TiePieLCR

Release V1.0

Martijn Schouten

Jan 24, 2022

CONTENTS:

This documentation documenents the code of the TiePieLCR multi frequency impedance analyser. The hardware for this device can be found here

CHAPTER ONE

APP MAINWINDOW CLASS

class app.**MainWindow**(**args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QMainWindow

static acquisition_function(*LCR_settings_queue*, *displayed_error*, *shared_state*, *plot_data_queue*, *stored_data_queue*, *stored_data_requested*)

closeEvent(*self*, *QCloseEvent*)

displayed_error = <SynchronizedString wrapper for <multiprocessing.sharedctypes.c_char_Array_300 object>>

fs_changed()

get_all_stored_data()

load_settings()

lockins = 1

lockins_changed()

multiprocessing_init()

new_data = False

old_error = ''

plot_data_queue = None

reference_range = 0

save_data(*save_location*)

save_data_clicked()

save_file_dialog()

save_settings()

settings_filename = 'settings'

settings_folder = 'settings/'

settings_queue = <multiprocessing.queues.Queue object>

start_button_clicked()

start_measurement()

state = -1

state_timer()

stop_measurement()

sub_blocks_changed()

sync_settings()

tcp_daemon()

update_freq_changed()

updating_common_gui = False

updating_gui = False

app.main()

CHAPTER TWO

LOCKIN TAB CLASS

class lockin_tab.**lockin_tab**(*mainwindow*, *instance*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QWidget

amplitude_changed()

bandwidth_changed()

buffer_size = 5

build_demodulate_plots()

build_ref_fft_plot()

build_sig_fft_plot()

build_signal_plot()

color_list = ((230, 25, 75), (60, 180, 75), (255, 225, 25), (0, 130, 200), (245, 130, 48), (145, 30, 180), (70, 240, 240), (240, 50, 230), (210, 245, 60), (250, 190, 212), (0, 128, 128), (220, 190, 255), (170, 110, 40), (255, 250, 200), (128, 0, 0), (170, 255, 195), (128, 128, 0), (255, 215, 180), (0, 0, 128), (128, 128, 128), (0, 0, 0))

delete_dem_freq_pressed()

delete_gen_freq_pressed()

dem_frequency_table_changed()

dem_is_gen_changed()

dem_time_vec = []

dynamic_update_gui()

first_time_reference_changed = True

fmax_changed()

fmin_changed()

format_changed()

format_demodulate_labels()

format_ref_fft_label()

format_ref_label()

format_sig_fft_label()

format_sig_label()

gain_changed()

gen_frequency_table_changed()

gui_update_required = False

host = '127.0.0.1'

insert_dem_freq_pressed()

insert_gen_freq_pressed()

int_demodulate_1 = 0

int_demodulate_2 = 0

integration_changed()

offset_bandwidth_changed()

offset_changed()

offset_integration_changed()

optimise_pressed()

pen_ref = <PyQt5.QtGui.QPen object>

pen_sig = <PyQt5.QtGui.QPen object>

periods_changed()

plot_time_changed()

plotting_timer()

port = 65432

ref_color = (0, 0, 255)

ref_coupling_changed()

ref_range_changed()

reference_changed()

set_demodulation_freqs()

set_multisine()

set_y_range()

sig_color = (255, 0, 0)

sig_coupling_changed()

sig_range_changed()

sync_tables()

update_colors()

update_gui()

update_signal_plot_views()

CHAPTER THREE

TIEPIELCR CLASS

class TiePieLCR.**TiePieLCR**(*all_settings*, *instance*)

Bases: `object`

I2C_ADDRESS = 96

I^2C address of the TLC59116 chip in the TiePieLCR analog frontend

I2C_CARSELECT = 16

Mask for selecting the signal coming directly from the function gegenerator as a reference instead a transimpedance/charge amplifier

I2C_GAIN = 65

Mask for enabling the 50 times gain amplifier

I2C_LEDOUT0 = 20

Register that controls which of the four transimpedance/charge amplifiers is used

I2C_LEDOUT1 = 22

Register that controls which the gain and if the carrier used as a reference

I2C_MODE1 = 0

Register that is written 0, proably to put the chip in normal mode (needs verification)

I2C_NONE = 0

Mask for selecting 1x gain or none of the transimpedance amplifiers

I2C_RANGE1 = 1

Mask for enabling the first (370uA/V) transimpedance amplifier

I2C_RANGE2 = 4

Mask for enabling the second (5uA/V) transimpedance amplifier

I2C_RANGE3 = 16

Mask for enabling the third (390pC/V) transimpedance/charge amplifier

I2C_RANGE4 = 64

Mask for enabling the third (3.9pC/V) transimpedance/charge amplifier

b_bandpass = None

The fir bandpass filter coefficients that will be used during the low frequent demodulation algorithm. Master tiepieLCR only.

static bandpass_fir_filter(*x*, *filter_b*, *prev_values*)

base_freq_vector = None

A vector with the frequency of each sample in output of the fft. Master tiepieLCR only.

