der Handel bietet
eine Vielzahl an
Sensoren zum
Preis von 1 bis ca. 10 €,
häufig als Sensor-Set:

Temperatur, Magnetfeld,
Druck, Licht, Infrarot,
Schall, Ultraschall,
Vibration, Lage, …

**sowie Komponenten
zur Steuerung und Regelung:**

Analog-Digital-Wandler,
Relais, Drehgeber,
Leuchtdioden ...

PhyPiDAQ

Standardisierter Messkoffer

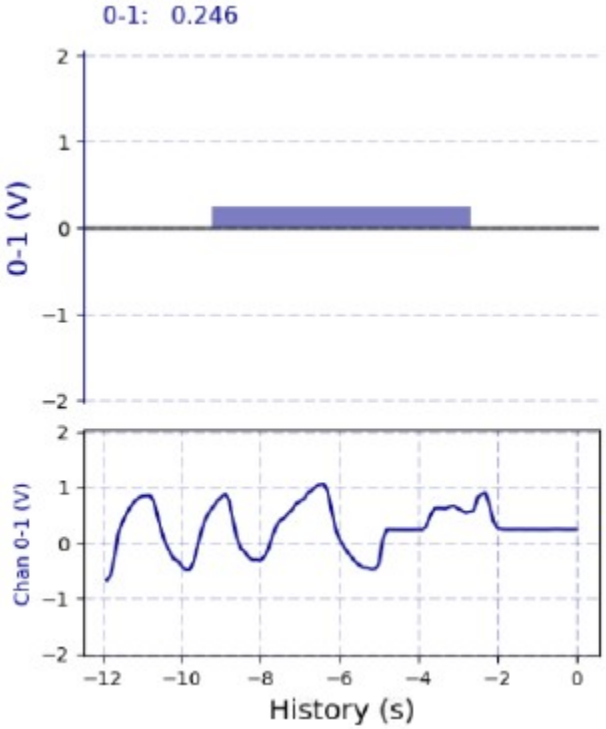
zur Einführung von Schülern in
die grundlegenden Konzepte der
Digitaltechnik und des digitalen Messens

Standardisierte Software-Umgebung für typische Messaufgaben:

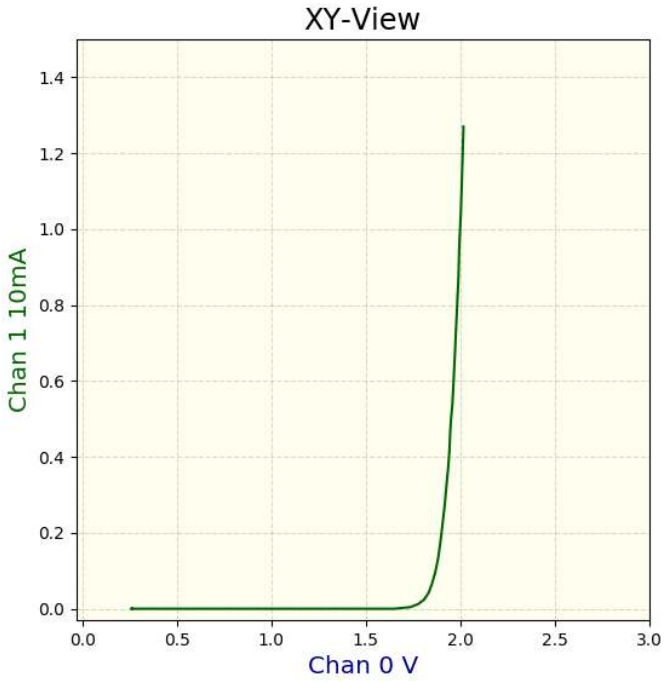
- einheitliche Schnittstelle für eine Vielzahl von Sensoren, erweiterbar
- graphische Anzeigen
  (Voltmeter, Oszilloskop, xy-Anzeige, zeitlicher Verlauf von Messgrößen)
- Datenaufzeichnung (im CSV-Format)
- Kalibration von (nicht-linearen) Sensoren
- Anwendung von Formeln auf Messwerte

```
import numpy as np, time
from phypidaq.ADS1115Config import *

device = ADS1115Config()
device.init() # Initialisierung

dt = 1.    # Ausleseintervall in s
T0 = time.time() # Start-Zeit
dat = np.array([0.])

print(' beginne Auslese, <ctrl-C> = stop')

while True:
 device.acquireData(dat)
 dT = time.time() - T0
 print('%.2g, %.4g' %(dT, dat) )
 time.sleep(dt)
```

PhyPiDAQ: Anzeigen

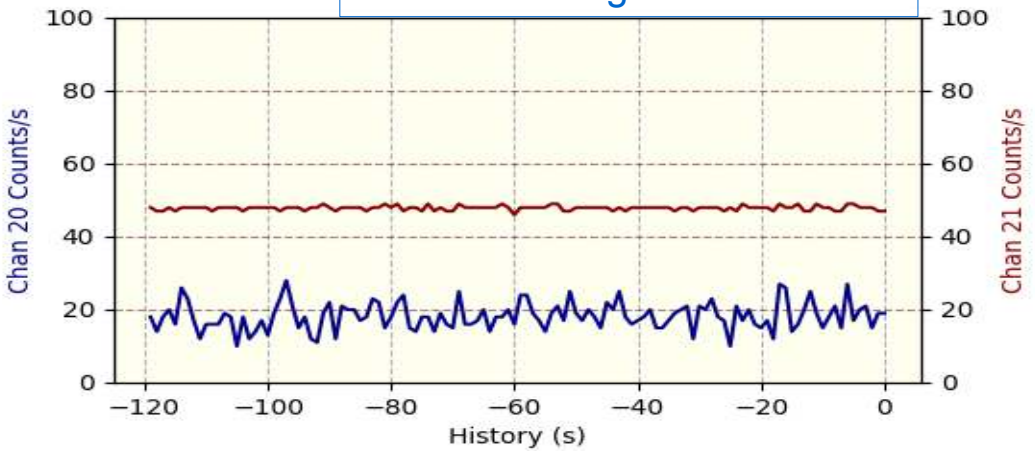


Balkenanzeige und zeitlicher Verlauf eines Signals

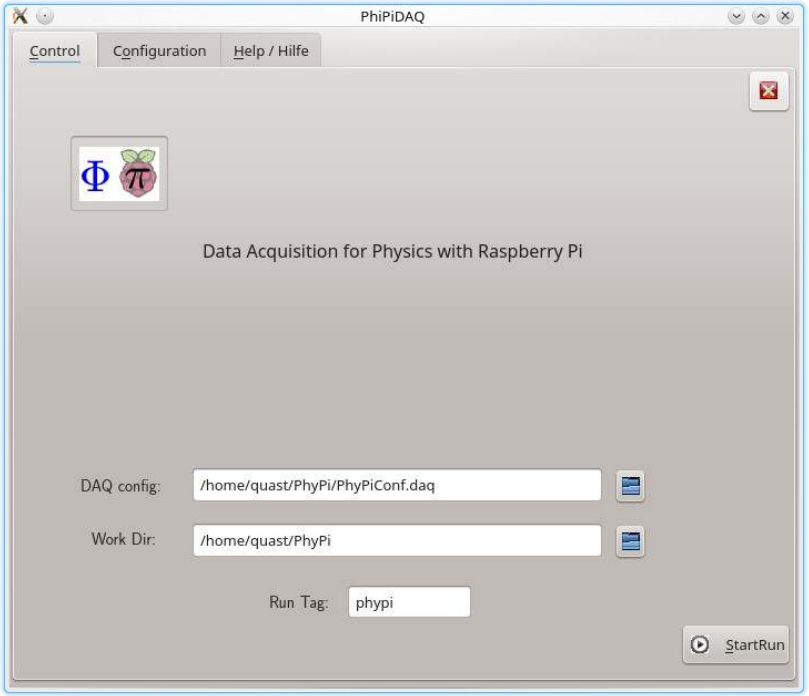


XY-Darstellung: Diodenkennlinie

Ratenmessung an GPIO-Pins



PhyPiDAQ: grafische Oberfläche



Grafische Oberfläche von
PhyPiDAQ

– Laden und Editieren von Konfigurationen

– Start der Datennahme

Wachsende Anzahl an vorbereiteten Konfigurationen:

– Temperaturmessung mit analogen (NTC) und digitalen Sensoren
– digitale Sensoren zur Messung von Strom, Spannung, Beschleunigung, Druck, ...
– Frequenzmessungen auf GPIO-Pins
– Analog-Digitalwandler ADS1115 (4Kanal, 16 bit) und MCP3208 (8Kanal, 12 bit)
– USB-Oszilloskop als Datenlogger und zur Registrierung von Einzelsignalen
– Kraftmessung mit Wägezelle und Dehnungsmessstreifen

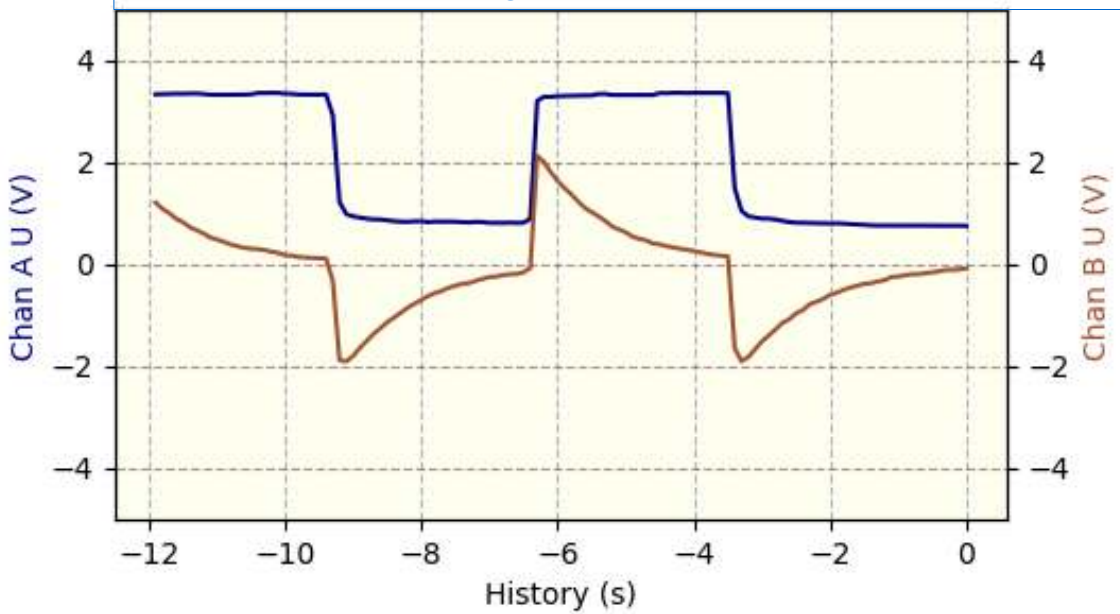
Schülerprojekte: Charakterisierung von GPIO-Pins, Hell-Dunkel-Schaltung, Kalibration eines Temperatursensors

python-Schnittstelle: Beispiel

Software-Paket unter openSource-Lizenz verfügbar:

– `PhyPiDAQ` s. https://github.com/GuenterQuast/PhyPiDAQ

Beispiel:
mit PicoScope aufgezeichnete Lade- und Entladekurven eines Kondensators



Eingangssignal (blaue Kurve) erzeugt durch Abdecken eines lichtempfind-lichen Widerstands unter einer Lampe

Der Plan für heute

Der Plan für heute

Vortrag: Einleitung, Raspberry Pi (Günter Quast)

Hands-On: Kurs_digitale_Messwerterfassung

(Günter Quast, Andres Seith, Christoph Heidecker)

Das Material für den Kurs findet sich hier: https://github.com/GuenterQuast/PhyPiDAQ/doc/
dann → Kurs_digitale_Messwerterfassung_mit_PhyPiDAQ.pdf

Vortrag: Messen mit dem Raspberry PI und PhyPiDAQ (Günter Quast)

Stationen mit verschiedenen Aufbauten:

- Kraftsensor (A.S.)

- Lade- und Entlade-Kurven am Kondensator mit ADS1115 (A.S)

- digitale Sensoren am I²C-Bus des Raspberry PI (G.Q.):
  Stromsensor INA219
  Beschleunigungssensor am I2C-Bus
  Mehrkanal-Messung mit ADS1115 (Diodenkennlinien)
- PicoScope mit Geophon oder Mikrophon (G.Q.)

- Detektoren des Netzwerks Teilchenwelt (C.H.) https://www.teilchenwelt.de
  Szintillator-Panels
  Kamiokanne

Software und Dokumentation siehe https:/github.com/GuenterQuast/picoCosmo

Messen mit dem Raspberry Pi

Der Raspberry Pi

Einplatinencomputer der Raspberry Pi Foundation
basiert auf Arm-Microprozessor (Handy-Technik)

Entwickelt mit den Ziel, jungen Menschen den Erwerb von Programmier- und Hardwarekenntnissen zu ermöglichen

Vollständiges Linux-Betriebssystem mit leistungsfähiger (HDMI-fähiger) Grafik, Netzwerkanschluss inc. WLAN, USB-Port(s) und und speziellen „GPIO“ (**G**eneral **P**urpose **I**nput/**O**utput) Anschlüssen zum Anbinden von digitalen Geräten

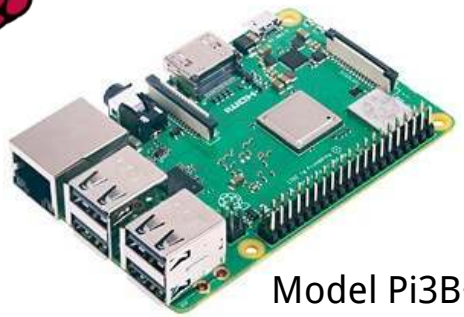
Standardisierte digitale Bussysteme werden unterstützt

Erweiterbar mit Analog-Digital-Wandler(n)

Großes Repository an openSource-Programmen verfügar,
incl. Libreoffice, Mathematica, grafische Programmiersprache Scratch

Große Entwicklergemeinschaft, die Hard- und Sofwarekomponenten beiträgt.

