pySiPM Documentation

Release 0.2

Edoardo Proserpio
Massimiliano Antonello
Romualdo Santoro

Aug 13, 2020

CONTENTS

1 Introduction **1**
- 1.1 What is pySiPM? . . . 1
- 1.2 Why it was created? . . . 1
- 1.3 Who has created it? . . . 1

2 Installation **3**
- 2.1 Requirments . . . 3
- 2.2 Linux How To . . . 4
- 2.3 Compilation of shared libraries . . . 4

3 Files Structure **5**
- 3.1 Main . . . 5
- 3.2 Variables . . . 6
- 3.3 Functions . . . 7
- 3.4 Other . . . 9

4 Setup **11**
- 4.1 External file setup . . . 11
- 4.2 Manual setup . . . 11
- 4.3 Command line options . . . 12

5 Variables **13**

6 Theory **17**
- 6.1 SiPM . . . 17
- 6.2 Cell Recovery . . . 18
- 6.3 Dark Counts . . . 18
- 6.4 Optical Crosstalk . . . 18
- 6.5 Afterpulsing . . . 18
- 6.6 Signal Shaping . . . 18
- 6.7 Signal Analysis . . . 18

7 Indices and tables **19**

Index **21**

CHAPTER

ONE

INTRODUCTION

1.1 What is pySiPM?

pySiPM is a toolkit developed with the aim of simulating SiPM signals. Even thou it has been initially developed to accomplish a specific task it has evolved in a way such that it can simulate generic SiPM signals.

1.2 Why it was created?

Originally it has been developed to simulate the SiPM digitized signals using the information given by a Geant4 simulation of the IDEA Dual Readout Calorimeter in which about 130 millions of SiPMs are coupled with scintillating and cherenkov fibers. Its aim is to study the effects induced by the SiPM readout system in the detector response and eventually to help in the right choice of sensors and electronic setup. It is also used to simulate the full geometry of IDEA Calorimeter and reconstruct high energy physics events with the granularity offered by the SiPM readout system.

Its aim is to help in the choice of the right SiPM sensor for a specific task and show how the stochastic noise effects may affect the values measured. Its aim is not to describe a SiPM in the most detailed way but to show how different parameters of the SiPM, signal shape and electric readout may affect the value measured by the user.

1.3 Who has created it?

pySiPM has been developed for the IDEA collaboration at University of Insubria, department of Science and HighTech.
The group that has developed pySiPM has an expertise in Silicon PhotoMultipliers devices and their applications.

- Edoardo Proserpio, physics student at Insubria University eproserpio@studenti.uninsubria.it
- PhD Massimiliano Antonello, research fellow at INFN Milano massimiliano.antonello@mi.infn.it
- PhD Romualdo Santoro, researcher and professor at Insubria University romualdo.santoro@uninsubria.it

CHAPTER

TWO

INSTALLATION

Tip: At the moment pySiPM is just a collection of .py files including all the necessaries functions to simulate a SiPM, hence you just need to clone the GitHub repository and make sure that you have installed all the needed Python modules.

2.1 Requirments

2.1.1 Obbligatory

- **Python**: pySiPM is created using Python3.6 but it should also work with newer versions of Python.
  - numpy
  - matplotlib
  - uproot
  - h5py
- **Cern ROOT**: Version 6.19 and newer. It has been tested on ROOT versions 6.19 - 6.20 and 6.20.2. At the moment ROOT is only used for visualization but in future versions it will be used for I/O operations. Download ROOT
- **Fortran**: f2py is used to wrap fortran code. f2py is installed by default when installing Numpy.

2.1.2 Optional

- **Cupy**: Cupy is used to take advantage of Nvidia GPUs computing power. It requires CUDA to be installed in the system. Download CUDA

2.2 Linux How To

When installing ROOT make sure to link it with Python3: `-DPYTHON_EXECUTABLE=\usr\bin\python3 -DPYTHON_INCLUDE_DIR=\usr\include\python3.6\Python.h`
If Cern ROOT is not installed it is better to install it **after** Nvidia CUDA, so it can find CUDA libraries and build upon them using `-Dcuda=ON`

To install Python packages needed:

```
python3 -m pip install numpy matplotlib uproot
```

To install CUDA (Nvidia Gpu needed) and Cupy (code example for Ubuntu 18.04):

```
wget http://developer.download.nvidia.com/compute/cuda/10.2/Prod/local_installers/
↪cuda_10.2.89_440.33.01_linux.run
sudo sh cuda_10.2.89_440.33.01_linux.run
```