`base_time_vector = None`
A vector with the time at which each sample in a block of data is taken, relative to the start of the block. Master tiepieLCR only.

`block_number = 0`
Number of blocks of data that have been retrieved from the TiePie oscilloscopes since the start of this measurement

`block_number_process = 0`
Number of blocks of data that have been processed

`clipping_counter_ref = False`
How many blocks the reference should not be clipping before it will be considered good (and will become green). Master tiepieLCR only.

`clipping_counter_sig = False`
How many blocks the signal should not be clipping before it will be considered good (and will become green). Master tiepieLCR only.

`close()`

`static design_bandpass_fir_filter(`*n*, *Wn*, *width*`)`

`do_the_ffts(`*input_package*`)`

`filt_win = None`
To calculate band pass filtered signals during the low frequency algorithm the inverse fourier transform of relevant fourier coefficients is taken. However because a window was used the inverse window needs to be applied to reduce noise. This vector will contain the inverse fourier transform of the fourier coeffients window used for this frequency, which can be used to apply the inverse window. Master tiepieLCR only.

`gen = None`
A list of instances of the libtiepie Generator object. One for each tiepie. Master tiepieLCR only.

`get_data()`

`get_stored_data()`

`i2c = []`
A list of instances of the libtiepie I^2C objects. These are used to control the I2C bus that goes from the back of the tiepie to the TLC59116 chip in the TiePieLCR analog frontend. Master tiepieLCR only.

`lockins = 0`
Total number of lockins that is being used, corresponds to the number of instances of the TiePieLCR class

`static low_freq_hilbert(`*x*, *Wn*, *prev_values*`)`

`offset_bins = 20`
When the index of the lowest fft bin that contains the signal is below this value, the low freqeuncy demodulation algorithm will be used

`offset_win = None`
To calculate the offset the inverse fourier transform of the lowest frequent fourier coefficients is taken. However because a window was used the inverse window needs to be applied. This vector will contain the inverse fourier transform of lowest fourier coeffients of the window, which can be used to apply the inverse window. Master tiepieLCR only.

`old_reference_data = []`
The previous package of reference data. This data will be needed to get rid of the edge effects. Master tiepieLCR only.

old_signal_data = []
The previous package of the signal data. This data will be needed to get rid of the edge effects. Master tiepieLCR only.

open_gen()
Open the connection to arbitraty waveform generators in the TiePie's

Returns True if succesfull, False if not

Return type Boolean

open_scope()
Open the connection to the oscillscopes inside the TiePie's

Returns True if succesfull, False if not

Return type Boolean

process_data(*input_package*)

reset_buffers()

static rfft_convolute(*sig1*, *sig2*)

scp = None
The libtiepie oscilloscope object. Used to get data from the TiePie HS5's

select_LCR_gain(*instance*, *reference_setting*, *gain_setting*)

select_reference(*instance*, *reference_setting*, *gain_setting*)

serial_numbers = []
After connecting this variable will contain the serial numbers of the connected TiePie HS5's. Master tiepieLCR only.

set_settings(*all_settings: list[*TiePieLCR_settings.TiePieLCR_settings*]*, *instance*, *master_lcr:* TiePieLCR.TiePieLCR)
(re)Initialise the coefficients of the band and low pass filters as well as the processed window functions that are used for the inverse window

Parameters

- **all_settings** – A list of the TiePieLCR_settings objects that contain the settings for this lockin
- **instance** – The index in the all_settings list that belongs to this tiepieLCR's settings
- **master_lcr** – The TiePieLCR object that will be used get the data

Returns None

Return type None

settings = <TiePieLCR_settings.TiePieLCR_settings object>
An instance of the TiePieLCR class that will contain the settings of this TiePieLCR

sos_decimate = None
The filter coefficients that will be used for the final low pass filter of both algorithms. Master tiepieLCR only.

sos_offset = None
The lowpass filter coefficients that will be applied to the offset. Master tiepieLCR only.

start_awg(*instance*, *data*)

start_measurement(*all_settings*)

`start_stream`(*all_settings*)

`state = 0`

State the system. The state is only set for the tiepieLCR instance that is used for getting the data (often called masterLCR)

- state 0: Not initialised
- state 1: Initialising
- state 2: Running
- state 3: Stopping
- state 4: Updating

`stop_gen`(*inst*, *disable_output*)

`stop_measurement`()

`stop_stream`()

`update_base_vectors`()

(re)Initialise some vectors that are needed for the demodulation algorithm, but depend on the settings.