Speichermedium ist eine kostengünstige SD-Karte; Verteilung von Software einfach durch Tauschen der SD-Karte



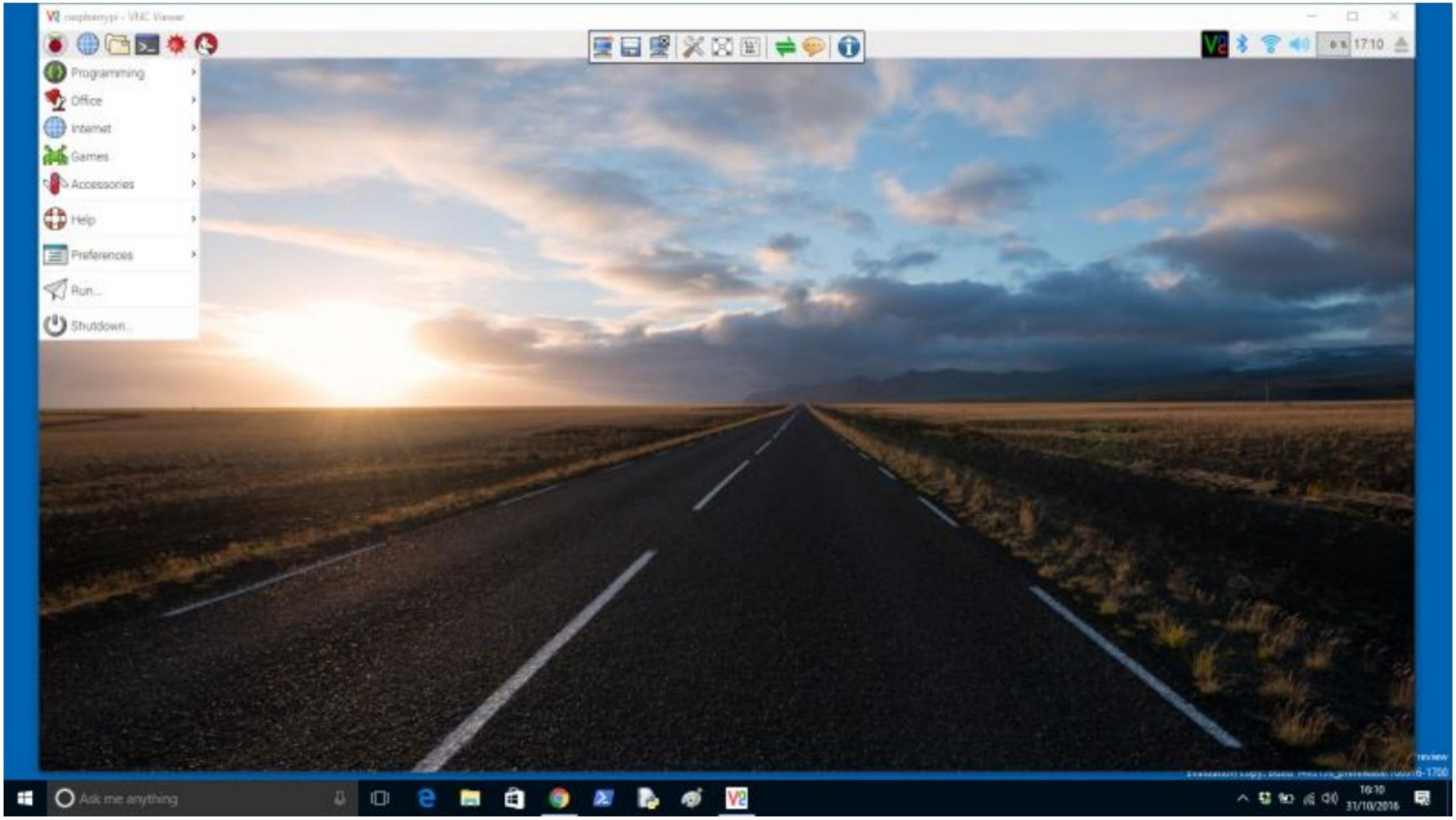
Model Pi3B+

Model Pi3A+

Model PiZero

Raspberry Pi – grafische Oberfläche

Die (konfigurierbare) grafische Oberfläche des Raspberry Pi

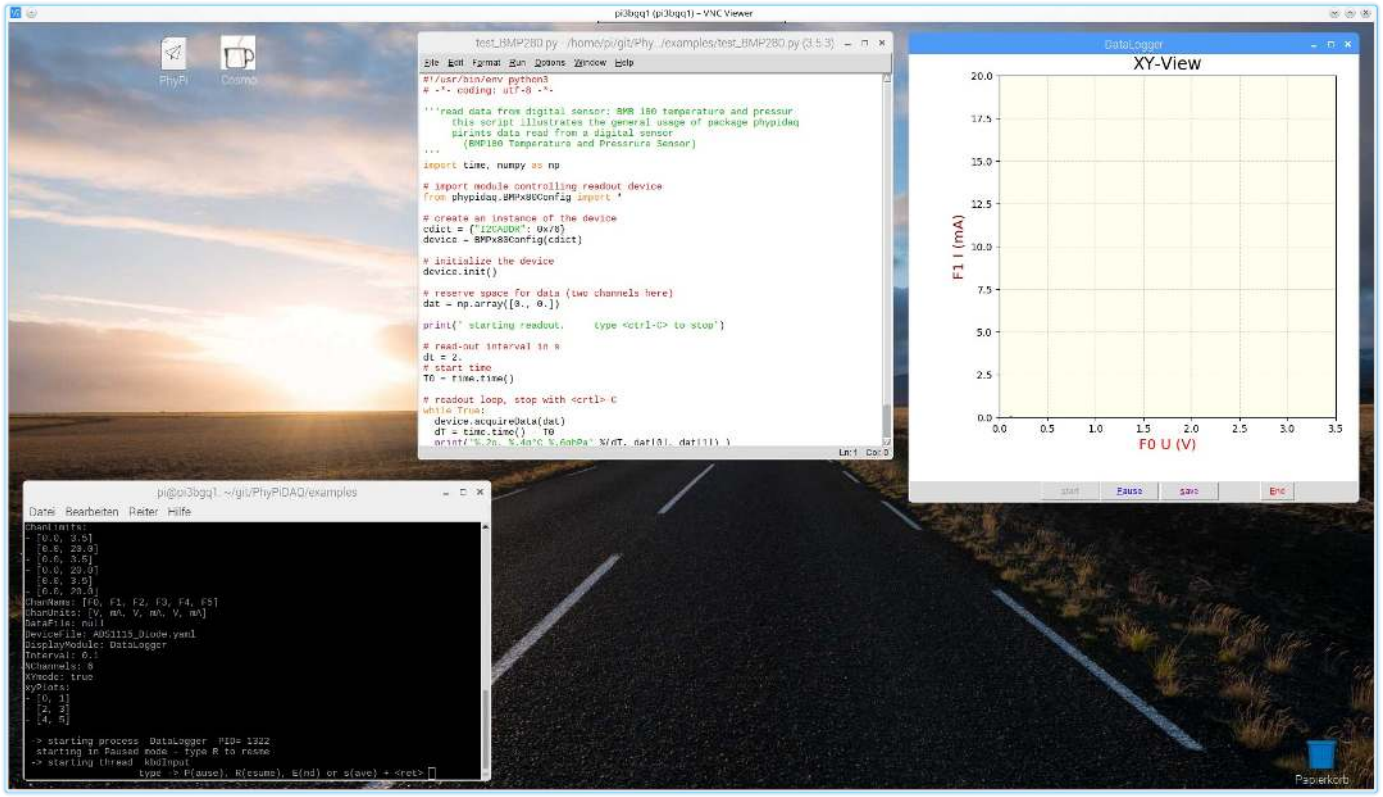


Remote-Zugriff von anderem Computer möglich mit VNC-Viewer

Raspberry Pi – Arbeiten auf der Textkonsole

Das Heranführen an das textuelle Programmieren ist eine Grundvoraussetzung zum aktiven Erarbeiten von digitalen Kompetenzen:

Frag nicht was einen App für dich tun kann (durch Suchen im Menu), sondern sag dem Rechner, was er tun soll !!!



Raspberry Pi Oberfläche mit Konsolen-Fenster, python Entwicklungsumgebung idle3 und grafischer Anzeige

Arbeiten auf der Texkonsole

Die wichtigsten Befehle für die Kommandozeile

Befehl	Beschreibung
`ls`	listet alle Dateien im aktuellen Verzeichnis auf
`cd` **Ordnername**	wechselt in den angegebenen Ordner
`cd`	wechselt ins Home-Verzeichnis
`pwd`	zeigt aktuelles (Arbeits-)Verzeichnis an
`cp` **Datei1 Datei2**	kopiert **Datei1** in (neue) **Datei2**
`mv` **Dateiname1 Dateiname2**	Datei mit **Dateiname1** in **Dateiname2** umbenennen
`mkdir` **Ordnername**	erzeugt einen Ordner mit namen **Ordnername**
`rmdir` **Ordnername**	Ordner **Ordnername** entfernen (muss leer sein)
`sudo nano` **Dateiname**	erstellt/öffent neue Datei **Dateiname**
`rm` **Dateiname**	löscht die angegebene Datei
`pyhton3` **Dateiname.py**	führt Datei **Dateiname.py** in *python3* aus
`./` **Dateiname.py**	alternativ: führt Datei **Dateiname.py** aus
`cat` **Dateiname**	zeigt Inhalt der Datei mit Namen **Dateiname** an
`less` **Dateiname**	zeigt Inhalt einer Datei an (auf und ab mit Pfeiltasten, Ende mit *q*)
`man` **Befehl**	zeigt Hilfsinformation zum angegebenen **Befehl** an
Pfeiltaste hoch/runter	zeigt zuletzt benutzte Befehle an
Pfeiltaste rechts/links	Befehl editieren
`<Str> + <c>`	Beendet das im Terminal ausgeführte Programm
`<Str> + <z>`	Verschiebt das im Terminal laufende Programm in den Hintergrund
`bg`	führt in den Hingergrund verschobenes Programm weiter aus
`jobs`	zeigt alle im Termnal laufenden Prozese an
`kill %i`	stoppt Prozess mit Nummer *i* (siehe Befehl `jobs`)
Befehl `&`	führt **Befehl** als Hintergrundprozess aus
`date`	zeigt Datum und Uhrzeit an

Die Konsole ist sehr mächtig, benötigt werden aber nur wenige Befehle

Befehl	Beschreibung
	`/` in Dateiangabe trennt Unterordner von Ordner- oder Dateinamen
	`~/` in Dateiangabe steht für das Home-Verzeichnis
	`*` in Dateiname steht für beliebige Zeichenfolge
	`?` in Datei- oder Verzeichnisname steht für ein beliebiges Zeichen
	`./` in Pfadangabe zu Datei steht für das aktuelle Verzeichnis
Befehl `>` **Dateiname**	Ausgabe von **Befehl** in Datei mit Namen **Dateiname** speichern
Befehl1 `|` **Befehl2**	Ausgabe von **Befehl1** als Eingabe an **Befehl2**
`sudo` **Befehl**	führt **Befehl** als Administrator aus (mit "**s**uper**u**ser-Rechten")
`sudo reboot`	System neu starten
`sudo halt`	System anhalten (kann danach ausgeschaltet werden)

https://github.com/PhyPiDAQ/doc/TerminalBefehle.pdf

Die Programmiersprache Python

python ist eine einfach zu erlernende, interpretierte Sprache

Ziele bei der Entwicklung waren Einfachheit und Übersichtlichkeit
es gibt nur sehr wenige „Schlüsselwörter“

heute gehört `python` zu den beliebtesten Programmiersprachen
- fast alle Programm im openSource-Bereich liefern neben einer grafischen Oberfläche auch eine `pyhton` – Schnittstelle
(python als „glue language“)
- es gibt zahlreiche mächtige Erweiterungen, z. B.
`matplotlib` zur einfachen Erzeugung von Grafiken in Publikationsqualität
`numpy` zur effizienten Beandlung numerischer Probleme
`scipy` mit mächtigen Bibliotheken für das wissenschaftliche Rechnen
auch moderne Bibliotheken zur *künstlichen Intelligenz* werden über eine pyhton-Schnittstelle angesprochen

Im Einführungskurs werden Elemente von `python` schrittweise eingeführt

auch das Software-Paket PhyPiDAQ ist vollständig in python geschrieben

Raspberry Pi: die GPIO-Schnittstelle

Der Raspberry Pi hat 40 “general-purpose“ Input / Output Anschlüsse

- Masse, 3,3V u. 5,5 Volt
- alle anderen Anschlüsse sind als digitale Ein- oder Ausgänge programmierbar;
  der Innenwiderstand liegt jenseits von 1 GΩ; dies ermöglicht eine sehr einfache Ansteuerung über Sensoren in Spannungsteilern
- manche der Eingänge sind für spezielle digitale Schnittstellen reserviert und werden vom Betriebssystem als „devices“ bereitgestellt

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Einrichten des Raspberry Pi

Die Einrichtung des Raspberry Pi für PhyPiDAQ ist hier beschrieben:

https://github.com/GuenterQuast/PhyPiDAQ/blob/master/doc/Einrichten_des_Raspberry_Pi.pdf

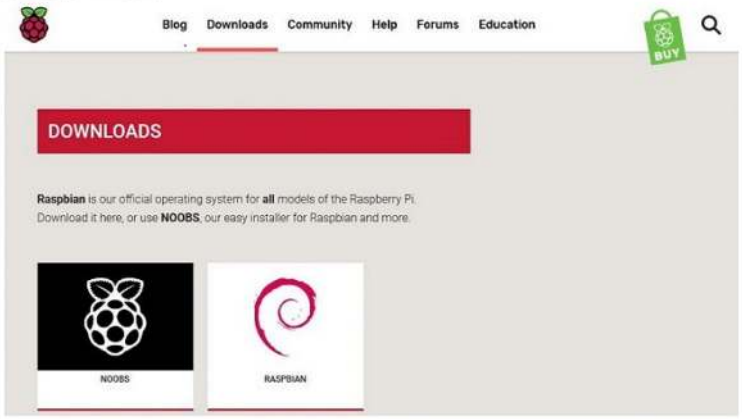
Einrichten des Raspberry Pi

Die nachfolgende Anleitung befasst sich mit den Vorarbeiten, die vor der Erstinbetriebnahme des Raspberry Pi durchgeführt werden müssen. Das Vorbereiten der Micro-SD-Karten sollte beim Einsatz in der Schule möglichst gesammelt einmal im Vorfeld durchgeführt werden, sodass mit den Schülern direkt gestartet werden kann. Die Anleitung ist sehr ausführlich gehalten. Durch Hervorhebungen ist die Anleitung so gestaltet, dass diese auch durch Überfliegen verständlich ist.

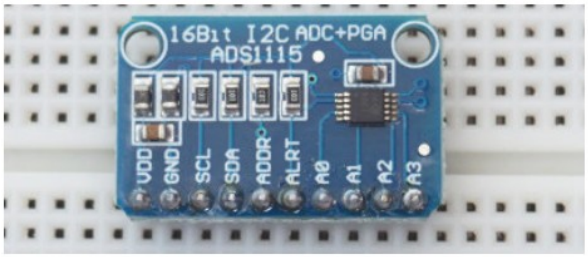
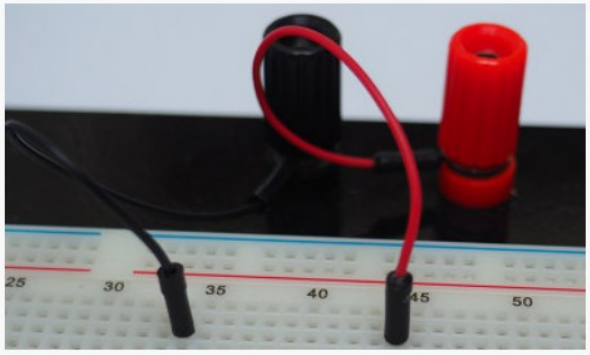
1. Micro-SD-Karte mit Raspian Betriebssystem beschreiben

Für die Inbetriebnahme des Raspberry Pi ist es notwendig, das zugehörige **Betriebssystem Raspian** auf die Micro-SD-Karte zu schreiben. Nachfolgend sind die durchzuführenden Schritte aufgeführt:

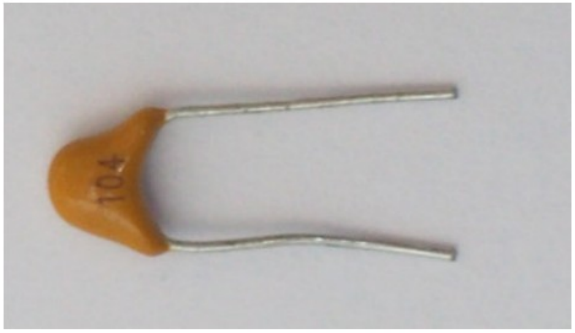
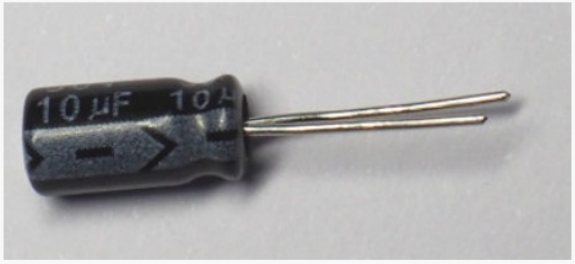
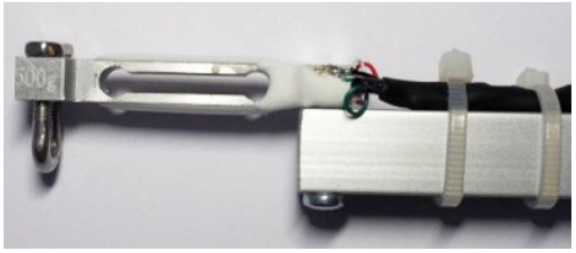
1. Rufen Sie über den Link die offizielle Downloadseite des Betriebssystems Raspian auf und laden Sie dieses herunter:
   - auf **Raspian** klicken.