To set environment variables in .bashrc add:

```
export CUDADIR=/usr/local/cuda-10.2
export PATH=$CUDADIR/bin${PATH:+:${PATH}}
LD_LIBRARY_PATH=$CUDADIR/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
LD_LIBRARY_PATH=$CUDADIR/extras/CUPTI/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

Then Cupy can be installed via pip:

```
python3 -m pip install cupy
```

For further help on CUDA installation check Nvidia installation guide: CUDA installation guide

To download CUDA for other architectures: Download CUDA

2.3 Compilation of shared libraries

Fortran libraries have to be compiled using f2py. A Makefile.py automatizes the procedure.
Before running the simulation make sure to compile the libraries by running:

```
python3 Makefile.py
```

Shared libraries can be eventually compiled manually by using:

```
f2py3.6 -c --opts = '-Ofast -march=native -mavx2' -m FortranFunctions␣
↪FortranFunctions.f90
```

CHAPTER

THREE

FILES STRUCTURE

In this section the file structure used to build the simulation is explained. You can find a brief description of the files used and they content. For a detailed explanation see the individual sections.

3.1 Main

The simulation is made of a series of function and variables that are used in the main file to generate a SiPM event. In the main file the number of cells triggered is calculated and the signal is generated accordingly. In this file the signal is also analysed.

The structure of the main file may be modified by the user to include more advanced analysis of the digitised. signal.

3.1.1 main.py

In this file the functions present in the *libs* folder are combined to generate a complete SiPM event

`main.`**`SiPM`**(*times*, *other*)

Function that calls all the procedures defined in libs to generate a complete SiPM event.

Parameters

- **`times`** (`np.ndarray`) – This array contains the arriving time of each photon on the SiPM sensor surface. This array is the input of the simulation.
- **`other`** (`tuple`) – This variable may contain other informations about the event generated. It can be the event id, the arriving time inserted in the simulation or the real number of photons inserted in the simulation. This tuple will be copied as it is in the output.

Returns

- **integral** (`double`) – The integral of the signal calculated in the integration gate
- **peak** (`double`) – The height of the signal in the integration gate
- **tstart** (`double`) – The time of arrival of the signal in ns defined as the first sample over the threshld of 1.5
- **other** (`tuple`) – The copy of the *other* variable given in the input
- **signal** (`np.ndarray(float32)`) – If the options -W is enabled the complete SiPM signal will be passed in the output. Otherwise this output is None

3.2 Variables

3.2.1 variables.py

The file *variables.py* contains all the variables needed to describe the SiPM sensor, the simulation behavior and the signal analysis.
Here is a brief description of all the parameters with their default value.

- SiPM Settings:
  - `SIZE = 1` Size of the SiPM sensitive area in mm (side)
  - `CELLSIZE = 25` Pitch of a single sensitive cell in um
  - `DCR = 200e3` Dark count rate in kHz
  - `XT = 0.03` Optical crosstalk probability
  - `AP = 0.02` Afterpulsing probability
  - `TRISE = 1` Rising time of the signal in ns
  - `TFALL = 50` Falling time of the signal in ns
  - `CELLRECOVERY = 20` Recovery time of the single SiPM cell in ns
  - `TAUAPFAST = 15` Time constant of afterpulses delay in ns (fast component)
  - `TAUAPSLOW = 85` Time constant of afterpulses delay in ns (slow component)
  - `CCGV = 0.05` Cell to cell gain variation (sigma)
  - `SNR = 30` Signal to noise ratio
  - `BASESPREAD = 0` Spread of the baseline of the signal (sigma)
- Signal analysis settings:
  - `SIGLEN = 500` Total length of the signal in ns
  - `SAMPLING = 0.1` Samlping time in ns (1 = 1ns, 0.1 = 100ps)
  - `INTSTART = 20` Start of the integration gate in ns
  - `INTGATE = 300` Length of the integration gate in ns
  - `PREGATE = 20` Length of the pregate
- Simulation settings:
  - `FASTDCR = False` Enables faster generation of dark count events
  - `FASTXT = False` Enable faster generation of crosstalk events
  - `FASTSIG = True` Enables faster computation of signals

For a more detailed description of each variable see . . .

3.3 Functions

3.3.1 libs/lib.py

This file contains all the functions that are used to generate the event and the SiPM signal.
Here is a brief description of all the functions with their signature.

libs.lib.**addDCR**(*rate*)
Function that generates times of dcr events.

Parameters **rate** (double) – Rate of dcr in kHz

Returns **dcrTime** – Array containing dcr times

Return type np.ndarray

libs.lib.**SiPMEventAction**(*time*, *XT*)
evtsGen(time,XT)

Function that calculates signal height for each SiPM cell and adds optical crosstalk events.

Parameters

- **time** (np.ndarray) – Array containing the time at which SiPM cells are fired
