calib

Release 0.8

Juan Andrés Bozzo

Aug 07, 2019

CONTENTS:

1 *gen* Module 1

2 Indices and tables 7

Python Module Index 9

Index 11

CHAPTER

ONE

GEN MODULE

This module contain classes and functions for GENerating data about the adc and the calibration conditions

class calib.gen.**PipeParameters**(*stages*, *tail*)

This class just holds multiple stages together and the thresholds of the final sub-ADC that converts the last stage residual.

property stages

The stages that make this adc.

Type data.tuple_of(StageParameters)

property tail

The thresholds of the final sub-ADC that converts the last stage's residual.

Type numpy.array

class calib.gen.**StageMeta**(*n_bits*, *n_refs*, *eff=0.995*, *cap=1*, *fsr=(-0.5*, *0.5)*, *differential=False*, *seed=None*, *half_bit=None*, *data_location=None*)

This class holds the metadata for a pipeline stage. The metadata is comprised of the design constraints that generate the ADC, these are: number of bits, number of references per capacitor and full scale range (FSR). The rest of the metadata is either deduced from that (LSB, number of capacitors, ...) or is arbitrary (capacitor magnitude, charge transfer efficiency, ...)

property cap

Ideal value of a capacitor.

property differential

True if the stage have a differential architecture.

property eff

Ideal of the charge transfer efficiency.

property fsr

Full scale range (FSR). It's a tuple with the minimum and maximum voltages the stage can handle without saturation.

generate_gaussian(*s_eff=0*, *s_cap=0*, *s_refs=0*, *s_thres=0*)

Instances a variation of the stage, with mostly gaussian perturbations.

Parameters

- **s_eff** (*numpy.array*) – Standard deviation of the charge transfer efficiency. The standard deviation is expressed in time constants as explained in *eff_random()*.
- **s_cap** (*numpy.array*) – Standard deviation of capacitors, in farads.
- **s_refs** (*numpy.array*) – Standard deviation of references, in volts.
- **s_thres** (*numpy.array*) – Standard deviation of sub-ADC thresholds, in volts.

Returns An ideal stage instance based on this metadata

Return type *StageParameters*

See also:

eff_random(), for information about the random distribution of the charge transfer efficiency parameter.

generate_ideal()
Instances an ideal stage.

Returns An ideal stage instance based on this metadata.

Return type *StageParameters*

property half_bit
Boolean indicating if the stage is half_bit. If n_bits is N and half_bit is True, then the stage is N-5 bits, otherwise is N bits.

property lsb
Least significant bit (LSB) value, given the number of references per capacitor, stage bits and FSR of this adc.

property n_bits
Integer number of bits of this stage.

See also:

half_bit, for information about half-bit stages.

property n_caps
Total number of capacitors in the stage.

property n_codes
Number of codes this stage handles (the sub-ADC and MDAC)

property n_refs
Number of references each capacitor can access.

property seed
Seed for generating random stage instances.

class calib.gen.**StageParameters**(*meta*, *eff*, *caps*, *refs*, *thres*, *data_location=None*)
This class holds the instanced data for a stage. The interpretation of this class is double. If interpreted as a pipeline stage, the thresholds represent the sub-ADC of this stage. It can be also used to represent a delta-sigma loop, with thresholds representing the sub-ADC of the next stage or the last sub-ADC in a pipeline converter.

property caps
The capacitors value, in farads.

Type numpy.ndarray

property caps_ext
The capacitors value, in farads with a last dummy capacitors of 0 farads. Useful for indexing null capacitors.

Type numpy.ndarray

property eff
The charge transfer efficiency parameter.

Type numpy.ndarray

property meta
The metadata of this stage.

Type *StageMeta*

property refs
The references value for each capacitor, in volts.

Type numpy.ndarray

property refs_ext
The references value for each capacitor, in volts, with a last dummy reference of 0 volts, useful for indexing null references.