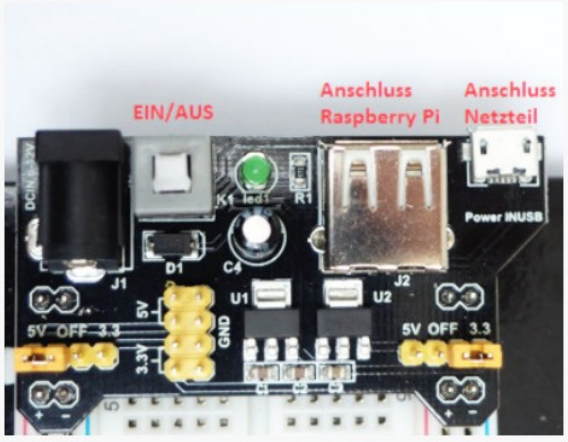
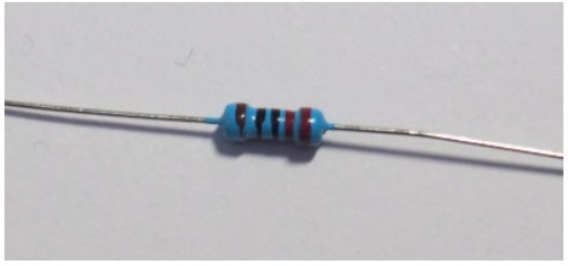


Einführungskurs zur digitalen Messwerterfassung mit dem Raspberry Pi und PhyPiDAQ

Übersicht über wichtige Komponenten	
AD-Wandler ADS1115	
Bananenbuchse mit angeschlossener Breadboardleitung	
Fotowiderstand (LDR Typ5516)	
Instrumentenverstärker AD623	

Einführungskurs (2)

Übersicht über wichtige Komponenten	
Keramikkondensator 0,1 µF	
Elektrolytkondensator 10 µF	
Kraftsensor, Wägezelle TAL220B	
NTC-Widerstand R_{25} = 10 kΩ	

Übersicht über wichtige Komponenten	
Potentiometer 10 kΩ	
Spannungsversorgung	
Widerstand (Bsp. 10 kΩ)	Farbcode: braun, schwarz, schwarz, rot, braun 

Einführungskurs (3)

Beginnen wir an dieser Stelle mit dem praktischen Teil.

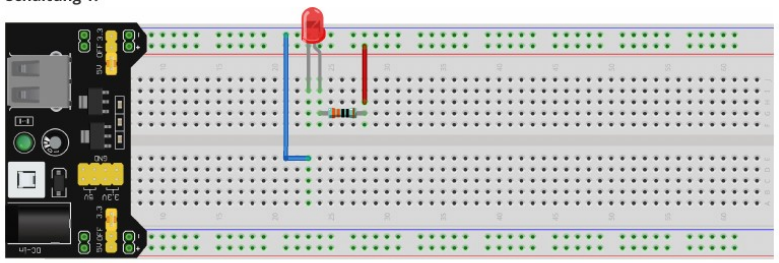
Sie finden den Einführungskurs unter dem Link:

https://github.com/GuenterQuast/PhyPiDAQ/blob/master/doc/Kurs_digitale_Messwerterfassung_mit_PhyPiDAQ.pdf

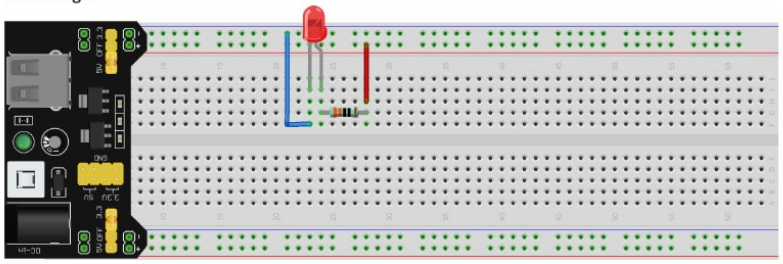
oder in Papierform neben Ihrem Raspberry Pi

Aufgabe 1: In welcher der nachfolgend abgebildeten Schaltungen leuchtet die LED? Begründe.

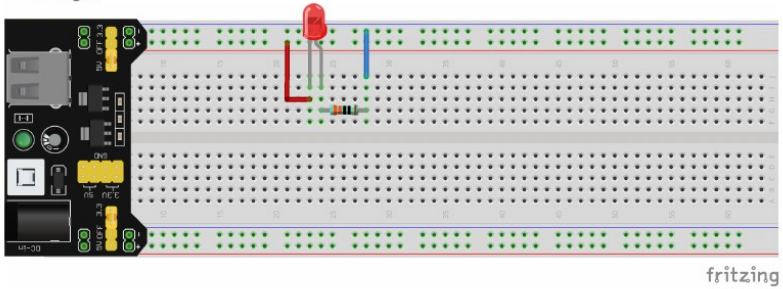
Schaltung 1:



Schaltung 2:



Schaltung 3:



Einführungskurs zur digitalen Messwerterfassung mit dem Raspberry Pi und PhyPiDAQ

Digitale Messtechnik

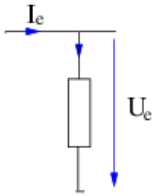
Messgröße

umwandeln

in eine Spannung

mit Hilfe von

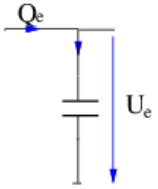
„Sensoren“



$I_e \rightarrow U_e$

Ohmsches Gesetz

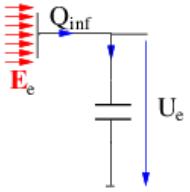
Strommessung



$Q_e \rightarrow U_e$

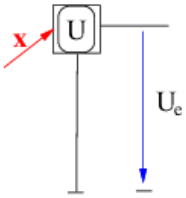
Kondensator-Grundgleichun

Ladungsmessung



$E_e \rightarrow Q_{inf} \rightarrow U_e$

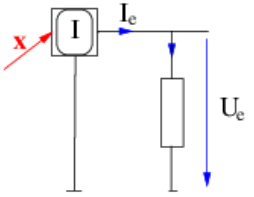
E-Feldmessung über influenzierte Ladung



$x \rightarrow U_e$

viele Sensoren wandeln Meßgröße x in Spannung

z. B. ***T, p, B ...***



$x \rightarrow I_e$

andere wandeln Meßgröße x in Strom

z. B. ***Licht, Radioaktivität***

Beispiele für Moderne Sensoren

NTC-Widerstand

Hall-Sensor

Lichtempfindlicher Widerstand (LDR)

Photodiode

Druckempfindlicher Widerstand

Resistiver Feuchtesensor

Kraft-Sensor mit Dehnungs-messstreifen

Photoröhre zum Nachweis einzelner Photonen

Silizium-Detektor zum Nachweis einzelner Photonen („SiPM“)

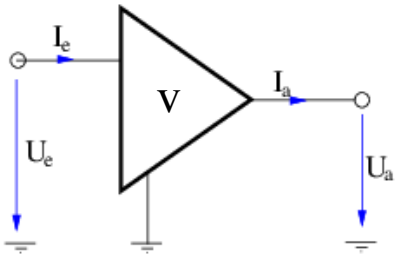
Messverstärker

Bisweilen werden für kleine Spannungen Messverstärker benötigt.

Dazu verwendet man heute Operationsverstärker (OP):

Eingangsseite

$$R_e = \frac{U_e}{I_e}$$
$$\simeq 100\,k\Omega - 10^{13}\,\Omega$$
(je nach Typ)



Ausgangsseite

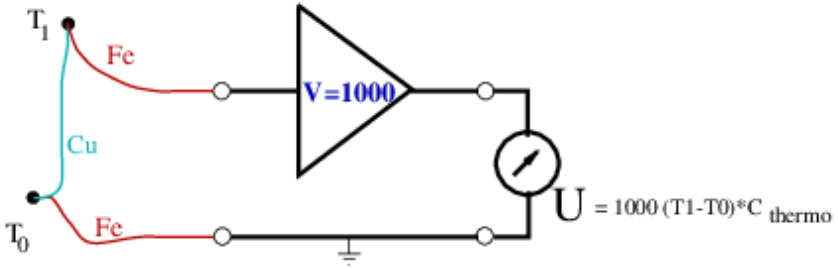
$$r_a = \frac{\partial U_a}{\partial I_a}$$
$\sim$mΩ

Der Verstärkungsfakor ist in weiten Grenzen einstellbar:

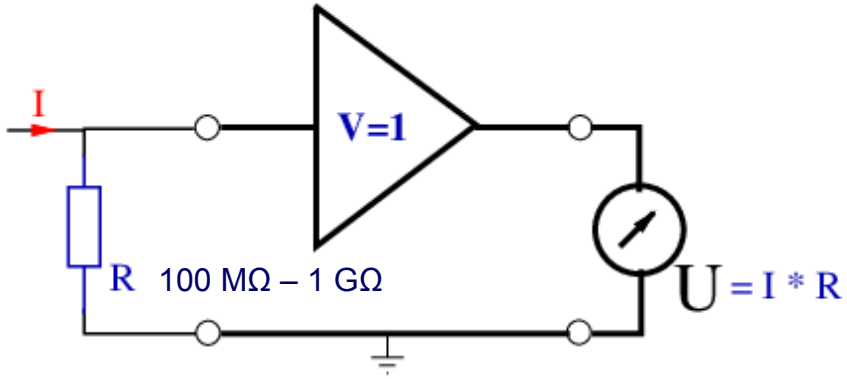
v = 1: reine Leistungsverstärkung

Beispiele: Messen mit OPs

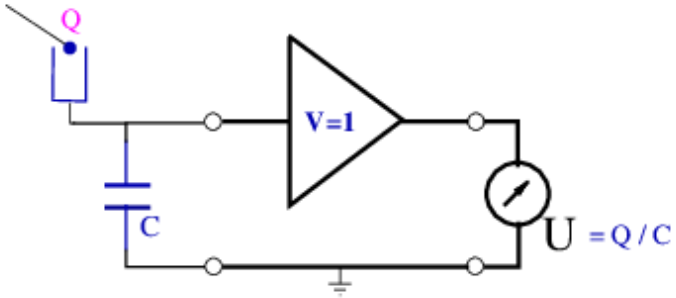
Messung einer Thermo-Spannung (μV-Bereich)

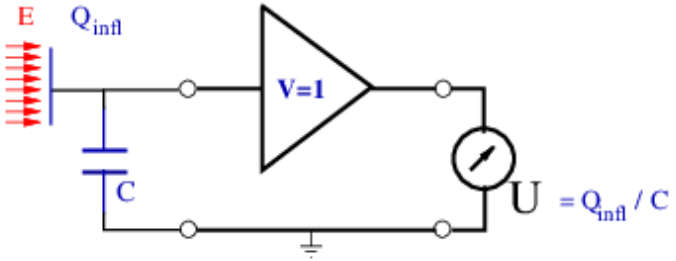


Strommessung im nA-Bereich

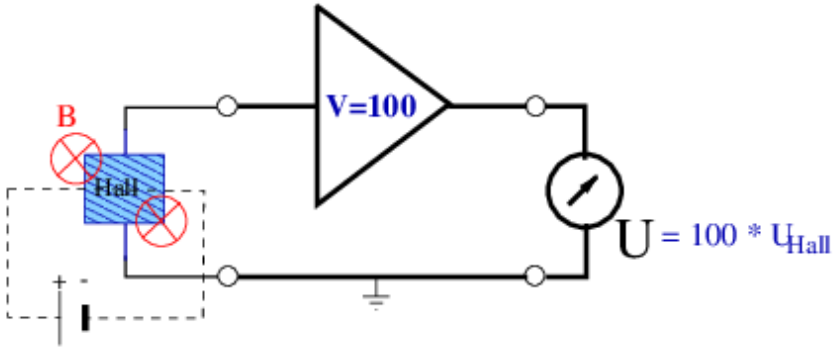


Ladungsmessung mit der Kondensatorgrundgleichung





Magnetfeldmessung mit Hall-Sensor



Digitale Sensoren

Meist arbeiten Sensoren heute „digital“, d.h. die Elektronik ist bereits „on-Chip“, Datenübertragung zu einem Mikrocontroller oder Computer geschieht seriell

Digitaler Temperatursensor (18B20)

Digitaler Druck-Sensor mit I²C-Interface (SM9453)

Digitaler Druck- und Temperatursensor (BMP280)

Beschleunigungssensor mit I²C-Interface auf Breakout-Board (MMA8451)

Strom- u. Spannungs-sensor mit I²C-Interface auf Breakout-Board (INA219)

Analog-Digital-Wandler für den Raspberry Pi

Betriebsspannung 2,0 V – 5,5 V
4 Kanäle oder 2 Kanäle differentiell
16 Bit Auflösung
programmierbarer Vorverstärker x1 – x16
interne Spannungsreferenz
I²C – Bus
bis 860 Hz Ausleserate

Betriebsspannung 2,7 V – 5,5 V
8 Kanäle oder 4 Kanäle pseudo-differentiell
12 Bit Auflösung

Betriebsspannung als Referenz
SPI – Bus
bis 100'000 Hz Ausleserate

Mit ADCs können eine Vielzahl analoger Sensoren angeschlossen
und typische Messaufgaben realisiert werden

Analoge Sensoren

Mit einem ADC kann eine Vielzahl analoger Sensoren angeschlossen werden:

Hallsensor

NTC Widerstand

Stromsensor bis 20 A

Pulssensor

resistiver Drucksensor

lichtempfindlicher Widerstand

(Infrarot) Fotosensor

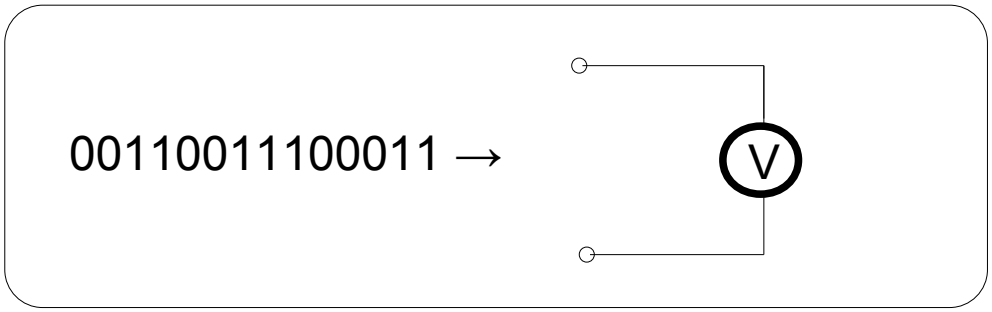
Gassensor

Infrarot Abstandssensor

Regensensor

kapazitiver Sensor für Bodenfeuchte

Digital-Analogwandlung



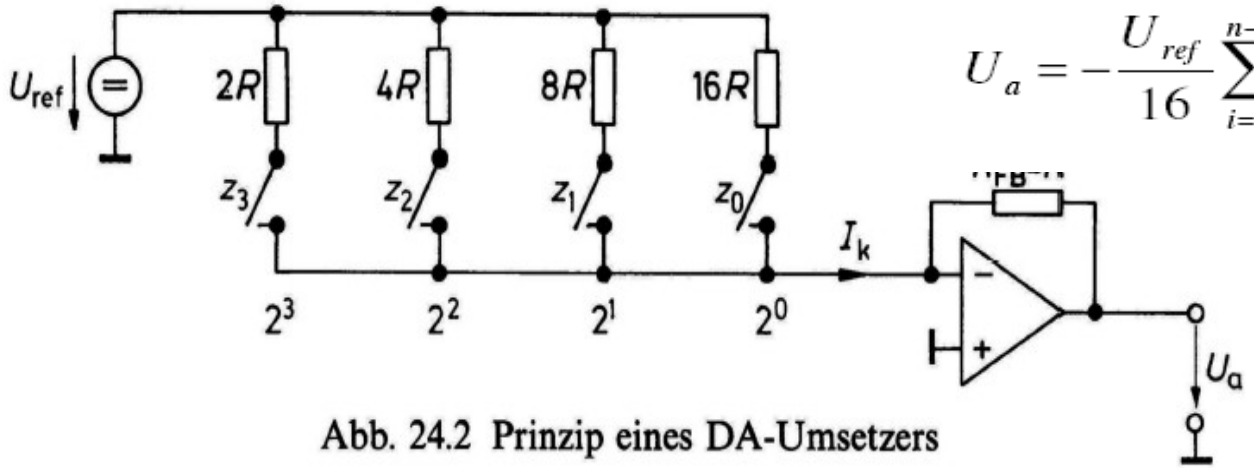
Wird immer dann benötigt, wenn ein Rechner Ausgangssignale zur Steuerung und Regelung liefern soll, z.B. Ansteuerung von

- Lautsprechern
- analogen Video-Geräten
- Steuerspannungen für alle Arten von Reglern
- ...

Digital-Analog-Wandler setzen einen digitalen Wert D in eine entsprechende Spannung um U.

d.h. Abbildung von $0 \leq D < 2^n \longrightarrow U_{min} \leq U < U_{max}$

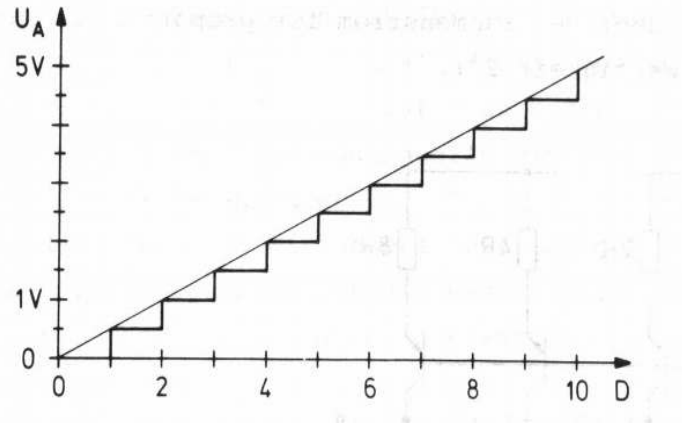
Prinzip Digital-Analog-Wandlung



$$U_a = -\frac{U_{ref}}{16}\sum_{i=0}^{n-1} z_i \cdot 2^i$$

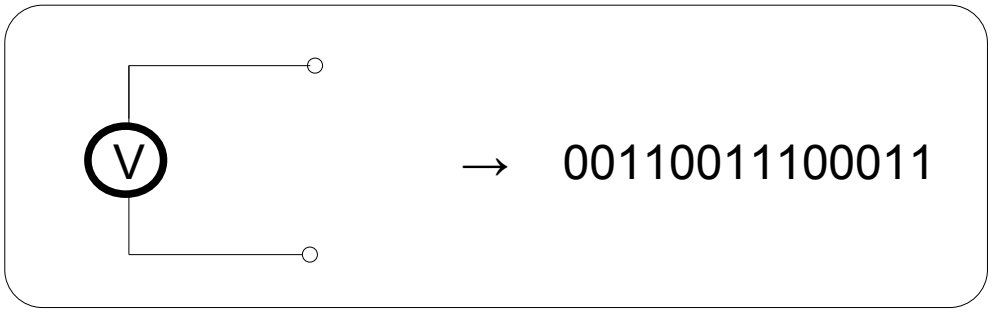
Abb. 24.2 Prinzip eines DA-Umsetzers

Prinzip: den Werten der einzelnen Bits einer Binärzahl entsprechende Ströme werden aufaddiert.



Stufencharakter der Ausgangs-spannung unvermeidlich !

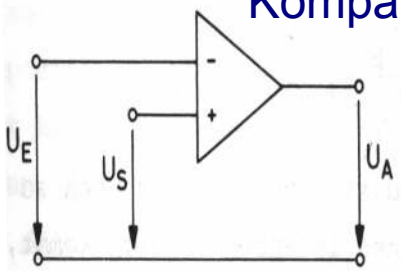
Prinzip Analog-Digitalwandlung



Erfassung von Messdaten durch Rechner erfordert
Wandlung der analogen Werte in digitale Daten

Schlüsselelement:

Komparator



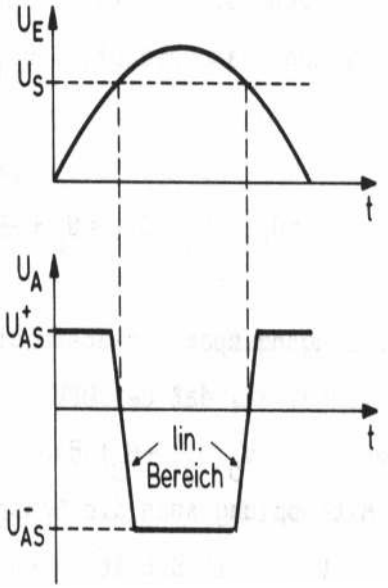
z.B. mit einfachem OP:

$U_E < U_S : U_A = U^+$

$U_E > U_S : U_A = U^-$

d.h. **digitale Ausgabe !**

Sukzessiv oder parallel erfolgender Vergleich einer Spannung U_E mit Referenz-Spannung(en) U_S ergibt Bitmuster, das auf eine Binärzahl codiert wird.



Grundprinzip einer Analog-Digitalwandlung mittels Komparator

Digitale Abtastung („sampling“)

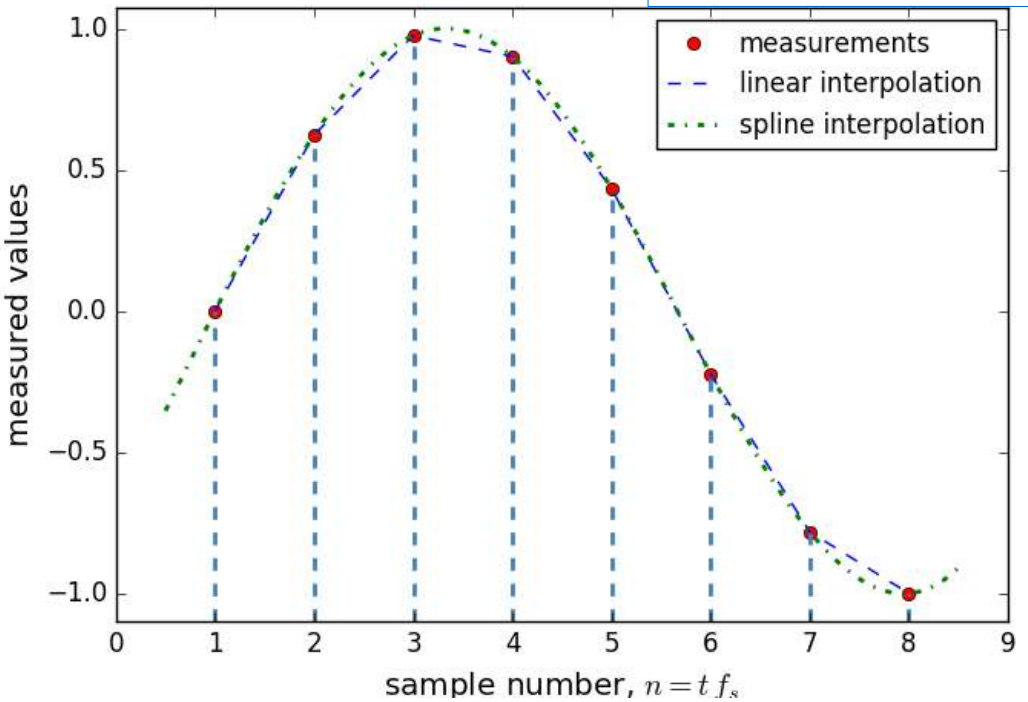
Digitale Messgeräte tasten Signale üblicherweise regelmäßig zu festen Zeiten $t_n = t_0 + n\,T$ mit der Abtast- oder Sampling-Rate

$f_s = 1/T$ ab.

Nummer der Messung

$n = t_n\ f_s$ (für $t_0 = 0$s)

Script `sampling.py`



Datenexport

in Felder mit

- Zeitpunkten t_i (`t[1, …, N]`)
- Messwerten v_i (`v[1, …, N]`)

Grafische Darstellung als Markierungen, evtl.

– verbunden durch gerade Linien „lineare Interpolation“

– verbunden durch Kurven, z.B. „Spline-Interpolation“

(kubische Splines:
stetig differenzierbar aneinander anschließende Polynomstücke 3. Grades)

Prinzip Analog-Digitalwandlung

erzeugter Datenstrom besteht aus Messwerten zu festen Zeiten t_i:

t_1, m_1
...
t_i, m_i
...
t_N, m_N

üblich:
Text-Datei im **csv**-Format
(=**c**omma **s**eparated **v**alues)

Fast alle modernen digitalen Messgeräte enthalten Exportfunktionen für die aufgezeichneten Rohdaten, die über Schnittstellen **(heute fast ausschließlich USB)** an einen PC übertragen werden.

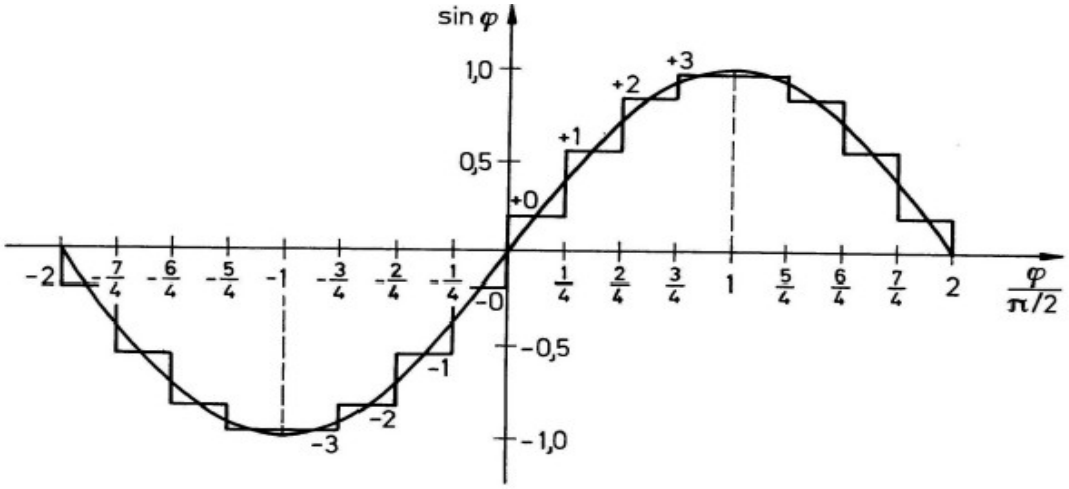


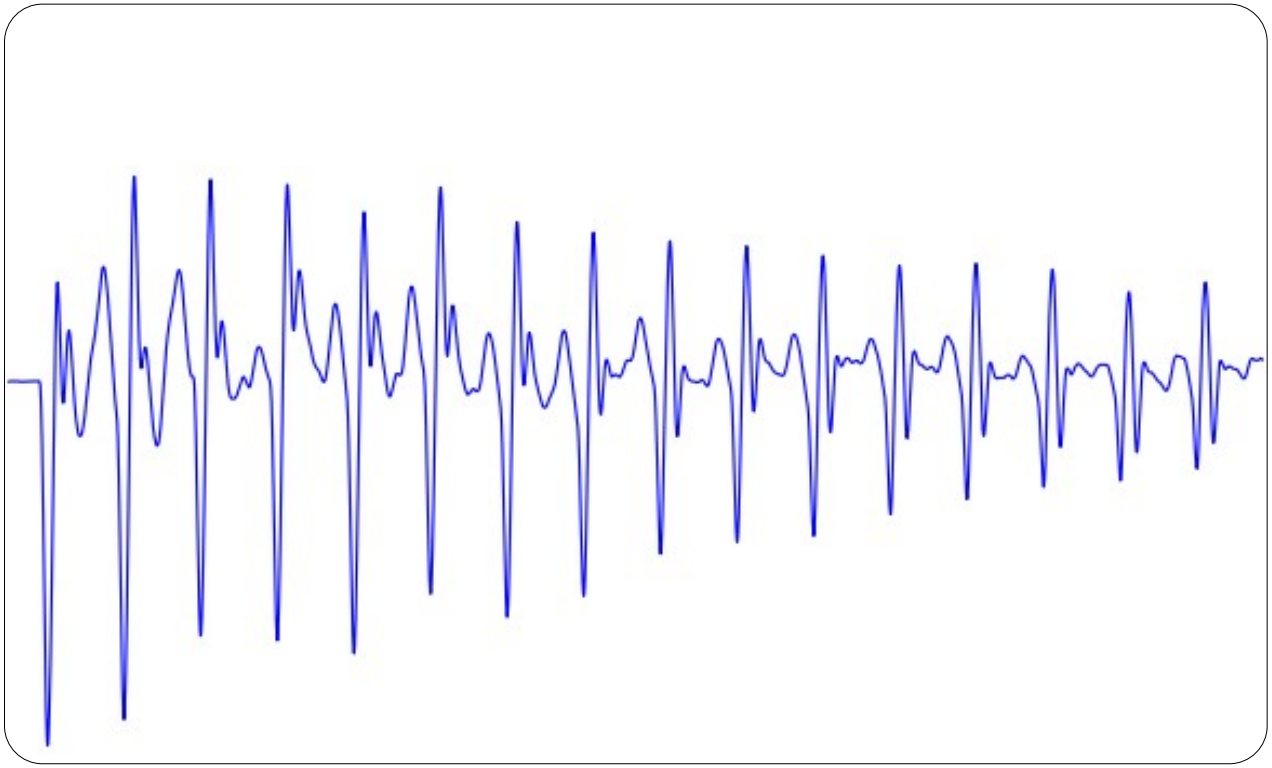
Abb. 24.17 Approximation einer Sinusschwingung mit 16 Stufen

Digitalisiertes Signal entspricht nur näherungsweise dem Original,
„Quantisierungsfehler“ sind unvermeidlich !