Returns None

Return type None

`update_filter_coefs`()

(re)Initialise the coefficients of the band and low pass filters as well as the processed window functions that are used for the inverse window

Returns None

Return type None

`window_cpu = None`

The window that will be applied to a block of data before applying the fft and is located in the CPU memory. Master tiepieLCR only.

`window_gpu = None`

The window that will be applied to a block of data before applying the fft and is located in the GPU memory. Master tiepieLCR only.

CHAPTER FOUR

ACQUISITION CLASS

class acquisition.**acquisition**(*LCR_settings_queue*, *displayed_error*, *shared_state*, *plot_data_queue*, *stored_data_queue*, *stored_data_requested*)

Bases: `object`

build_tiepie_list()

data_aquisition_daemon()

fft_daemon()

fft_package_list = []

package_list = []

processing_daemon()

CHAPTER

FIVE

TIEPIELCR SETTINGS CLASS

class TiePieLCR_settings.**TiePieLCR_settings**

Bases: object

Vmax = 0.8

bandwidth = 20

base_vector_update_required = False

static calculate_downsapling_rate(*n*, *Wn*)

Calculate the maximum possible downsampling rate

Returns The downsampling rate

Return type Complex double

calculate_gen_data_size()

crest_factor_cost_function(*phases*, *freqs*, *weights*, *t*, *device*)

dem_freq_update_required = False

dem_freqs = [5000]

dem_is_gen = True

dem_tiepies = [1]

demodulate_plot_points = 500

enabled = [True, True]

f_fun = 0

f_max = 0

f_min = 0

fft_plot_points = 500

fft_sensitivity = 0

fmax_plot = 1000000

fmin_plot = 100

fs_list = [6250000, 3125000, 1562500, 781250]

fs_name_list = ['6250000', '3125000', '1562500', '781250']

fs_setting = 1

gain_list = [1, 50]

`gain_name_list = ['1x', '50x']`

`gain_setting = 0`

`gain_update_required = False`

`static gcd(L)`

`gen_amplitude = 0.7`

`gen_amplitude_update_required = False`

`gen_freqs = [5000]`

`gen_offset = 0`

`gen_offset_update_required = False`

`gen_phases = [0]`

`gen_restart_required = False`

`gen_sample_freq = 700000.0`

`gen_sample_freq_max = 240000000.0`

`gen_samples_max = 1000000.0`

`gen_samples_min = 500000.0`

`gen_weights = [1.0]`

`get_block_size()`

Get number of samples in each block of data that will be retrieved from the scope

Returns The number of samples in one block

Return type Int

`get_demodulate_plot_points()`

Get the number of points plotted in the bottom two plots. Setting this to a to high value will make things slow, settings this to a too high value will cause aliasing.

Returns The number of points

Return type Float

`get_demodulation_bandwidth()`

`get_demodulation_freqs()`

Get a list with the frequencies at which the measured singal is demodulated.

Returns The frequencies

Return type list of floats

`get_demodulation_tiepies()`

Get a list with tiepie the reference signal of a specific frequency can be found.

Returns The tiepie

Return type list of ints

`get_demodulation_time_vector(blocks)`

Calculate the time vector for a certain number of block of demodulation data.

Parameters **blocks** (*Int*) – The number of blocks of scope data for which a time vector should be calcualted

Returns The calculated time vector

Return type Numpy vector

`get_df()`
Get number of frequencies in one bin of the fft

Returns The number of frequencies in one bin

Return type Float

`get_f_fun()`
Calculate frequency with which the multisine will repeat itself.

Returns The fundamental frequency

Return type float

`get_f_max()`
The maximum frequency in the multisine

Returns The maximum frequency

Return type float

`get_f_min()`
The minimum frequency in the multisine

Returns The minimum frequency

Return type float

`get_fft_sensitivity()`
Calculate the minimum width of a peak in the fft

Returns The minimum widht in Hertz

Return type Float

`get_final_offset_block_size()`
Get the number of offset samples that will result from each block after downsampling.