Type numpy.ndarray

property thres
The threshold values, in volts. Depending on the interpretation, those are associated with the stage sub-ADC (pipeline) or the next stage sub-ADC (delta-sigma).

Type numpy.ndarray

calib.gen.**adjust_map**(*map_*, *n_codes*)
Expands a pipeline map to work with an sub-ADC with the same or more codes than the original sub-ADC.

Parameters

- **map** (numpy.ndarray) – The pipeline map.
- **n_codes** (*int*) – The number of codes the sub-ADC support.

Returns The expanded pipeline map.

Return type numpy.ndarray

Raises **AssertionError** – If the map is bigger than the target number of codes.

See also:

pipe_map()

calib.gen.**compute_lsb**(*n_bits*, *fsr_min*, *fsr_max*, *n_refs*, *half_bit=None*)
Computes the least significant bit (LSB) magnitude in the stage MDAC and sub-ADC to achieve a desired full scale range (FSR). The input FSR and output FSR are assumed to be the same so only one value is returned.

Parameters

- **n_bits** (*str, float, int*) – Number of bits of the stage
- **fsr_min** (*float*) – Minimum value of the full voltage scale
- **fsr_min** – Maximum value of the full voltage scale
- **n_refs** (*int*) – Number of references each active capacitor can use.
- **half_bit** (*bool*) – If the number of bits is half-bit or not

Returns The voltage value of the LSB.

Return type float

See also:

parse_bits(), for n_bits and half_bit specification.

calib.gen.**compute_n_caps**(*n_bits*, *n_refs*, *half_bit=None*)
Computes the number of capacitors a stage needs to achieve the desired resolution.

Parameters

- **n_bits** (*str, float, int*) – Number of bits of the stage
- **n_refs** – Number of references each active capacitor can use.
- **half_bit** (*bool*) – If the number of bits is half-bit or not

Returns The number of capacitors the stage needs.

Return type int

Raises **AssertionError** – If the number of references is invalid for the number of bits.

See also:

parse_bits(), for n_bits and half_bit specification.

calib.gen.**compute_n_codes**(*n_bits*, *half_bit=None*)
Computes the number of codes a pipeline stage can process.

Parameters

- **n_bits** (*str, float, int*) – Number of bits of the stage
- **half_bit** (*bool*) – If the number of bits is half-bit or not

Returns The number of codes the pipeline stage can handle

Return type int

See also:

parse_bits(), for n_bits and half_bit specification.

calib.gen.**compute_ref**(*n_bits*, *fsr_min*, *fsr_max*, *n_refs*, *half_bit=None*)
Computes the capacitor references votage for a pipeline stage.

Parameters

- **n_bits** (*str, float, int*) – Number of bits of the stage
- **fsr_min** (*float*) – Minimum value of the full voltage scale
- **fsr_min** – Maximum value of the full voltage scale
- **n_refs** (*int*) – Number of references each active capacitor can use.
- **half_bit** (*bool*) – If the number of bits is half-bit or not

Returns The values of the active capacitor references.

Return type numpy.ndarray

See also:

parse_bits(), for n_bits and half_bit specification.

calib.gen.**compute_thres**(*n_bits*, *fsr_min*, *fsr_max*, *n_refs*, *half_bit=None*)
Computes the threshold voltages for a flash sub-ADC of a pipeline stage.

Parameters

- **n_bits** (*str, float, int*) – Number of bits of the stage
- **fsr_min** (*float*) – Minimum value of the full voltage scale
- **fsr_min** – Maximum value of the full voltage scale
- **n_refs** (*int*) – Number of references each active capacitor can use.

- **half_bit** (*bool*) – If the number of bits is half-bit or not

Returns The values of the theshold voltages.

