PhyPiDAQ

Data Acquisition and analysis for Physics education with Raspberry Pi

This is the **English** version of the documentation.

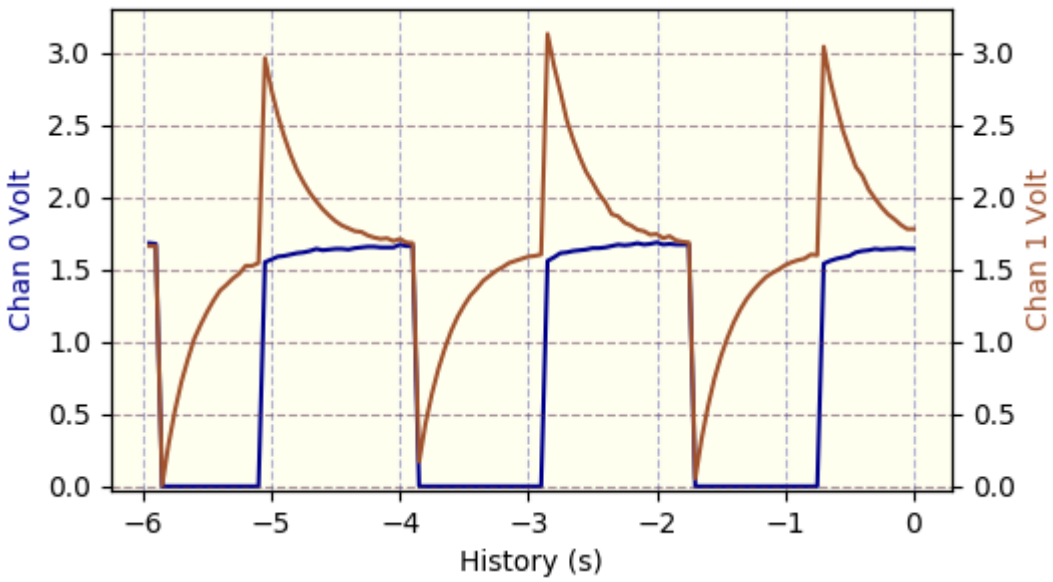
For German readers:
Die **deutsche Version** dieses Dokuments findet sich unter dem Link README_de.md bzw. README_de.pdf .

This *python3* code provides some basic functionality for data acquisition and visualisation like data logger, bar-chart, XY- or oscilloscope display and data recording on disk.

In addition to the GPIO inputs/outputs of the Raspberry Pi, the analog-to-digital converters ADS1115 and MCP3008 and PicoScope USB-oscilloscopes are supported as input devices for analog data, as well as a number of digital sensors using protocols like I²C or SPI.

The package provides an abstraction layer for measurement devices and sensors connected to a Raspberry Pi. Dedicated classes for each device provide a simple, unified interface, containing only the methods `init(<config_dictionary>)`, `acquireData(buffer)` and `close()`. Simple examples with minimalist code illustrate the usage. The graphical user interface `phypi.py` and the script `run_phypi.py` provide a configurable environment for more complex measurements.

Fig. 1: Visualisation of the time dependence of two signals connected to an ADC



Quick-start guide

After installation - see below - a number of unified classes for data acquisition, visualisation and recording is available from the sub-directory `./phypidaq`. Each supported device needs a specific configuration, which is read from configuration files in sub-directory `./config`. The overall configuration is given in files of type `.daq`, specifying which devices and display modules to use, the readout rate, calibrations or analytical formulae to be applied to recorded data, or ranges and axis labels of the graphical output.

The graphical user interface `phypi.py` aids in the administration of the configuration options and can be used to start data acquisition. In this case, configurations and produced data files are stored in a dedicated sub-directory in `$HOME/PhyPi`. The sub-directory name is derived from a user-defined tag and the current date and time.

Data acquisition may also be started via the command line:

```
run_phypi.py <config_file_name>.daq
```

If no configuration file is given, the default `PhyPiConf.daq` is used.

The sub-directory `./examples` contains a number of simple *python* scripts illustrating the usage of data acquisition and display modules with minimalist code.