Beispiel: Rohdaten einer Wellenform

```
Time,Channel A
(ms),(V)

-0.34927999,-0.00045778
-0.34799999,-0.00045778
-0.34671999,-0.00045778
-0.34543999,-0.00045778
-0.34415999,-0.00045778
-0.34287999,-0.00033570
-0.34159999,-0.00018311
-0.34031999,-0.00018311
-0.33903999,-0.00018311
-0.33775999,-0.00003052
-0.33647999,0.00006104
-0.33519999,0.00006104
-0.33391999,0.00006104
-0.33263999,0.00006104
-0.33135999,0.00021363
-0.33007999,0.00021363

        . . .

9.63983995,0.02642903
9.64111995,0.02655110
9.64239995,0.02655110
9.64367995,0.02655110
9.64495995,0.02655110
9.64623995,0.02655110
9.64751995,0.02655110
9.64879995,0.02642903
9.65007995,0.02612384
9.65135995,0.02584918
9.65263995,0.02557451
9.65391995,0.02526933
9.65519995,0.02487259
```

7816 Wertepaare exportiert aus USB-Oszilloskop **PicoScope** im **csv**-Format (**c**omma **s**eparated **v**alues)



Messen mit dem Raspberry Pi –

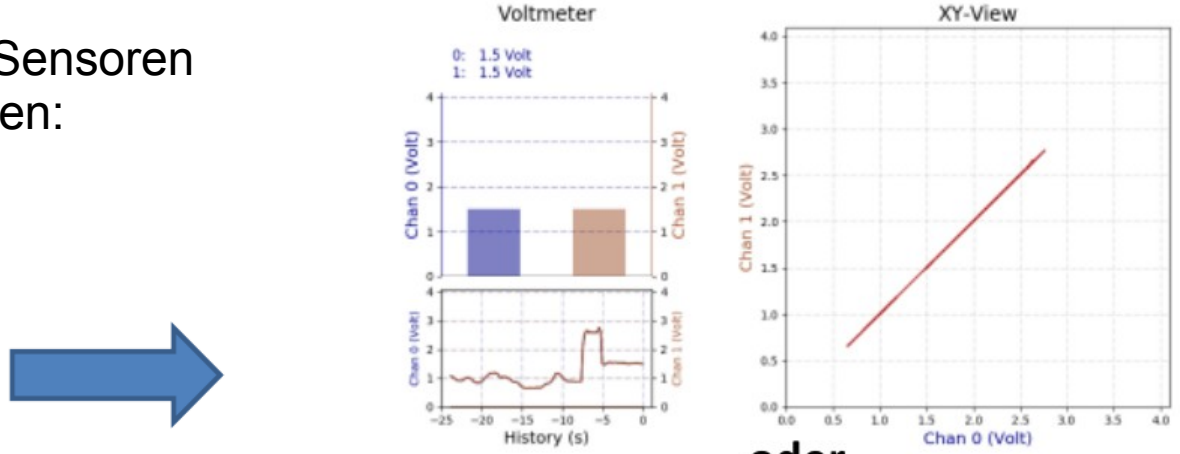
python-Beispiele und das Paket PhyPiDAQ

Digitale Messwerterfassung mit dem Raspberry Pi

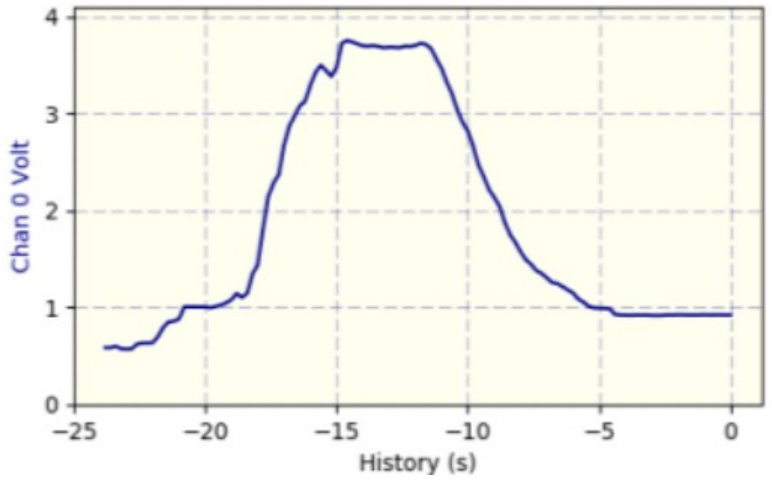
Messdatenanzeige

Auswahl verschiedener Sensoren
für physikalische Größen:

- elektrische Spannung
- elektrischer Strom
- Temperatur
- Druck
- Magnetfeld
- Distanz
- Beschleunigung
- Spannung
- ...



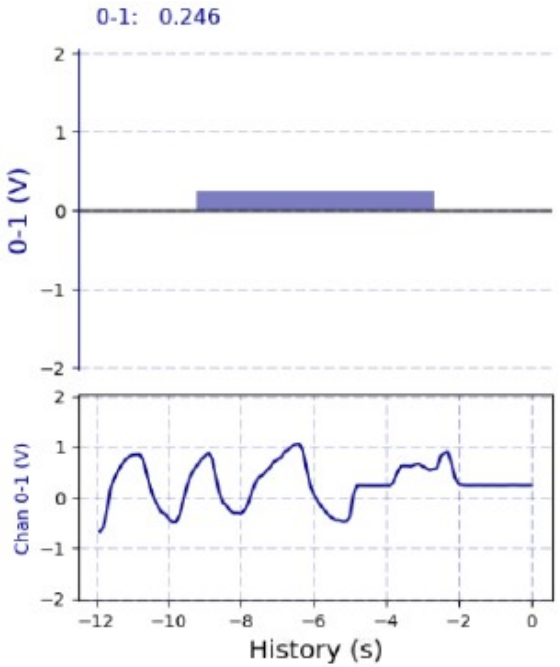
oder



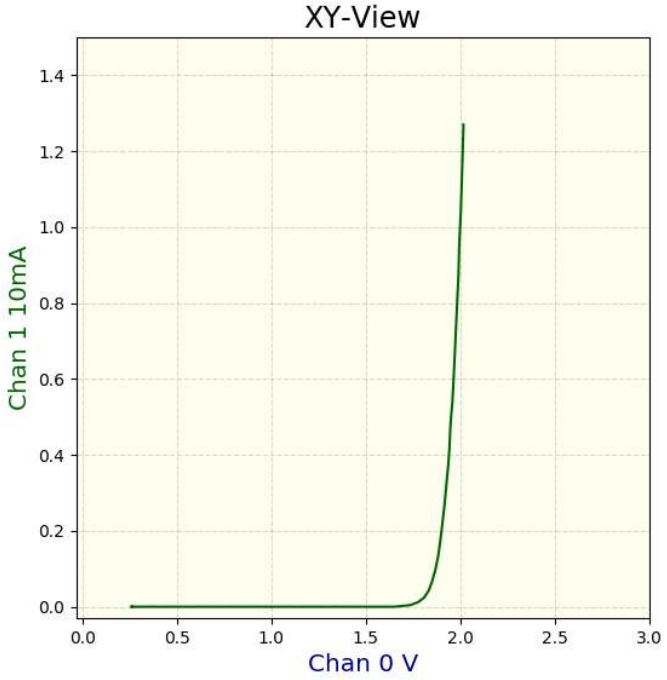
Speicherung zur
späteren Auswertung
(auch mit Excel)

Liste der unterstützen Sensoren sowie verfügbare Anzeigen werden ständig erweitert

PhyPiDAQ: Anzeigen

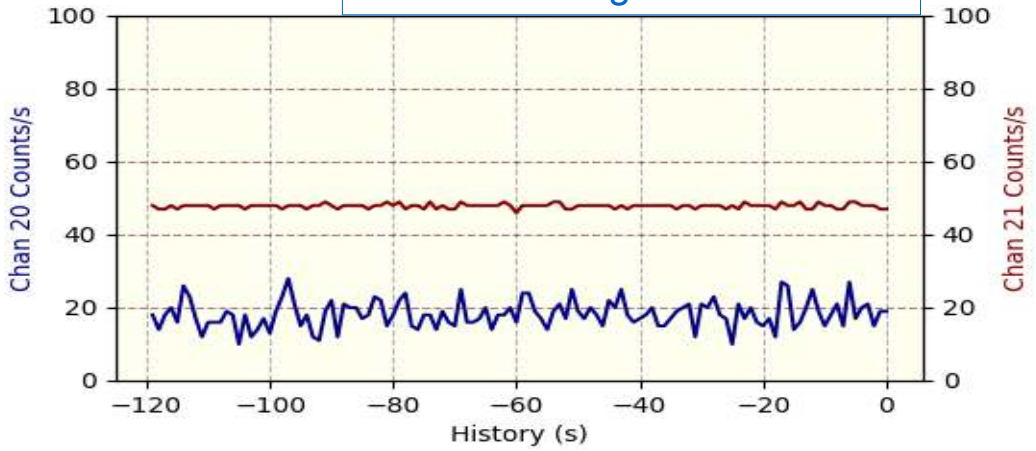


Balkenanzeige und zeitlicher Verlauf eines Signals



XY-Darstellung: Diodenkennlinie

Ratenmessung an GPIO-Pins

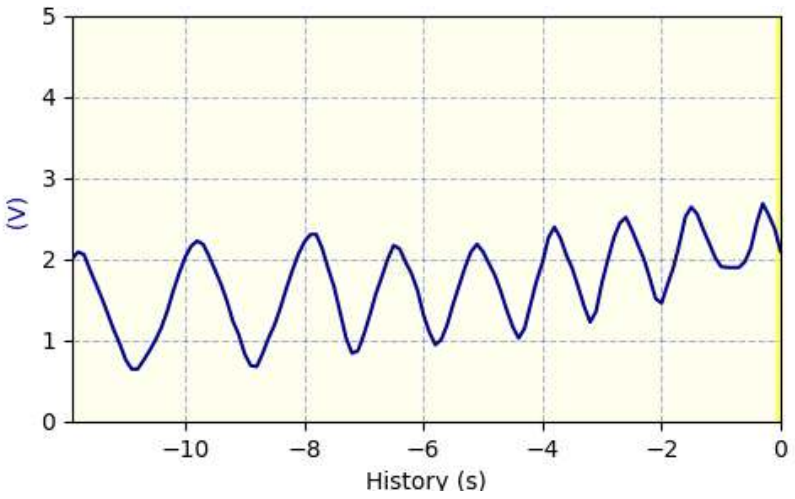


Beispiel: Anzeige eines analogen Werts

Einfaches python-Script:

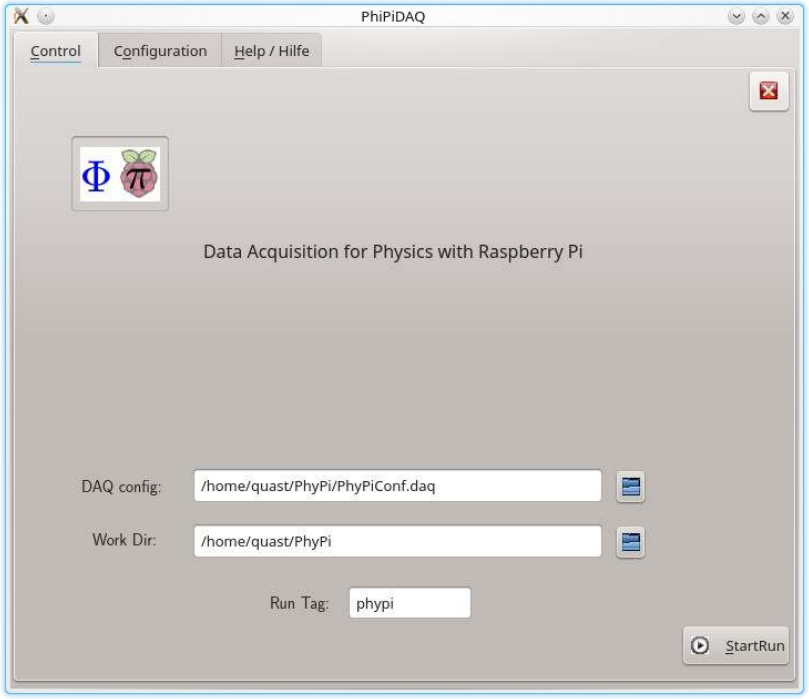
```
#!/usr/bin/env python3
'''display_analog.py
     illustrate general usage of package phypidaq
     prints and displays data read from an analog channel
'''
import time, numpy as np
# import module for readout device and display
from phypidaq.ADS1115Config import *
from phypidaq.Display import *

# create an instance of device and display ...
device = ADS1115Config()
display = Display(interval=0.1)  # 0.1 sec update interval
# ... and initialize
device.init()
display.init()

# reserve space for data (here only one channel)
dat = np.array([0.])

print(' starting readout,     type <ctrl-C> to stop')
# start time
T0 = time.time()
try:
# readout loop, stop with <crtl>-C
  while True:
    device.acquireData(dat)
    dT = time.time() - T0
    print('%.1f, %.4g' %(dT, dat) )
    display.show(dat)
except KeyboardInterrupt:
  print('ctrl-C received - ending')
  device.closeDevice()
  display.close()
```

```
pi@pi3bgq1:~/git/PhyPiDAQ/examples $ ./display_analog.py
 -> starting subprocess  DataLogger  PID= 3055
 starting readout,     type <ctrl-C> to stop
0.0, 2.622
0.0, 2.622
1.1, 2.622
```



Datenaufnahme mit der grafischen Oberfläche

Datenanzeige und Aufzeichnung starten
über die grafische Oberfläche
phypi.py oder mit dem Script run_phypi <config.daq>