Return type numpy.ndarray

See also:

parse_bits(), for n_bits and half_bit specification.

calib.gen.**eff_random**(*eff_mean*, *tau_std*)

Generates random number following a lognormal distribution, appropiate for the charge efficiency transfer parameter (eff). The lognormal distribution has mean in the mean parameter, but the standard deviation is computed by proyecting the mean in logspace, generating a normal distribution with that standard deviation and then exponeating again. This emulates the linear settling time given by equation:

$$eff = e^{\frac{-1}{2\pi\tau}}$$

Parameters

- **eff_mean** (numpy.ndarray) – Mean of the distribution. A factor between 0 and 1.
- **tau_std** (numpy.ndarray) – Standard deviation in logspace, in time constants units.

Returns The random numbers following the lognormal distribution. The shape is the shape of the mutual boradcast of eff_mean and tau_std.

Return type numpy.ndarray

calib.gen.**format_bits**(*n_bits*, *half_bit=None*)

Converts from int + bool representation to str representation

Parameters

- **n_bits** (*str, float, int*) – Number of bits
- **half_bit** (*bool*) – If the representation is half bit or not.

Returns The str representation of n_bits

Return type str

See also:

parse_bits(), for n_bits and half_bit specification.

calib.gen.**parse_bits**(*n_bits*, *half_bit=None*)

Recieves either string, float or integer representations of a number of bits and returns the int + bool respresentation.

Parameters

- **n_bits** (*str, float, int*) – Number of bits
- **half_bit** (*bool*) – If the representation is half bit or not.

Returns The int + bool representation of n_bits

Return type (int, bool,)

Raises **ValueError** – If the type of n_bits is not suported or n_bits and half_bit conflict with each other (eg: "2.5" and False)

calib.gen.**pipe_map**(*n_caps*, *n_refs*)

Creates the code->reference map for pipeline operation. Each column in this map contains the reference index to be used for the active capacitors.

Parameters

- **n_caps** (*int*) – Number of capacitors with active references, usually the number of stage capacitors -1.
- **n_refs** (*int*) – Number of references per capacitor.

Returns The map with shape `(n_caps, n_codes,)`

Return type `numpy.ndarray`

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

calib.gen, 1

INDEX

A

adjust_map() (*in module calib.gen*), 3

C

calib.gen (*module*), 1
cap() (*calib.gen.StageMeta property*), 1
caps() (*calib.gen.StageParameters property*), 2
caps_ext() (*calib.gen.StageParameters property*), 2
compute_lsb() (*in module calib.gen*), 3
compute_n_caps() (*in module calib.gen*), 3
compute_n_codes() (*in module calib.gen*), 4
compute_ref() (*in module calib.gen*), 4
compute_thres() (*in module calib.gen*), 4

D

differential() (*calib.gen.StageMeta property*), 1

E

eff() (*calib.gen.StageMeta property*), 1
eff() (*calib.gen.StageParameters property*), 2
eff_random() (*in module calib.gen*), 5

F

format_bits() (*in module calib.gen*), 5
fsr() (*calib.gen.StageMeta property*), 1

G

generate_gaussian() (*calib.gen.StageMeta method*), 1
generate_ideal() (*calib.gen.StageMeta method*), 2

H

half_bit() (*calib.gen.StageMeta property*), 2

L

lsb() (*calib.gen.StageMeta property*), 2

M

meta() (*calib.gen.StageParameters property*), 2

N

n_bits() (*calib.gen.StageMeta property*), 2
n_caps() (*calib.gen.StageMeta property*), 2
n_codes() (*calib.gen.StageMeta property*), 2
n_refs() (*calib.gen.StageMeta property*), 2

P

parse_bits() (*in module calib.gen*), 5
pipe_map() (*in module calib.gen*), 5
PipeParameters (*class in calib.gen*), 1

R

refs() (*calib.gen.StageParameters property*), 3
refs_ext() (*calib.gen.StageParameters property*), 3

S

seed() (*calib.gen.StageMeta property*), 2
StageMeta (*class in calib.gen*), 1
StageParameters (*class in calib.gen*), 2
stages() (*calib.gen.PipeParameters property*), 1

T

tail() (*calib.gen.PipeParameters property*), 1
thres() (*calib.gen.StageParameters property*), 3