pymepix Documentation

Release 0.5

Ahmed F. Al-Refaie

Jul 02, 2020

Contents

Pymepix documentation

CHAPTER 1

Introduction

Pymepix is intended to bridge the gap between Timepix3 and Python. The goal of the library is to allow a user without deep technical knowledge of Timepix3 to establish a connection, start acquisition, and retrieve and plot pixel and timing information in as few lines of code as possible; at the same time it provides all details to unleash the full power of Timepix3-SPIDR hardware. This is achieved by classes that act as a black-box, handling all of the low level TCP communication and decoding of the UDP data-stream, presenting them in a pythonic fashion. More advanced and lower-level control of SPIDR and Timepix3 is still available from these black-box classes or can be established directly by the user. For easy installation, it only depends on the standard python library, numpy and scikit-learn.

CHAPTER 2

Getting started

2.1 Installing

2.1.1 Installing from PyPI (platform-independent)

Simply to:

```
pip install pymepix
```

This should install all dependencies.

2.1.2 Installing from git source directly (platform-independent)

You can clone pymepix from our main git repository:

```
git clone https://<desy username>@stash.desy.de/scm/cmipublic/timepix.git
```

Move into the pymepix library:

```
cd timepix/pymepix
```

Then, just do:

```
pip install .
```

To build documentation do:

```
python setup.py build_sphinx
```

2.2 Dependencies

The majority of pymepix only depends on numpy. To use centroiding, the sklearn package is required

CHAPTER 3

Connecting and Configuring

3.1 Connecting

Connecting to SPIDR can be done with:

```
>>> timepix = Pymepix(('192.168.1.10',50000))
```

Where the IP address is the one seen on the OLED screen on timepix. The number of devices can be found using:

```
>>> len(timepix)
1
```

Meaning we have one device. To access this device directly, use:

```
tpx0 = timepix[0]
```

And to check the device name:

```
>>> tpx0.deviceName
W0026_K08
```

3.2 Configuring

To set the biasVoltage to 50 Volts in spidr you can do:

```
>>> timepix.biasVoltage = 50
```

Setting the we can manage its settings directly. To easily setup the device we can use a SoPhy config file (.spx):

```
tpx0.loadConfig('myFile.spx')
```

This sets up all the DAC setting and pixel configurations. Individual parameters can also be set for example. To set the fine threshold to 100 mV do:

```
>>> tpx0.Vthreshold_fine = 100
```

pixel threshold configurations can be set by passing a 256x256 numpy array:

```
import numpy as np
tpx0.pixelThreshold[...] = 0
```

The same for pixel masks, to set a checkboard mask do:

```
tpx0.pixelMask[::2] = 1
```

These need to be uploaded to timepix before they take effect:

```
>>> tpx0.uploadPixels()
```

The full list of parameters that can be set can be found in *pymepix.timepixdevice module*.

CHAPTER 4

Acquisition

Acquisition can be started and stopped by:

```
import time
from pymepix import Pymepix

#Connect
timepix = Pymepix(('192.168.1.10',50000))

#Start acquisition
timepix.start()

#Wait
time.sleep(1.0)

#Stop acquisition
timepix.stop()
```

Pymepix provides data as a tuple given by (`MessageType`,data). These are explained in *Data Formats*. Retrieving the data can be done in to ways: Polling or Callback

4.1 Polling

Polling is where pymepix will place anything retrieved from Timepix into a ring polling buffer. This is the default mode but to reenable it you can use:

```
>>> timepix.enablePolling(maxlen=1000)
```

where *maxlen* describes the maximum number of elements in the buffer before older values are overwritten.

The user can retrieve this data by using:

```
>>> timepix.poll()
(MessageType.RawData,(array[9873240589723458980 2345,dtype=uint8],12348798))
```

If there is nothing in the polling buffer then a `PollBufferEmpty` exception is raised The poll buffer is limited in size but can be extended by doing:

```
>>> timepix.pollBufferLength = 5000
```

This will clear all objects using the polling buffer.

4.2 Callback

The callback method allows the user to deal with the data immediately when it is recieved. Setting this will clear the polling buffer of any contents.

To set a callback, first you need a function, for an example:

```
def my_callback(data_type,data):
    print('My callback is running!!!!')
```

The format of the function must accept two parameters, `MessageType` and an extra data parameter. These are explained in *Data Formats*. Now to make pymepix use it simply do:

```
>>> timepix.dataCallback = my_callback
```

Now when acquisition is started:

```
>>> timepix.start()
```

The output seen is:

```
My callback is running!!!!
My callback is running!!!!
My callback is running!!!!
My callback is running!!!!
My callback is running!!!!
```

4.3 Pipelines

Pymepix uses pipelines objects in order to process data. Each pipeline is set for each timepix device so each timepix can have a different data pipeline. You can configure them to postprocess or output data in certain ways. For example the `PixelPipeline` object will read from a UDP packet stream and decode the stream into *pixel x*, *pixel y*, *time of arrival* and *time over threshold* arrays. All data is progated forward through the pipeline so both UDP packets and decoded pixels are output.