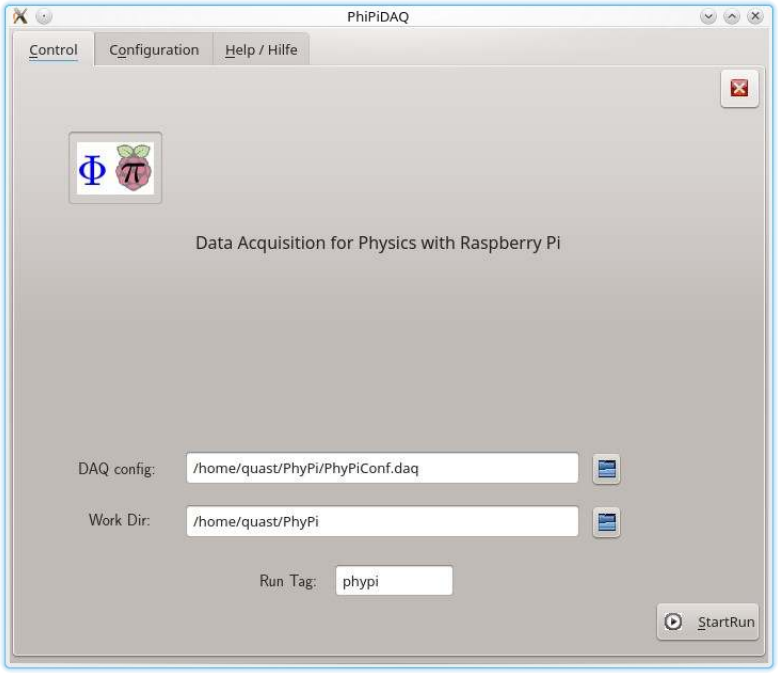


grafische Oberfläche ./phypi.py:
erlaubt das Editieren von Konfigurationen
Abspeichern von Konfigurationen
Start der Datennahme

```
pi@pi3bgq1:~/git/PhyPiDAQ $ ./run_phypi.py PhyPiConf.daq

*==* script ./run_phypi.py running

  Configuration from file PhyPiConf.daq
  configuring device ADS1115Config

PhyPiDAQ Configuration:
ChanColors: [darkblue, sienna]
ChanLabels: [U, U]
ChanLimits:
- [-6.114, 6.114]
- [-6.114, 6.114]
ChanNams: [0-1, 2-3]
ChanUnits: [V, V]
DataFile: null
DeviceFile: config/ADS1115Config.yaml
DisplayModule: DataLogger
Interval: 0.1
NChannels: 2
XYmode: false
startActive: false

 -> starting process  DataLogger  PID= 2298
  starting in Paused mode - type R to resume
 -> starting thread  kbdInput
                    type -> P(ause), R(esume), E(nd) or s(ave) + <ret>
```

Kommandozeile ./run_phypi.py:
ausführen der Datennahme mit den in der Dateil PhyPiConf.daq angegebenen Parametern

PhyPiDAQ: grafische Oberfläche



Grafische Oberfläche von
PhyPiDAQ

– Laden und Editieren von Konfigurationen

– Start der Datennahme

Wachsende Anzahl an vorbereiteten Konfigurationen:

- Temperaturmessung mit analogen (NTC) und digitalen Sensoren
- Frequenzmessungen auf GPIO-Pins
- Analog-Digitalwandler ADS1115 (4Kanal, 16 bit) und MCP3208 (8Kanal, 12 bit)
- PicoScope als Datenlogger oder zur Registrierung kurzer Pulse
- Kraftmessung mit Wägezelle und Dehnungsmessstreifen
- Auslese von digitalen Sensoren am I²C-Bus (Druck, Beschleunigung, Temperatur …)
- Mehr-Kanal-Darstellungen (Kennlinienschreiber)

Schülerprojekte: Hell-Dunkel-Schaltung, Kalibration eines Temperatur- oder Kraftsensors, einfaches Programmier-Interface für Projekte

Konfigurationsdateien

Haupt-Konfiguration *.daq

Festlegung von
- Gerät für Datenaufnahme
- Anzeige-Modul
- Art der gewünschten Grafik
- Angaben zu Messgrößen
  (Name, Einheit, Grenzen)
- ggf. Umrechnung zur Kalibration
- ggf. Anwendung von Formeln
- ggf. Datei zum Abspeichern

```
# Konfigurations-Optionen fuer PhyPiDAQ

# Konfigurationsdateien fuer Geraete
DeviceFile: config/ADS1115Config.yaml
#DeviceFile: config/MCP3008Config.yaml
#DeviceFile: config/PSConfig.yaml
#DeviceFile: config/MAX31865Config.yaml
#DeviceFile: config/GPIOCount.yaml
#DeviceFile: config/DS18B20Config.yaml
#DeviceFile: config/MAX31855Config.yaml
## ein Beispiel fuer mehrere Geraete
#DeviceFile: [config/ADS1115Config.yaml, config/ GPIOCount.yaml]

Interval: 0.1               # Datennahme-Intervall in Sekunden

DisplayModule: DataLogger   # zeitlicher Verlauf der Messgroessen
```

```
# DisplayModule: DataGraphs   # text, Balkendiagramm, zeitlicher Verlauf und xy-
Darstellung
Chan2Axes: [0,1]            # Kanal auf linker(0) oder rechter(1) Achse
                            # Voreinstellung [0,1,1,...]

XYmode:     false           # XY-Darstellung ein/aus
#xyPlots:   # Paare von Kanaelen als xy-Grafik
# - [0,1]     # x: Kanal 0, y: Kanal 1
# - [0,2]     # x: Kanal 0, y: Kanal 2 (falls vorhanden)
     # Voreinstellung [0,1], [0,2], ..., [0, n-1] bei n aktiven Kanaelen

#  Meta-Daten fuer jeden Kanal
ChanLabels: [U, U]            # Namen
ChanUnits: [V, V]             # Einheiten
ChanColors: [darkblue, sienna] # Farbzuordnung in der Anzeige

# ggf. werden hier die Informationen aus der Geraete-Konfiguration ueberschrieben
##ChanLimits:
## - [0., 1.]   # chan 0
## - [0., 1.]   # chan 1
## - [0., 1.]   # chan 2

# ggf. Kalibration der Rohmessungen
#ChanCalib:
#  - null    oder  - <Faktor> or  - [ [ <wahre Werte> ], [ <Rohwerte> ] ]
#  - 1.                        # chan0: ein einfacher Faktor fuer Kanal 0
#  - [ [0.,1.], [0., 1.] ]     # chan1: Interpolation [wahr]([roh] )
#  - null                      # chan2: Keine Kalibration

# Formel auf Werte anwenden
#ChanFormula:
#  - c0 + c1     # chan0 = Summe von Kanal 0 und 1
#  - c1          # chan1 : = Kanal 1 (Keine Aenderung)
#  - null        # chan2 : Keine Formel

# Name der Ausgabedatei im CSV-Format
#DataFile:   testfile.csv     # Dateiname
DataFile:   null             #   null falls keine Ausgebe gewuenscht
#CSVseparator: ';'           # Feld-Trenner auf ';' setzen, Vorgabe ist ','
```

Gerätekonfigurationen

Konfiguration für ADC ADS1115


```
# Beispiel einer Konfiguration fuer den Analog-Digital-Wandler ADS1115

DAQModule: ADS1115Config    # relevantes phypidaq-Modul

ADCChannels: [0, 3]         # aktive ADC-Kanaele
                          # moegliche Werte: 0, 1, 2, 3
                          #  in differentiellem Modus:
                              #    -  0 = ADCChannel 0
                              #           minus ADCChannel 1
                              #    -  1 = ADCChannel 0
                              #           minus ADCChannel 3
                              #    -  2 = ADCChannel 1
                              #           minus ADCChannel 3
                              #    -  3 = ADCChannel 2
                              #           minus ADCChannel 3

DifModeChan: [true, true] # differentiellen Modus einschalten

Gain: [2/3, 2/3]          # programmierbarer Verstaerkungsfaktor
                          #   moegliche Werte:
                          #     - 2/3 = +/-6.144V
                          #     -   1 = +/-4.096V
                          #     -   2 = +/-2.048V
                          #     -   4 = +/-1.024V
                          #     -   8 = +/-0.512V
                          #     -  16 = +/-0.256V
sampleRate: 860           # programmierbare Datenrate des ADS1115
                          #    moegliche Werte:
                          #    8, 16, 32, 64, 128, 250, 475, 860
```

Konfiguration für USB-Oszilloskop als Datenlogger


```
# Konfiguration für PicoScope als Datenlogger

DAQModule: PSConfig  # relevantes phypidaq-Modul

PSmodel: 2000a       # PicoScope Modell (PS2000a ist die Vorgabe)

# Konfiguration der Kanäle
picoChannels: [A, B]   #  Kanal A und B
ChanRanges: [2., 2.]   # Wertebereich
ChanOffsets: [-1.95, -1.95] # analoger Offset, wir vor Anzeige addiert
ChanModes: [DC, DC]    # Kanal-Kopplung (DC oder AC)
sampleTime: 2.0E-02    # Dauer der Datenaufnahme
Nsamples: 100          # Zahl der Messungen

# trigger
trgActive: false  # Aufnahme ohne Oszilloskop-Trigger
trgChan: A

# Interner Signalgenerator
frqSG: 0.    # aus
```

– Abspeichern des (quadratischen) Mittelwerts einer Sequenz von Messwerten in einem kurzen (20ms) Zeitfenster
– auch Registrierung von Oszillogrammen möglich

USB-Oszilloskop

Moderne, kostengünstige **USB-Oszilloskope**

- haben eine genügend hohe Bandbreite, um auch sehr kurze Pulse zu registreiren, z.B. von Einzelphoton-Detektoren, Geiger-Rohren oder Photoröhren
- die Trigger-Konfiguration ist flexibel genug, um zu zufälligen Zeitpunkten entstehenden Signale abzuspeichern
- bieten eine Programmierschnittstelle zur Steuerung und Datenauslese

Unsere Wahl fiel auf die Einsteiger-Varianten der Firma pico Technology :

PicoScope mit

– 10 – 100 MHz Bandbreite
– hohen Abtastraten (0.1 – 1 GHz)
– 2 oder 4 Kanälen
– gut dokumentierter API
(Application Programming Interface),
lauffähig auf Raspberry Pi
– **`python`**-Schnittstelle

die volle grafische Oberfläche läuft [leider noch] nicht auf dem Raspberry...

werden am KIT auch vielfältig in Praktika und Laboren eingesetzt

Die python-Schnittstelle

Das Paket `picoDAQ` (https://github.com/GuenterQuast/picoDAQ)
kapselt das Programmier-Interface, um einfache

- Konfiguration u.
- Auslese zu ermöglichen:

Einfaches Code-Beispiel

```
import numpy as np

from picodaqa.picoConfig import PSconfig

PS = PSconfig(confdict)
PS.init()

buffer = np.zeros(PS.NChannels, PS.NSamples)

while true:
  PS.acquireData(buffer)

  # … Code zur Verarbeitung der Daten in buffer
```

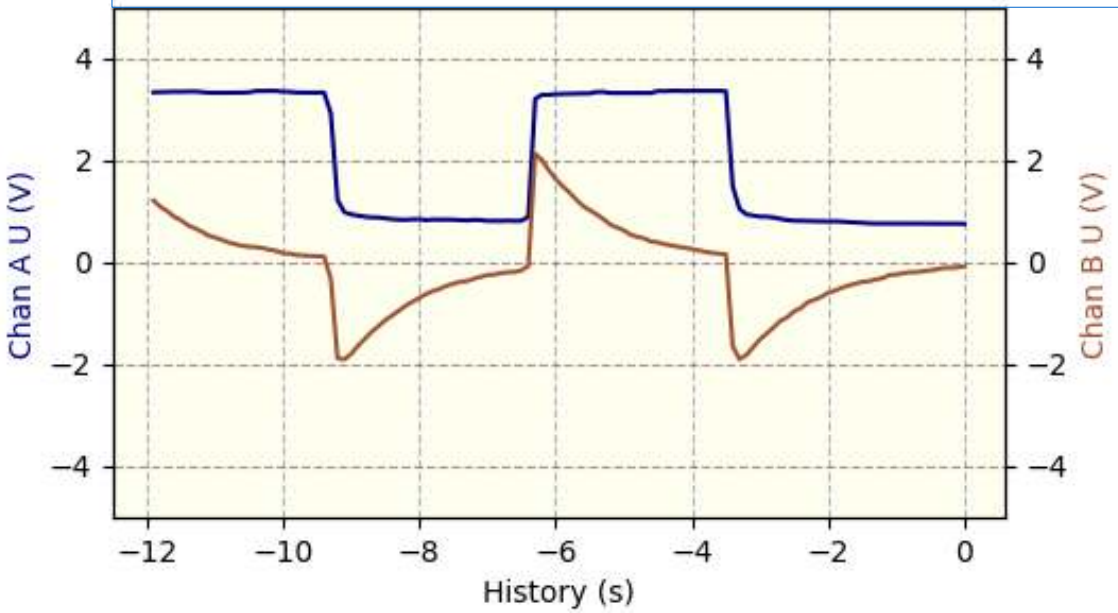
Daten zur Konfiguration

```
# Konfiguration PicoScope 2000B

PSmodel: 2000a  # Treiber-Typ

picoChannels:    [A]
## picoChannels:   [A, B]
ChanRanges:      [0.5, 0.2]
ChanOffsets:     [0.4, 0.45]

sampleTime: 16.E-6
Nsamples:   3500

trgChan:    A
trgThr:     -45.E-3
trgTyp:     Falling
pretrig:    0.05
```

`picodaqq` nutzt das Paket `pico-python` sowie das PicoTech SDK

Beispiel: PicoScope

Beispiel:
mit PicoScope aufgezeichnete Lade- und Entladekurven eines Kondensators



Eingangssignal (blaue Kurve) erzeugt durch Abdecken eines lichtempfind-lichen Widerstands unter einer Lampe

Zufällige Ereignisse

Wenn die **Wahrscheinlichkeit** des Eintreffens eines Ereignisses für jedes Zeitintervall **konstant** ist $\longrightarrow$

- die Anzahl k der in Zeitintervallen ΔT registrierten Ereignisse folgt einer **Poisson-Verteilung**

$$P(k;\mu) = \frac{\mu^k}{k!}\mathrm{e}^{-\mu}$$

Erwartungswert $\mathrm{E}[k] = \mu$ Varianz $\mathrm{V}[k] = \mu$

- Die Zeit t_w zwischen zwei mit der mittleren Rate R aufeinanderfolgenden Ereignissen folgt einer Exponentialverteilung

$$f(t_w) = R \cdot \exp(-t_w \cdot R)$$

Häufig liegen also die Ereignis-Zeiten dicht beieinander – mit langen Pausen, ein Rechner bearbeitet die Daten aber mit konstanter Rechengeschwindigkeit

$\longrightarrow$ benötigen **besondere Art der Datenaufzeichung:**

– schnelles Sammeln der Daten in einem Puffer-Speicher

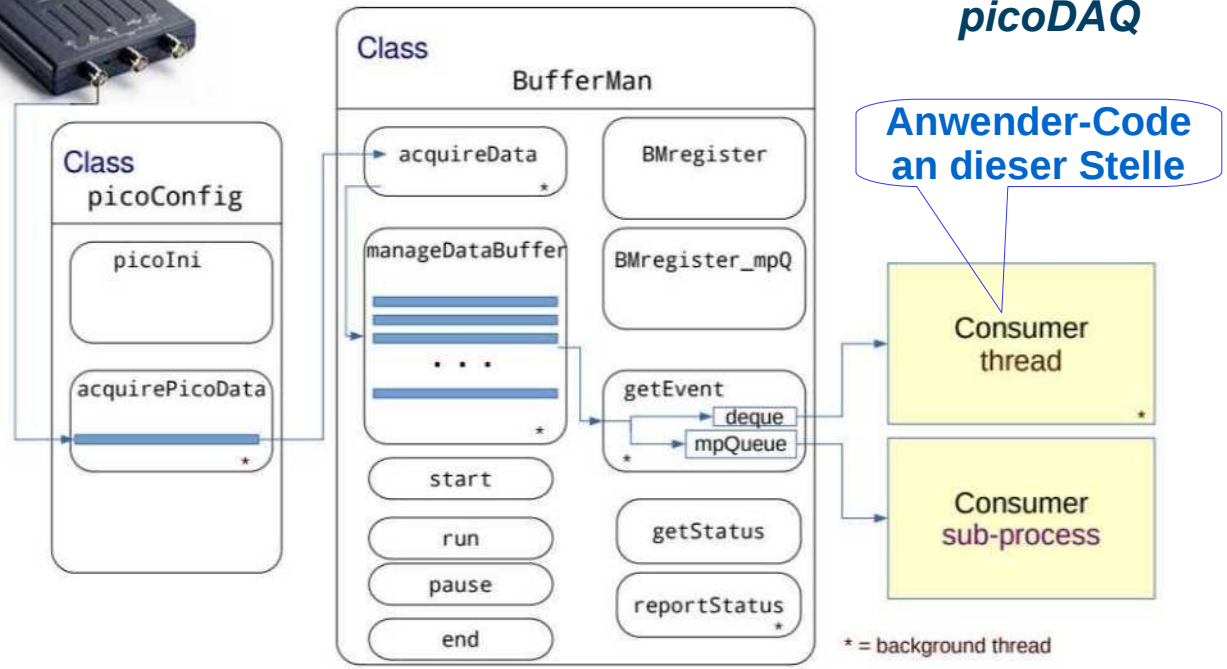
– Verteilung der Daten an langsame, parallel ablaufende Prozesse