Configuration files for PhyPiDAQ

The script `run_phypi.py` allows users to perform very general measurement tasks without the need to write custom code. The options for configuration of input devices and their channels as well as for the display and data storage modules are specified in a global configuration file of type `.daq` (in `yaml` markup language), which contains references to device configuration files of type `.yaml`.

Main configuration file

A typical, commented example of the main configuration file is shown here:

file PhyPiConf.daq

```
# Configuration Options for PhyPiDAQ

# device configuration files
DeviceFile: config/ADS1115Config.yaml
#DeviceFile: config/MCP3008Config.yaml
#DeviceFile: config/PSConfig.yaml
#DeviceFile: config/MAX31865Config.yaml
#DeviceFile: config/GPIOCount.yaml

## an example for multiple devices
#DeviceFile: [config/ADS1115Config.yaml, config/ GPIOCount.yaml]

DisplayModule: DataLogger
# DisplayModule: DataGraphs  # text, bar-graph, history and xy-view
Interval: 0.1                    # logging interval
XYmode:     false                # enable/disable XY-display

# channel-specific information
ChanLabels: [(V), (V) ]          # names and/or units for channels
ChanColors: [darkblue, sienna]    # channel colours in display

# eventually overwrite Channel Limits obtained from device config
##ChanLimits:
## - [0., 1.]   # chan 0
## - [0., 1.]   # chan 1
```

```
## - [0., 1.]   # chan 2

# calibration of channel values
#  - null    or  - <factor> or  - [ [ <true values> ], [ <raw values> ] ]
#ChanCalib:
#  - 1.                       # chan0: simple calibration factor
#  - [ [0.,1.], [0., 1.] ]    # chan1: interpolation: [true]([<raw>] )
#  - null                     # chan2: no calibration

# apply formulae to calibrated channel values
#ChanFormula:
#  - c0 + c1  # chan0
#  - c1          # chan1
#  - null        # chan2 : no formula

# name of output file
#DataFile:   testfile.csv     # file name for output file
DataFile:   null             #      use null if no output wanted
#CSVseparator: ';'           # field separator for output file, defaults to ','
```

Device configuration files

Typical, commented examples of device configurations are shown below. The device configuration file for the analog-to-digital converter **ADS1115** specifies the active channels, their ranges and single or differential operation modes.

file ADS1115Config.yaml

```
# example of a configuration file for ADC ADS1115

DAQModule: ADS1115Config    # phypidaq module to be loaded

ADCChannels: [0, 3]         # active ADC-Channels
                        # possible values: 0, 1, 2, 3
                        # when using differential mode:
                        #    -  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] # enable differential mode for Channels

Gain: [2/3, 2/3]          # programmable gain of ADC-Channel
                          #   possible values for Gain:
                          #     - 2/3 = +/-6.144V
                          #     -   1 = +/-4.096V
```

```
                          #     -   2 = +/-2.048V
                          #     -   4 = +/-1.024V
                          #     -   8 = +/-0.512V
                          #     -  16 = +/-0.256V
sampleRate: 860           # programmable Sample Rate of ADS1115
                          #    possible values for SampleRate:
                          #    8, 16, 32, 64, 128, 250, 475, 860
```

The **USB-oscilloscope** PicoScope can also be used as data logger. In this case the average of a large number of measurements at high rate is taken. Choosing a measurement time of 20 ms very effectively eliminates 50 Hz noise.