- **XT** (double) – Value of optical crosstalk probability

Returns

- **evtTimes** (np.ndarray(int32)) – Array containing the time at wich SiPM cells are fired, including xt events (sorted)
- **sigH** (np.ndarray(float32)) – Array containing the pulse height of each fired SiPM cell
- **idx** (np.ndarray(int16)) – Array containing the unique index of the hitted cells

libs.lib.**SiPMSignalAction**(*times*, *sigH*, *SNR*, *BASESPREAD*)
signalGen(times,sigH,SNR,BASESPREAD)

Function that passes signal height and times to the main function that generates single signals. Also adds noise.

Parameters

- **times** (np.ndarray(int32)) – Array containing the time at wich SiPM cells are fired, including xt events (sorted)
- **sigH** (np.ndarray(float32)) – Array containing the pulse height of each fired SiPM cell
- **SNR** (double) – The signal to noise ratio of the noise to add
- **BASESPREAD** (double) – Sigma of the value to add as baseline

Returns **signal** – Array containing the generated SiPM signal

Return type np.ndarray

libs.lib.**PulseCPU**(*t*, *h*)
Function that generates the signal from a single SiPM cell. This is the "fast" version that uses a pre-computed signal shape and translates it in time.

Parameters

- **t** (int32) – Time at which the cell is triggered
- **h** (float32) – The relative pulse height of the cell signal
- **gainvar** (float32) – Value of cell to cell gain variation for this signal
- **nap** (int32) – Number of afterpulses in this signal

Returns **s** – Array containing the generated cell signal

Return type np.ndarray

libs.lib.**sigPlot**(*signal*, *sigTimes*, *dcrTime*, *dev*)
Function that plots each signal pulse produced in the simulation.

Parameters

- **signal** (np.ndarray) – Array containing the generated SiPM signal
- **sigTimes** (np.ndarray) – Array containing all photons events times (including xt)
- **dcrTime** (np.ndarray) – Array containing dcr events times
- **dev** (str) – String that describes the device on which the signal is computed

libs.lib.**initializeRandomPool**()
Function that initializes random seeds for each worker in the multiprocessing Pool

libs.lib.**somestats**(*output*)
Function that displays histograms of generated events

Parameters **output** (np.ndarray) – Array containing output of the simulation. This array contains the integral, peak and starting time in the first three columns.

Note: See docs for a detailed description of integral, peak and tstart

---

3.3.2 libs/libCPU.py

This file contains the functions used to compute the SiPM signal using the CPU.
Here is a brief description of all the functions with their signature.

libs.libCPU.**SiPMSignalAction**(*times*, *sigH*, *SNR*, *BASESPREAD*)
signalGen(times,sigH,SNR,BASESPREAD)

Function that passes signal height and times to the main function that generates single signals. Also adds noise.

Parameters

- **times** (np.ndarray(int32)) – Array containing the time at wich SiPM cells are fired, including xt events (sorted)
- **sigH** (np.ndarray(float32)) – Array containing the pulse height of each fired SiPM cell
- **SNR** (double) – The signal to noise ratio of the noise to add
- **BASESPREAD** (double) – Sigma of the value to add as baseline

Returns **signal** – Array containing the generated SiPM signal

Return type np.ndarray

libs.libCPU.**PulseCPU**(*t*, *h*)

Function that generates the signal from a single SiPM cell. This is the "full" version that computes the signal shape on CPU by evaluating the signal shape function.

Parameters

- **t** (int32) – Time at which the cell is triggered
- **h** (float32) – The relative pulse height of the cell signal
- **gainvar** (float32) – Value of cell to cell gain variation for this signal
- **nap** (int32) – Number of afterpulses in this signal

Returns **s** – Array containing the generated cell signal

Return type np.ndarray

3.3.3 libs/libGPU.py

This file contains the functions used to compute the SiPM signal using the GPU.
Here is a brief description of all the functions with their signature.

3.4 Other

3.4.1 scr

This folder contains the source file of the FORTRAN functions that are used to speed-up the simulation

3.4.2 files

This folder contains some files used by the simulation and contains pre-configurations of some SiPM models that can be eventually loaded. For an explanation on how to write and load an external configuration file see ...

CHAPTER

FOUR

SETUP

4.1 External file setup

The recommended way to change the parameters of the simulation or the SiPM is to launch the simulation with the option `-f filename.txt`. The file loaded will overwrite the default options. You can decide to overwrite all the default options or just some of them.

Note: The file loaded with `-f filename.txt` is read as a .py file, so comments must be preceded by # and every line will be executed as Python code.

Here is an example of file for a Hamamatzu SiPM S13615-1025