Auslese von Myon-Detektoren mit USB-Oszilloskop

Modernes Konzept zur Datennahme: Trigger und frühe Digitalisierung

- **Buffermanager** zur Datenaufnahme der zufälligen Trigger-Ereignisse und Verteilung an andere Prozesse zur Überwachung, Echtzeit-Darstellung, Analyse oder Datenspeicherung
- Validierung des Triggerpulses durch Korrelationsfilter mit Referenzpuls
- Suche nach weiteren Pulsen mit der gleichen Methode
  - in anderen Kanälen (Koinzidenz)
  - oder zu späteren Zeitpunkten (Doppelpuls-Suche)

Struktur des `python`*-Codes* ***picoDAQ***



Vorteile:

- „nur“ Rechner als „black Box“
- Oszilloskop anschaulich und bekannt
- einfache Funktions-erweiterungen durch Software-Updates

https://github.com/GuenterQuast/picoDAQ

Kosmische Strahlung und der Raspberry Pi

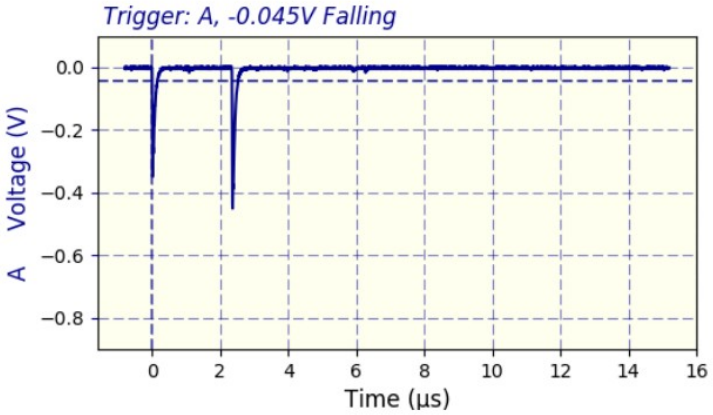
Messung Kosmischer Myonen

Kaffeekanne mit Wasser als

Wasser-Cherenkov-Zähler

zum Nachweis geladener Teilchen

aus der kosmischen Strahlung



Mit PicoScope und SW picoCosmo aufgezeichnete Pulse aus der „Kamiokanne“

Aufzeichnung mit PicoScope,

Steuerung, Datenvisualisierung und Datenauswertung mit Raspberry Pi

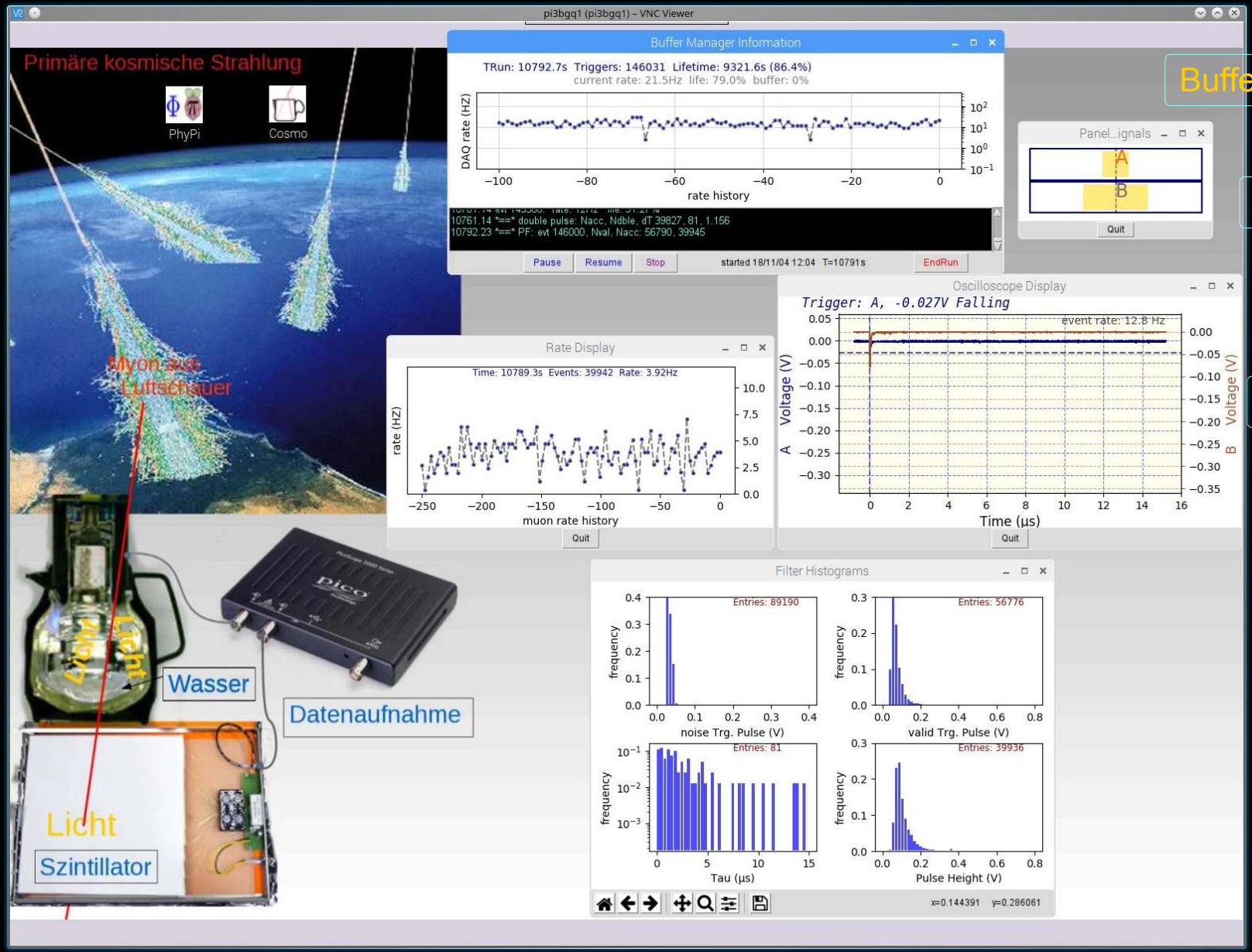
Doppelpuls entsteht, wenn ein Myon nach Durchgang gestoppt und das Elektron aus dem Zerfall im Detektor nachgewiesen wird

→ Messung der Myon-Lebensdauer

Aufbau am Raspberry Pi

- Einheitliche Auslese mit Oszilloskop von
  - Kamiokanne u.
  - CosMO-Panels
- hohe Transparenz des Messverfahrens
- gleiche Art der Datenauswertung
- Einfache Bereitstellung von Software

Bildschirmansicht der laufenden Datennahme



BufferManager Info

Event-Display

Oszillogramm

Pulshöhen- & Lebensdauerverteilungen

Praktischer Teil II

Praktischer Teil 2

Komplexere Messaufgaben mit phypi.py und vorbereiteten Konfigurationen

Stationen mit verschiedenen Aufbauten:

- Kraftsensor (A.S.)
- Lade- und Entlade-Kurven am Kondensator mit ADS1115 (A.S)

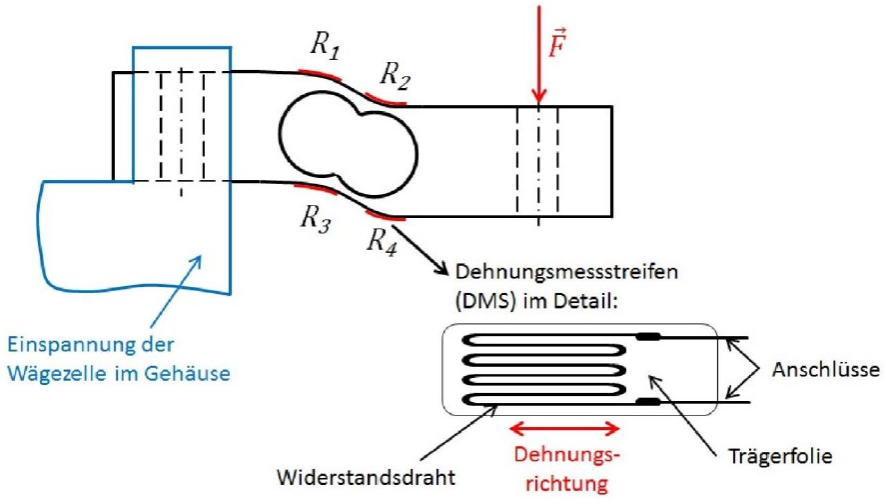
- digitale Sensoren am I²C-Bus des Raspberry PI (G.Q.):
  Stromsensor INA219
  Beschleunigungssensor am I2C-Bus
  Mehrkanal-Messung mit ADS1115 (Diodenkennlinien)
- PicoScope mit Geophon oder Mikrofon (G.Q.)

- Detektoren des Netzwerks Teilchenwelt (C.H.) https://www.teilchenwelt.de
  Szintillator-Panels
  Kamiokanne

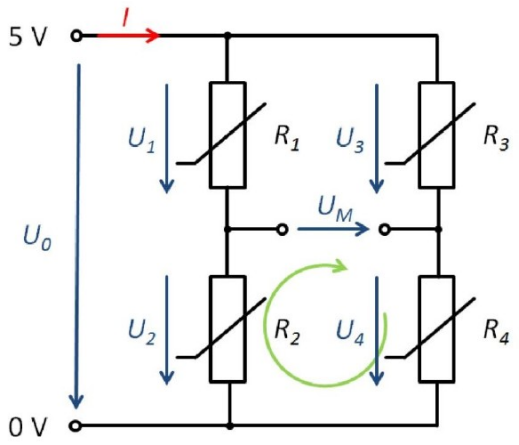
Software und Dokumentation siehe https:/github.com/GuenterQuast/picoCosmo

Kraftsensor

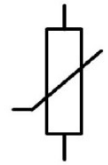
Wägezelle mit Dehnungsmesssteifen



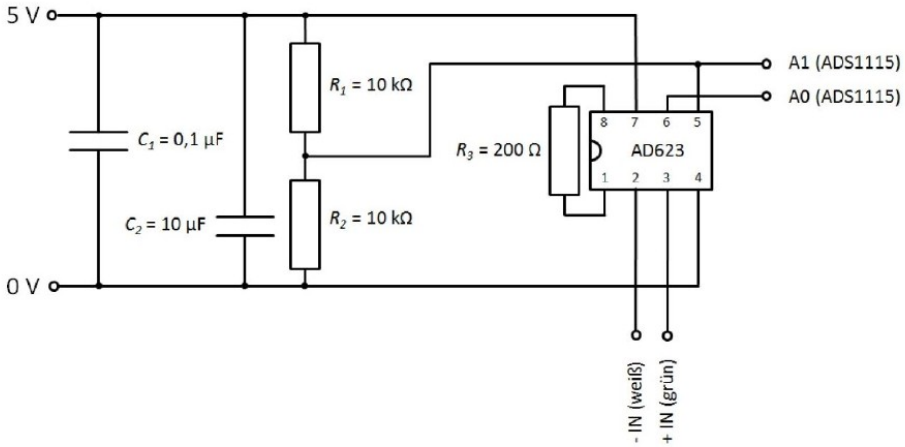
Schaltung der Dehnungsmesssteifen

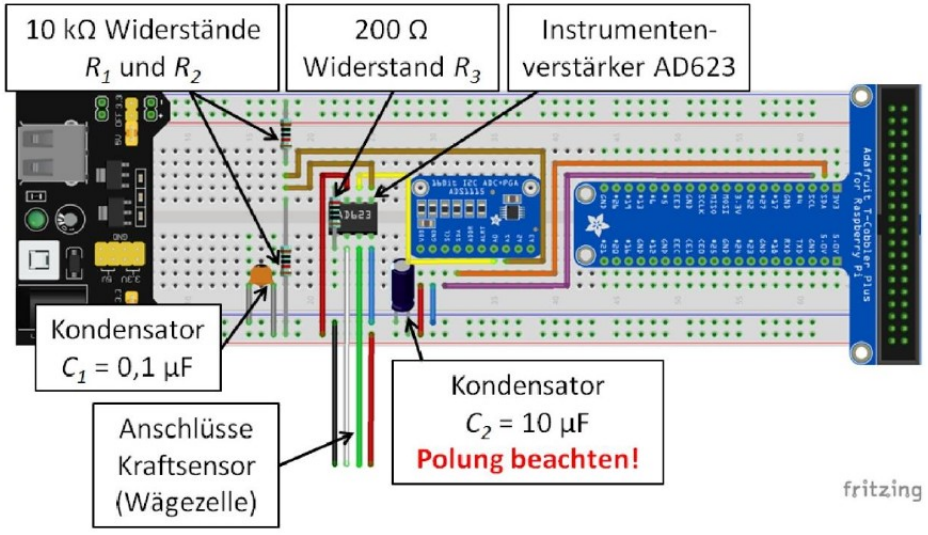


Schaltzeichen
für einen DMS



Aufbau mit Instrumentenverstärker

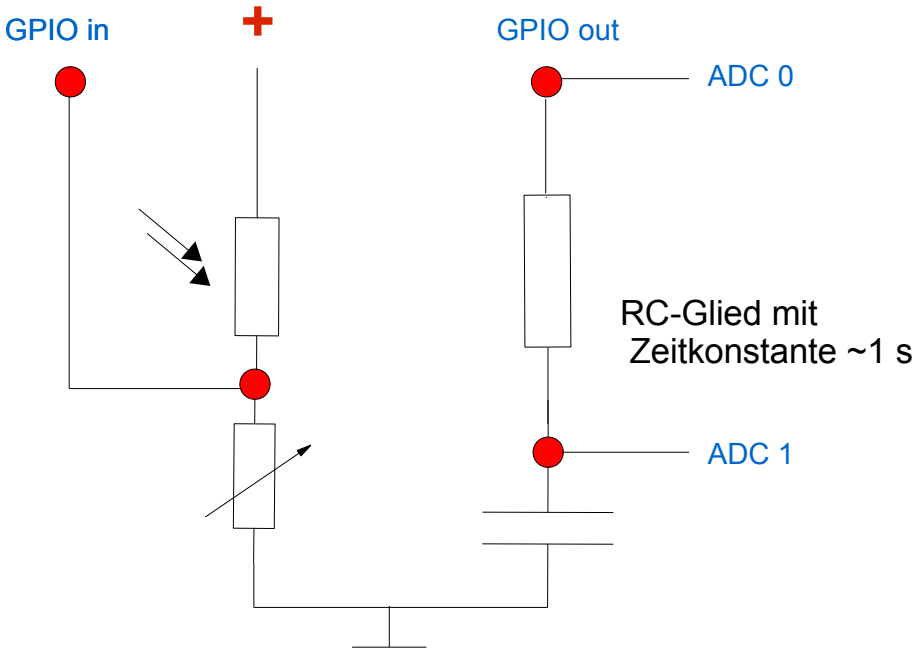




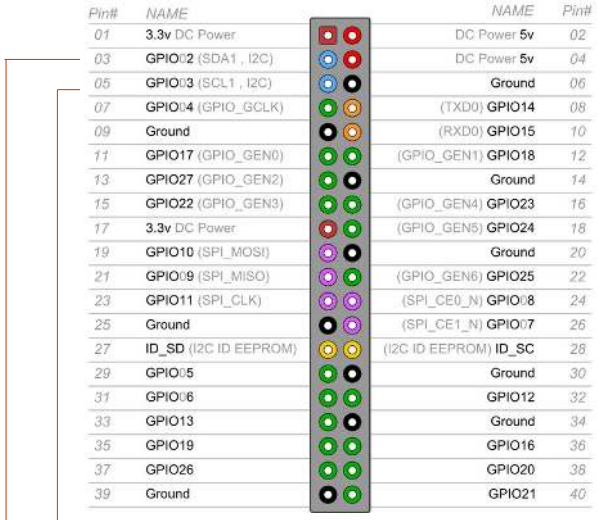
Lade- und Entladekurven

Rechteckspannung an GPIO-Pin

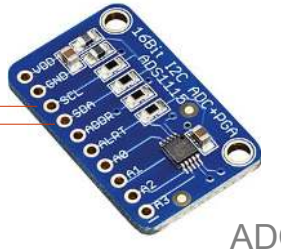
durch Abdecken einer Fotodiode an GPIO-Pin (analog Hell-Dunkelschaltung mit Kondensator statt LED; Anzeige der Spannungsverläufe mir ADC 1115 und Klasse DataLogger)