Returns The number of samples

Return type Int

`get_final_output_block_size()`
Get the number of impedance samples that will result from each block after downsampling.

Returns The number of samples

Return type Int

`get_gain_name_list()`
Get a list of possible gains for the instrumentation amplifier.

Returns List of gains

Return type List of floats

`get_gain_setting()`
Get the index of the currently selected gain setting. The list of possible gain can be obtained using *TiePieLCR_settings.TiePieLCR_settings.get_gain_name_list()*

Returns The index

Return type Int

`get_gain_value()`
Get the current gain of the instrumentation amplifier

Returns The gain

Return type float

`get_gen_amplitude()`
Get the maximum amplitude of the multi-sine used for the excitation. The excitation signal will determine the voltage on HcurV and when multiplied with 370uA/V also the current through HcurI. The inverse of the excitation signal will determine the voltage on nHcurV and when multiplied with 370uA/V also the current through nHcurI. This is equivalent to the excitation amplitude in the interface.

Returns The amplitude

Return type float

`get_gen_offset()`
Get the offset of the excitation. The excitation signal will determine the voltage on HcurV and when multiplied with 370uA/V also the current through HcurI. The inverse of the excitation signal will determine the voltage on nHcurV and when multiplied with 370uA/V also the current through nHcurI. This is equivalent to the excitation amplitude in the interface.

Returns The offset

Return type float

`get_gen_sample_freq()`
Calculate the sample frequency that should be used for the multisine

Returns The sample frequency

Return type Float

`get_gen_samples()`
Calculate the number of samples that should be used for the multisine

Returns The number of samples

Return type Int

`get_impedance_format()`
Get the complex impedance measured by the LCR can be represented in different ways:

- XY: As a complex and an imaginary part
- RpCp: As a capacitor and a resistor in parallel
- RsCs: As a capacitor and a reisistor in series
- ZPhi: As an absolute value and a phase

Returns The used format as a string i.e. 'XY' or 'RpCp'

Return type String

`get_impedance_format_label1()`
Get the label of the first value of the impedance format

Returns The used format as a string i.e. 'F'

Return type String

`get_impedance_format_label2()`
Get the label of the second value of the impedance format

Returns The used format as a string i.e. 'F'

Return type String

`get_impedance_format_unit1()`
Get the unit of the first value of the impedance format

Returns The used format as a string i.e. 'F'

Return type String

`get_impedance_format_unit2()`
Get the unit of the second value of the impedance format

Returns The used format as a string i.e.'F'

Return type String

`get_integration_time()`
Get the time over which the demodulation signals will be avaraged to compute the offset displayed in the interface and the offset obtain using `TiePieLCR_api.TiePieLCR_api.get_impedance()`.

Returns The amount of time

Return type Float

`get_maximum_plot_frequency()`
Get the maximum frequency that is shown in the frequency plots on the top right.

Returns The frequency

Return type float

`get_measurement_z(`*freqs*`)`
Calculate the impedance used in the feedback of the currently selected transimpedance amplifier for a specific frequency, taking into account stability caps, bias resistors and anti-aliassing filters.

Parameters `freqs` – The frequency at which the impedance should be calculated

Typ freqs float

Returns The impedance

Return type Complex double

`get_minimum_plot_frequency()`
Get the minimum frequency that is shown in the frequency plots on the top right.

Returns The frequency

Return type float

`get_multisine_crest_factor()`

`get_multisine_freqs()`
Get a list with the frequencies in the multisine that are used as an excitation signal.

Returns The frequencies

Return type list of floats

`get_multisine_phases()`
Get a list with phases of each of the frequencies in the multisine. The weights are defined relative to the output of `TiePieLCR_api.TiePieLCR_api.get_gen_amplitude()` .

Returns The phases

Return type list of floats

`get_multisine_vector()`
Get the multisine signal that the AWG in the TiePieLCR will be set to.

Returns The calculated multisine signal

Return type Numpy vector

`get_multisine_weights()`
Get a list with weights of each of the frequencies in the multisine. The weights are defined relative to the output of `TiePieLCR_api.TiePieLCR_api.get_gen_amplitude()` .