file PSconfig.yaml

```
# example of a configuration file for PicoScope 2000 Series

DAQModule: PSConfig

PSmodel: 2000a

# channel configuration
picoChannels: [A, B]
ChanRanges: [2., 2.]
ChanOffsets: [-1.95, -1.95]
ChanModes: [DC, DC]
sampleTime: 2.0E-02
Nsamples: 100

# oscilloscope trigger
trgActive: false  # true to activate
trgChan: A
#trgThr: 0.1
#pretrig: 0.05
#trgTyp: Rising
#trgTO: 1000  # time-out

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

Examples of other devices like the analog-to-digital converter MCP3008, of rate measurements via the GPIO pins of the Raspberry Pi or temperature measurements with PT100 sensors and the resistance-to-digital converter MAX31865 are also contained in the configuration directory, see files `MCP3008Config.yaml`, `GPIOcount.yaml` or `MAX31865Config.yaml`, respectively.

Installation of PhyPiDAQ on a Raspberry Pi

This package relies on code from other packages providing the drivers for the supported devices:

- the Adafruit Pyhon MCP3008 library https://github.com/adafruit/Adafruit_Python_MCP3008
- the Adafruit Python ADX1x15 library https://github.com/adafruit/Adafruit_Python_ADS1x15
- components from the picoDAQ project https://github.com/GuenterQuast/picoDAQ
- the *python* bindings of the *pico-python* project by Colin O'Flynn https://github.com/colinoflynn/pico-python
- the low-level drivers contained in the Pico Technology Software Development Kit https://www.picotech.com/downloads

For convenience, installation files for external packages and for modules of this package in pip wheel format are provided in sub-directory *./whl*.

The visualization modules depend on *matplotlib.pyplot*, *Tkinter* and *pyQt5*, which must also be installed.

After setting up your Raspberry Pi with the actual stable Debian release *stretch*, the following steps should be taken to update and install all necessary packages:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-scipy
sudo apt-get install python3-matplotlib
sudo apt-get install python3-pyqt5

sudo pip3 install pyyaml

# PicoTech base drivers for picoScope USB devices
#   see https://www.picotech.com/support/topic14649.html
# after inclusion of the picotech raspbian repository:
sudo apt-get install libps2000a
# allow access of user pi to usb port
sudo usermod -a -G tty pi

# get PhyPiDAQ code and dependencies
mkdir git
cd git
git clone https://GuenterQuast/PhyPiDAQ
cd PhyPiDAQ/whl
sudo pip3 install *.whl
```

Overview of files contained in PhyPiDAQ

Programs

- `run_phypi.py`
  run data acquisition and display modules as specified in configuration files

- `phypi.py`
  graphical user interface to edit configuration files and start the script `run_phypi.py`

Modules

- `phypidaq/__init__.py`
  initialisation for package *phypidaq*
- `phypidaq/_version_info.py`
  version info for package *phypidaq*
- `phypidaq/ADS1115Config.py`
  class for handling of analog-to-digital converter ADS1115
- `phypidaq/MCP3008Config.py`
  class for handling of analog-to-digital converter MCP3008
- `phypidaq/GPIOCount.py`
  class reading rates from GPIO pins
- `phypidaq/MAX31865Config.py`
  class handling MAX31865 resistance-to-digital converter
- `phypidaq/PSConfig.py`
  class handling PicoScope USB oscilloscopes
- `phypidaq/mpTkDisplay.py`
  background-process handling data visialation
- `phypidaq/DataLogger.py`
- class for display of data histories and xy diagrams
- `phypidaq/DataGraphs.py`
  general display module for data as bar graphs, history plots and xy-graphs
- `phypidaq/DataRecorder.py`
  store data in CSV format

Configuration files

- `PhyPiConf.daq`
  main configuration file, depends on device configurations in sub-directory *config/*
- `config/ADS1115.yaml`
- `config/GPIOCount.yaml`
- `config/MCP3008.yaml`
- `config/PSConfig.yaml`

Examples

- `examples/runOsci.py`
  run an oscilloscope display, configuration as specified in *.yaml* file
- `example/PSCosci.yaml`
  configuration file for PicoScope USB oscilloscope, used by *runOsci.py*
- `examples/poissonLED.py`
  generate a random signal following Poisson statistics on a GPIO pin
- `examples/FreqGen.py`
  generate a fixed frequency signal on a GPIO pin

Documentation

- `doc/Kurs_digitale_Messtechnik.md (.pdf)`
  German only: Introductory course to measuring with the Raspberry Pi
- `doc/Einrichten_des_Raspberry_Pi.md (.pdf)`
  German only: setting up the Raspberry Pi for this project
- `doc/Komponenten_fuer_PhyPi.md (.pdf)`
  recommended components for this project