Sensoren am I²C-Bus



Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40



ADC ADS1115

Viele Sensoren haben ein digitales Interface

beliebt: serielle Übertragung über I²C-Bus

- bidirektional, zwei Leitungen (**SCL** und **SDA**)
- bis 5Mbis/s Datenrate
- 112 individuelle Adressen

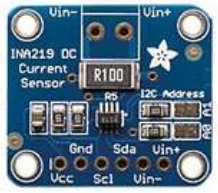
Treibersoftware sorgt für Ansteuerung der Chips und für die Datenübertragung.

Einfache Implementierung in PhyPiDAQ:

```
sensor = <DeviceClass>()
sensor.init()
sensor.acquireData(data)
  . . .
sensor.closeDevice()
```

MMA7451
Beschleunigungssensor

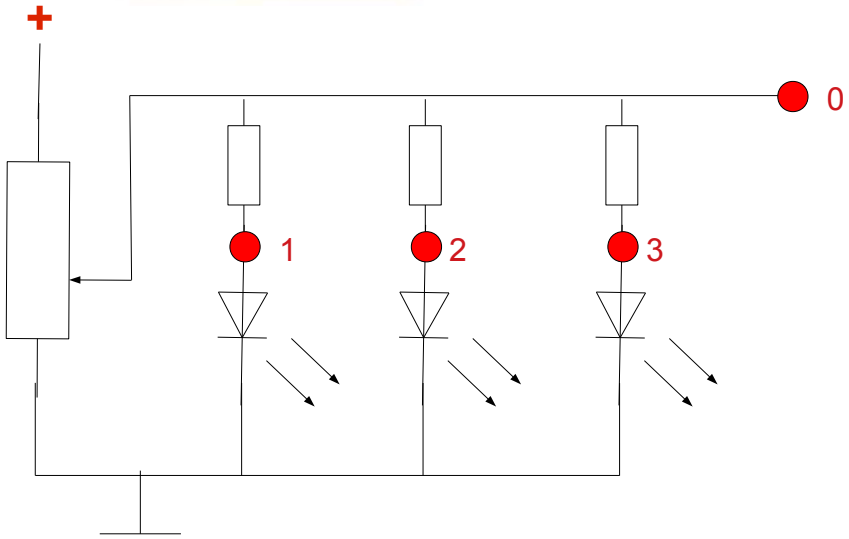
BMPx80
Druck- & Temperatursensor

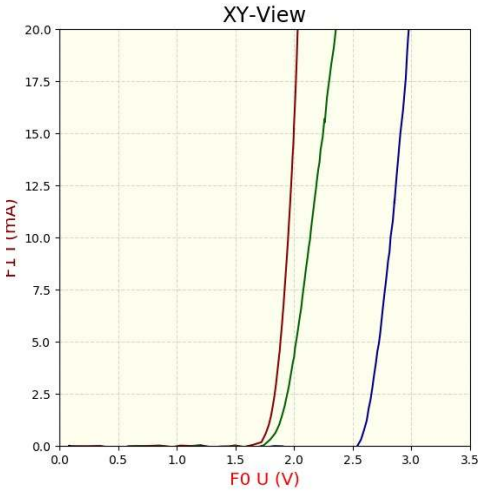


INA219 Strom-
& Spannungssensor

MAX31855 Thermo-
elementverstärker

Mehrere Kennlinien



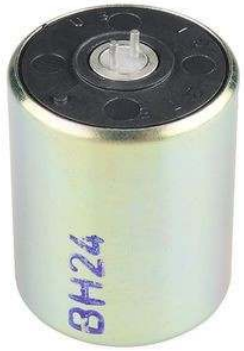



```
# Konfiguration fuer PhyPiDAQ

DeviceFile: ADS1115_Diode.yaml      # 4 aktive Kanaele mit Verst. 1

# Anwenden von Formeln auf die gemessenen Spannungen
# c0 ist Betriebsspannung, c1 - c3 Diodenspannungen
ChanFormula:
 - c1                   #  U Diode c1
 - (c0 - c1) / 0.120    #  I Diode c1
 - c2                   #  U Diode c2
 - (c0 - c2) / 0.100    #  I Diode c2
 - c3                   #  U Diode c3
 - (c0 - c3) / 0.082    #  I Diode c3

ChanLabels: [U, I, U, I, U, I]          # Labes fur Kanaele
ChanUnits: [V, mA, V, mA, V, mA]    # Einheiten
ChanColors: [red, darkred, green, darkgreen, blue, darkblue]

# Wertebereiche
ChanLimits:
 - [0., 3.5]   # U D1
 - [0., 10.]   # I D1
 - [0., 3.5]   # U D2
 - [0., 10.]   # I D2
 - [0., 3.5]   # U D3
 - [0., 10.]   # I D3

DisplayModule: DataLogger
Chan2Axes: [0,1,0,1,0,1]
XYmode:   true              # enable/disable XY-display
xyPlots:        # channels to display as x-y graph
 - [0,1]
 - [2,3]
 - [4,5]

Interval: 0.1               # Datennahme-Intervall
```

Geophon am PicoScope

Mit einem Oszilloskop lassen sich Effektivwerte von Wechselsignalen bestimmen – z.B. Lautstärke-Messung

oder – hier -
ein „Erdbeben-sensor“ (Sm-24)



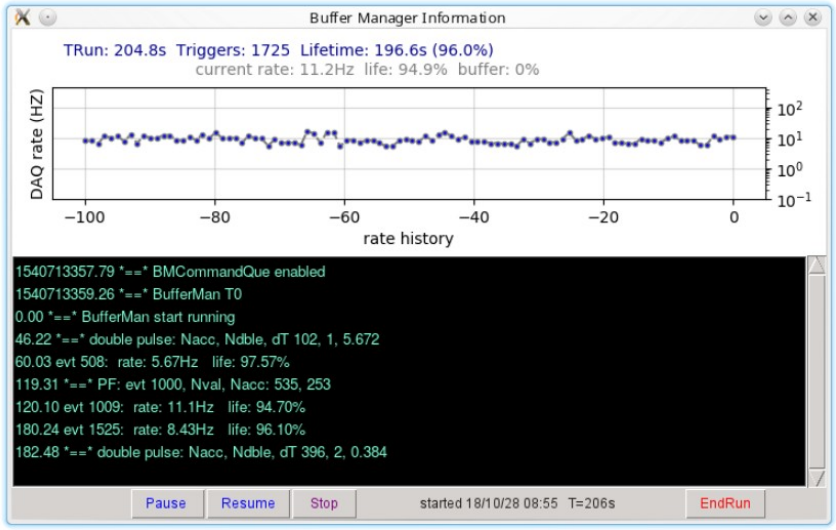
- 200 Werte in 20 ms aufnehmen
- quadratischen Mittelwert bilden
- Effektivwerte mit PhyPiDAq darstellen

Die Klasse PSConfig
erledigt Auslese und Mittelwertbildung

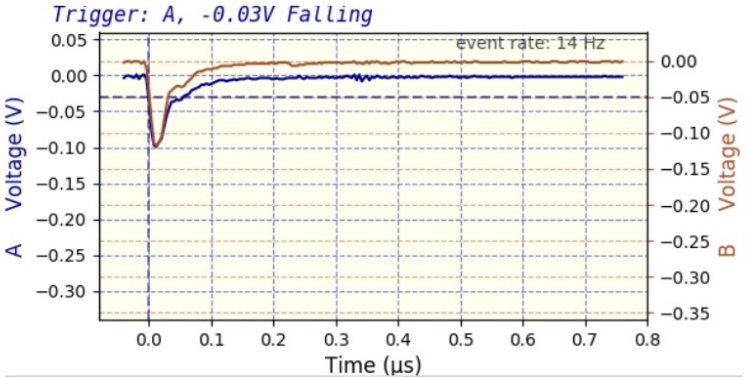
```
# Messung von Effektiv-Werten mit PS2000(A)
DAQModule: PSConfig

PSmodel: '2000'      # PS model 220xA
## PSmodel: '2000a'     # PS model 2x0xB

# channel configuration
picoChannels: [A]
ChanRanges: [0.05]
ChanModes: [AC]
## ChanOffsets: [-1.95, -1.95] # !!! not for A
series

sampleTime: 2.0E-02
Nsamples: 200

# trigger
trgActive: false  # true to activate
trgChan: A
trgThr: 0.
trgTyp: Rising
trgTO: 4  # set short time-out for A series
       # values<4 lead to readout instabilities

## pretrig: 0.05 # !!! not for A series

# signal generator
# frqSG: 100.E+3 # put 0. do disable
frqSG: 0.

# special flags for PhyPiDAQ
ChanAverages: ['rms'] # 'mean'
```

Detektor für Kosmische Strahlung mit PicoScope

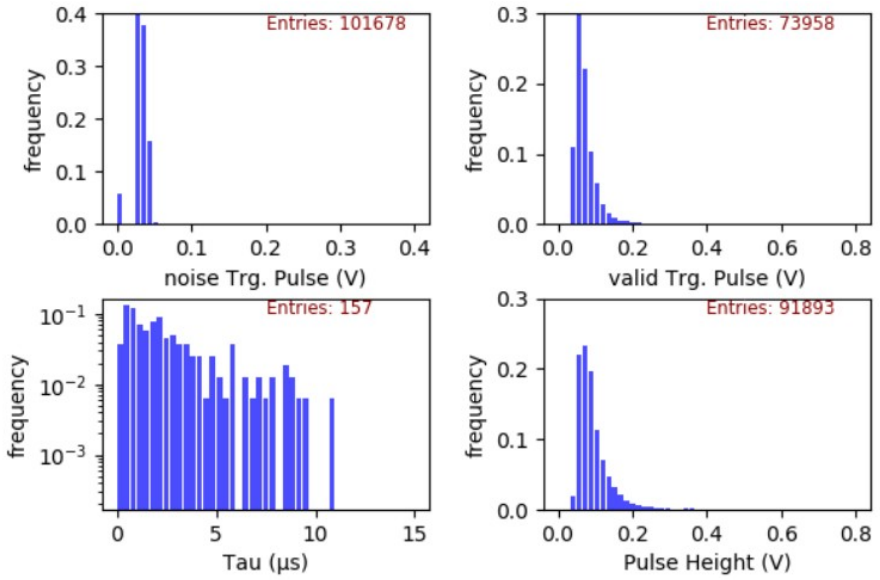
Verlauf der Datennahme



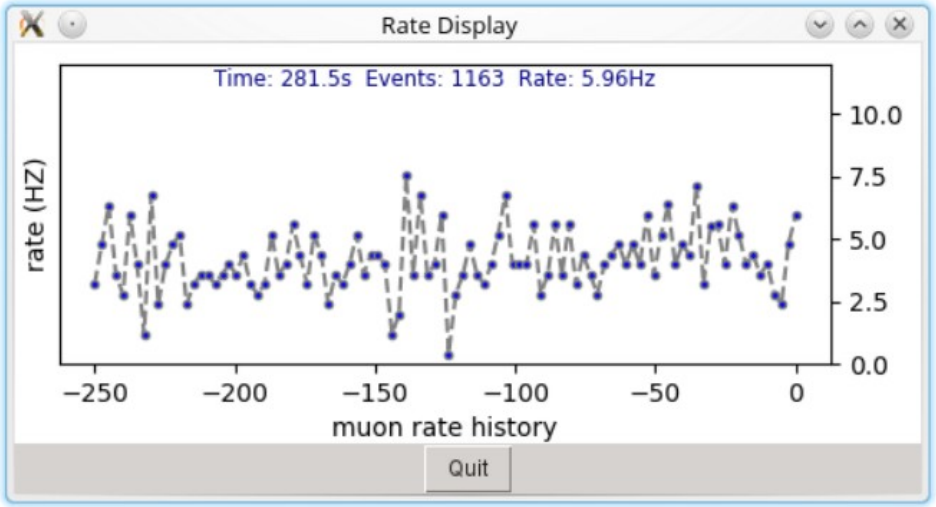
Oszilloskop-Anzeige



Häufigkeitsverteilungen: Pulshöhen u. Myon-Lebensdauern



Rate der akzeptierten Myonen (Zweierkoinzidenz)



Links

- Auslese von PicoScope USB-Oszilloskopen
  https://github.com/GuenterQuast/picoDAQ

- Auslese von Detektoren für Kosmische Strahlung
  https://github.com/GuenterQuast/picoCosmo

- Netzwerk Teilchenwelt
  http://www.teilchenwelt.de

NETZWERK TEILCHENWELT

- Digitales Messen im Physikunterricht mit Raspberry Pi
  htts://github.com/GuenterQuast/PhyPiDAQ

Die Zukunft von PhyPiDAQ

Der Einsatz von „Alltagssensoren“ (wie in Geräten oder dem Handy verwendet)
führt zu einer starken Kostenreduktion für Messtechnik

→ **vom Demonstrationsexperiment zum Schülerversuch**

Die gleiche Messtechnik wie Schüler auch verstärkt im Physikunterricht verwenden
- wo möglich, „moderne“ Sensoren transparent einsetzen
- vermehrt Experimente durch Schüler ausführen lassen

Die nächsten Schritte:

– Unterstützung für weitere Sensoren

– Erprobung des Einsatzes für klassische Experimente im Physikunterricht
Bau entsprechender „Boxen“ zum leichteren praktischen Einsatz

– Erweiterung der Software durch neues Projekt am KIT
„Studierende entwickeln Open-Source-Lehrsoftware“

– Feedback aus der Praxis einholen

bitte um rege Diskussion !