```
SIZE = 1
CELLSIZE = 25
DCR = 100e3
XT = 0.01
AP = 0.03
SNR = 30
CCGV = 0.05

INTSTART = 20
INTGATE = 300
PREG = 20
```

The list of editable variables can be found at . . . with a detailed description at . . .
Configuration files for some SiPM models are present in the *files* folder and it is a good practice to put your configuration files here.

4.2 Manual setup

It is also possible to manually change the values in the *variables.py* file. Keep in mind that changes in this files are permanent.

Danger: Editing the *variables.py* file may change the behavior of the simulation in an unexpected way! Edit this file only if you know what you are doing.

4.3 Command line options

Some command line options can be parsed to de simulation and can be used to quickly change some parameters of the simulation.

-H	Shows an help message and exits. Using `-h` will display the ROOT help message.
-V	Shows the software version and exits.
-q	Switch to silent operation of the simulation.
-G	Displays each generated signal (only for debugging purposes).
-g	Shows histograms of integral, peak and starting time at the end of the simulation.
-NDCR	Turns off dark count events generation.
-NXT	Turns off the generation of optical crosstalk events.
-NAP	Turns off the generation of afterpulses events.
-FDCR	Enables faster generation of dark counts *Dark Counts*
-FXT	Enables faster generation of optical crosstalk *Optical Crosstalk*
-SIG	Enables computation of each single signal (slower)
-j <N>	Set the number of parallel workers to N (default is the number of cores detected) -d <dev> If using -SIG option this can select the device to use *cpu* or *gpu*
-f <file.txt>	Reads the configuration file selected
-w <file.root>	Writes the results on a rootfile.
-W <file>	Saves each digitized wave in a group of a hdf5 file.

CHAPTER

FIVE

VARIABLES

The variables needed to describe the SiPM and the simulation behavior are global variables and are declared in the *variables.py* file.

Here is the detailed description of each variable:

SIZE : int
The SiPM lateral size given in millimeters. This value is used to calculate the total number of cells.

CELLSIZE : int
The pitch of a single SiPM cell in micrometers. This value is used to calculate the total number of cells.

After converting these two variables both in mm the total number of cells considered in the simulation is then given by:

$$N_{cell} = \left(\frac{size}{cellsize}\right)^2$$

Note: On a real SiPM the number of cells is usually a bit smaller than the one calculated via geometrical considerations since some area of the sensor is used for the electrical connections. Thou the difference is small and negligible.

DCR : double
The dark count rate in kHz of the SiPM. This value must be greater than 0, otherwise it will throw a Division by Zero error. To turn off dcr use the option `-NDCR` as explained in *Command line options* . For a detailed description of dark counts generation see *Dark Counts*

XT : double
The optical crosstalk probability expressed in the range 0-1. 0 means no crosstalk events. For a detailed description of optical crosstalk generation see *Optical Crosstalk*

AP : double
The afterpulsing probability expressed in the range 0-1. 0 means no afterpulses events. For a detailed description of afterpulses generation see *Afterpulsing*

TRISE : double
The time constant of the rising edge of the signal shape in nanoseconds. For a detailed description of signal generation see *Signal Shaping*

TFALL : double
The time constant of the rising edge of the signal shape in nanoseconds.

CELLRECOVERY : double
The recovery time of the single SiPM cell in nanoseconds. This value is used to calculate the signal height in case a cell is hitted multiple times. A detailed description of this process can be found at *Cell Recovery*.

TAUAPFAST : double
The time constant in nanoseconds of the distribution of the fast component of afterpulses delays from their main signal.

TAUAPSLOW : double