To use the (default) `PixelPipeline` pipeline on the first connected timepix device you can do:

```
from pymepix.processing import PixelPipeline,CentroidPipeline

timepix[0].setupAcquisition(PixelPipeline)
```

If you need centroid you instead can do:

```
>>> timepix[0].setupAcquisition(CentroidPipeline)
```

Configuring the pipelines can be done using the acquisition property for the timepix device, for example to enable TOFs you can do:

```
>>> timepix[0].acquisition.enableEvents = True
```

A list of pipelines and setting can be found in *pymepix.processing.acquisition module*

CHAPTER 5

Data Formats

Contains a list of possible data formats output during acquisition. Each entry of the data section represents another element in the tuple. Example shows how to read the data through polling

5.1 UDP Packets

Data Type: `MessageType.RawData`

Data:

array(uint64) list of UDP packets

uint64 global timer from Timepix at time packets were recieved

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.RawData:
    packets,longtime = data
```

5.2 Decoded Pixels

Data Type: `MessageType.PixelData`

Data:

array(uint64) pixel x position

array(uint64) pixel y position

array(float) global time of arrival in seconds

array(uint64)) time over threshold in nanoseconds

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.PixelData:
    x,y,toa,tot = data
```

5.3 Decoded Triggers

Data Type: `MessageType.TriggerData`

Data:

array(uint64) trigger number

array(float) global trigger time in seconds

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.TriggerData:
    t_num,t_time = data
```

5.4 Time of Flight/Event

Data Type: `MessageType.EventData`

Data:

array(uint64) trigger number

array(uint64) pixel x position

array(uint64) pixel y position

array(float) time of flight relative to its trigger in seconds

array(uint64)) time over threshold in nanoseconds

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.EventData:
    trigger,x,y,tof,tot = data
```

5.5 Centroid Data

Data Type: `MessageType.CentroidData`

Data:

array(uint64) trigger number

array(uint64) center of mass x position

array(uint64) center of mass y position

array(uint64) total area

array(uint64) total time over threshold

array(uint64) Ignore (used in future)

array(uint64) Ignore (used in future)

array(uint64)) time of flight

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.CentroidData:
    trigger,x,y,area,integral,nu,nu,tof = data
```

CHAPTER 6

Examples

Starting timepix and polling data:

```
import pymepix
from pymepix.processing import MessageType
import numpy as np

#Connect to SPIDR
timepix = pymepix.Pymepix(('192.168.1.10',50000))

#Set bias voltage
timepix.biasVoltage = 50

#Set pixel masks
timepix[0].pixelThreshold = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].pixelMask = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].uploadPixels()

#Start acquisition
timepix.start()

while True:
    try:
        #Poll
        data_type,data = timepix.poll()
    except pymepix.PollBufferEmpty:
        #If empty then just loop
        continue

    #Handle Raw
    if data_type is MessageType.RawData:

        print('UDP PACKET')

        packets,longtime = data
```

(continues on next page)

(continued from previous page)

```
        print('Packet ',packets)
        print('Time', longtime)

    #Handle Pixels
    elif data_type is MessageType.PixelData:

        print('I GOT PIXELS!!!!')

        x,y,toa,tot = data

        print('x',x)
        print('y', y)
        print('toa', toa)
        print('tot',tot)

#Stop
timepix.stop()
```

Using callbacks to acquire:

```
import pymepix
from pymepix.processing import MessageType
import numpy as np
import time

#Connect to SPIDR
timepix = pymepix.Pymepix(('192.168.1.10',50000))

#Set bias voltage
timepix.biasVoltage = 50

#Set pixel masks
timepix[0].pixelThreshold = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].pixelMask = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].uploadPixels()

#Define callback
def my_callback(data_type,data):
    print('MY CALLBACK!!!!')
    #Handle Raw
    if data_type is MessageType.RawData:

        print('UDP PACKET')

        packets,longtime = data

        print('Packet ',packets)
        print('Time', longtime)

    #Handle Pixels
    elif data_type is MessageType.PixelData:

        print('I GOT PIXELS!!!!')

        x,y,toa,tot = data
```

(continues on next page)

(continued from previous page)

```
        print('x',x)
        print('y', y)
        print('toa', toa)
        print('tot',tot)

#Set callback
timepix.dataCallback = my_callback

#Start acquisition
timepix.start()
#Wait 5 seconds
time.sleep(5.0)
#Stop
timepix.stop()
```

CHAPTER 7

PymepixAcq - Command line

Included with pymepix is a command line code using the pymepix library to acquire from timepix. It is run using:

```
pymepix-acq --time 10 --output my_file
```

Doing:

```
pymepix-acq --help
```

Outputs the help:

```
usage: pymepix-acq [-h] [-i IP] [-p PORT] [-s SPX] [-v BIAS] -t TIME -o OUTPUT
                   [-d DECODE] [-T TOF]

Timepix acquisition script

optional arguments:
-h, --help                 show this help message and exit
-i IP, --ip IP             IP address of Timepix
-p PORT, --port PORT       TCP port to use for the connection
-s SPX, --spx SPX          Sophy config file to load
-v BIAS, --bias BIAS       Bias voltage in Volts
-t TIME, --time TIME       Acquisition time in seconds
-o OUTPUT, --output OUTPUT output filename prefix
-d DECODE, --decode DECODE Store decoded values instead
-T TOF, --tof TOF          Compute TOF if decode is enabled
```

TODO: MORE DOCS

- genindex
- modindex
- search

CHAPTER 8

Pymepix API

Full API for pymepix

8.1 pymepix

8.1.1 pymepix package

8.1.1.1 Subpackages

pymepix.SPIDR package

Submodules

pymepix.SPIDR.error module

pymepix.SPIDR.spidrcmds module

pymepix.SPIDR.spidrcontroller module

pymepix.SPIDR.spidrdefs module

pymepix.SPIDR.spidrdevice module

Module contents

pymepix.config package

pymepix.pymepix module

pymepix.timepixdef module

pymepix.timepixdevice module

8.1.1.3 Module contents

CHAPTER 9

Pymepix Documentation

Pymepix is a python library for interfacing, controlling and acquiring from SPIDR-Timepix detectors.