The time constant in nanoseconds of the distribution of the slow component of afterpulses delays from their main signal.

CCGV : double
The spread of the peak height of the single cell signal. Its value is the σ of the gaussian distribution of the peak height.

SNR : double
The signal to noise ratio of the noise generated. As signal it is considered the mean value of the first peak in the multi-photon peak spectrum and as noise it is considered the σ "zero peak". To generate the white gaussian noise this value is converted in sigma:

$$\sigma_{noise} = \sqrt{10^{-SNR/20}}$$

BASESPREAD : double
The spread of the baseline value of the signal in units of σ. This is currently set to 0 because the baseline subtraction is turned off too.

SIGLEN : int
The length in nanoseconds of the signal to generate.

SAMPLING : double
The sampling time intended as time granularity of the signal to generate. This means that two consecutive point in the signal are separated by a $\Delta t =$ sampling.

Important: Keep in mind that the total number of samples per signal is given by $siglen/sampling$ so generating signals with many points after the region of interest or with too small sampling time may slow down a lot the computation of the signals.

INTSTART : double
Starting time of the integration gate in nanoseconds. Must be greater than 0 and smaller than the signal length. For more details on signal analysis see *Signal Analysis*

INTGATE : double
Lenght of the integration gate in nanoseconds. Must be greater than 0 and smaller than $siglen - intstart$.

PREGATE : double
Lenght of the pregate in nanoseconds. The pregate is placed before the intstart. This feature is currently being unused since the baseline subtraction is turned off.

FASTDCR : bool
If true enables faster generation of dark count events.

> **Warning:** This feature is still experimental. The results given by activating this option are acceptable but still to discuss. Use this feature just for debugging purposes and not for actual data production.

FASTXT : bool
If true enables faster generation of optical crosstalk events.

Warning: This feature is still experimental. The results given by activating this option are acceptable but still to discuss. Use this feature just for debugging purposes and not for actual data production.

CHAPTER

SIX

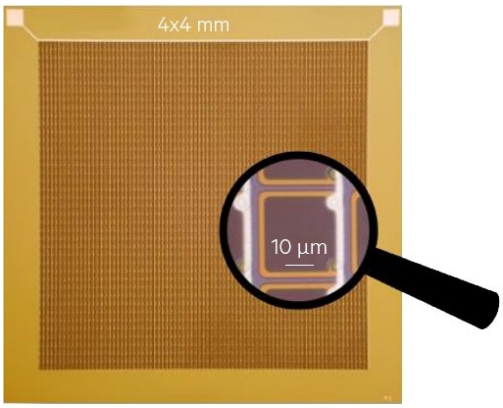
THEORY

In this section the physical background of the phenomena related to SiPM signal generation are explained along with the algorithm used to describe them.

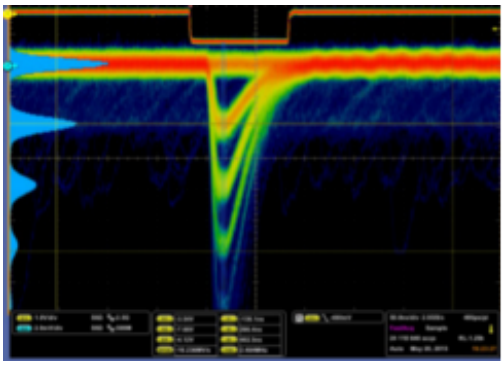
6.1 SiPM

SiPMs are state of the art light detectors with single photon sensitivity and photon counting capabilities.

A SiPM is composed by a dense matrix of Single Photon Avalanche Photo Diodes (SPADs) operated in Geiger-Muller mode. The impact of a photon on the single APD of the SiPM produces an electron - hole pair that, due to the intense electric field, triggers an avalanche of charge carriers that can be collected and measured.



Each single cell of the SiPM provides a binary information (hitted or not hitted). Since all APDs are almost identical they produce the same signal amplitude when a photon is detected, so by summing up the signals of each cell it is possible to reconstruct the total number of cells that have produced an avalanche. Thus the signal amplitude of the SiPM is expected to be proportional to the number of triggered cells and so to the number of impinging photons.



6.2 Cell Recovery

6.3 Dark Counts

6.4 Optical Crosstalk

6.5 Afterpulsing

6.6 Signal Shaping

6.7 Signal Analysis

CHAPTER

SEVEN

INDICES AND TABLES

- genindex

INDEX

A

addDCR() (*in module libs.lib*), 7

I

initializeRandomPool() (*in module libs.lib*), 8

P

PulseCPU() (*in module libs.lib*), 7
PulseCPU() (*in module libs.libCPU*), 8

S

sigPlot() (*in module libs.lib*), 8
SiPM() (*in module main*), 5
SiPMEventAction() (*in module libs.lib*), 7
SiPMSignalAction() (*in module libs.lib*), 7
SiPMSignalAction() (*in module libs.libCPU*), 8
somestats() (*in module libs.lib*), 8