Returns The weights

Return type list of floats

`get_number_of_demodulate_freqs()`
Get the number of frequencies inside the multisine

Returns The used format as a string i.e. 'F'

Return type Int

`get_number_of_demodulation_freqs()`
The of frequencies in the multisine

Returns The number of frequencies

Return type Int

`get_number_of_multisine_freqs()`
The of frequencies in the multisine

Returns The number of frequencies

Return type Int

`get_offset_bandwidth()`
Get the bandwidth of the offset signals. The signal computed using this bandwidth can be found in the stored mat file`.

Returns The bandwidth

Return type Float

`get_offset_block_size()`
Get number of offset samples that will be calculated during each block using the ffts, before the impedance signal is downsampled.

Returns The number of offset samples per sub-block

Return type Int

`get_offset_downsampling_rate()`
Get the factor by which the offset signal calculated using the ffts will be downsampled.

Returns The downsampling ratio

Return type Int

`get_offset_integration_time()`
Get the time over which the offset signals will be avaraged to compute the offset displayed in the interface and the offset obtain using `TiePieLCR_api.TiePieLCR_api.get_impedance()`.

Returns The amount of time

Return type Int

get_offset_sample_frequency()
Get the sample frequency of the offset samples that will be calculated using the ffts, before the offset signal is downsampled.

Returns Number of offset samples per second

Return type Float

get_offset_sub_block_size()
Get number of offset samples that will be calculated during each sub-block using the ffts, before the impedance signal is downsampled.

Returns The number of offset samples per sub-block

Return type Int

get_offset_time_vector(*blocks*)
Calculate the time vector for a certain number of block of offset data.

Parameters **blocks** (*Int*) – The number of blocks of scope data for which a time vector should be calcualted

Returns The calculated time vector

Return type Numpy vector

get_output_block_size()
Get number of impedance samples that will be calculated during each block using the ffts, before the impedance signal is downsampled.

This currently is calculated incorrectly and will probably cause the gui to crash if the demodulation frequency is set too high

Returns The number of impedance samples per block

Return type Int

get_output_downsampling_rate()
Get the factor by which the impedance signal calculated using the ffts will be downsampled.

Returns The downsampling ratio

Return type Int

get_output_oversample_ratio()
Get the number of times the output signal will be oversamples. The sample frequency fill be oversample ratio times 2 times the bandwidth.

Returns The oversampling ratio of the output

Return type Int

get_output_sample_freq()
Get the sample frequency of the impedance samples that will be calculated using the ffts, before the impedance signal is downsampled.

Returns Number of impedance samples per second

Return type Float

get_output_sub_block_size()
Get number of samples of impedance samples that will be calculated during each sub-block using the ffts, before the impedance signal is downsampled.

This currently is calculated incorrectly and will probably cause the gui to crash if the demodulation frequency is set too high

Returns The number of impedance samples per sub-block

Return type Int

`get_plot_blocks()`
Get how many blocks of data should be used to plot impedance data for a period equal to the plot time.

Returns Number of blocks

Return type Int

`get_plot_periods()`
Get the number of repetitions that are shown in the time plot in the left top of the interface.

Returns The number of periods

Return type float

`get_plot_time()`
Get the amount of time shown on the x-axis of the bottom two graphs.

Returns The amount of time

Return type Float

`get_reference_gain()`
Get the gain of the transimpedance amplifier

Returns The gain of the transimpedance amplifier in A/V or C/V

Return type float

`get_reference_name_list()`
Get a list of available transimpedance amplifiers

Returns List of transimpedance amplifiers

Return type List of floats

`get_reference_offset_unit()`
Get the unit of the reference offset.

Returns A string with the abbreviation of the unit. i.e. 'V' or 'A'

Return type String

`get_reference_scope_coupling()`
Get the index of the coupling of the channel of the scope that is used to measure the reference

Returns Index in scope_coupling_name_list

Return type Int

`get_reference_scope_range_index()`
Get which element of the scope range list is selected for the reference channel

Returns The current index

Return type Int

`get_reference_scope_range_value()`
Get the voltage range of the channel of the scope that is used to measure the reference

Returns The range as a float. i.e. 0.2 or 2

Return type Float

`get_reference_setting()`
Get the index of the currently selected reference setting. The list of possible references can be obtained using *TiePieLCR_settings.TiePieLCR_settings.get_gain_name_list()*

Returns The index

Return type Int

`get_reference_unit()`
Get the unit of the reference signal.

Returns A string with the abbreviation of the unit. i.e. 'V', 'A' or 'C'

Return type String

`get_sample_frequency()`
Get the sample frequency at which the scope is running. A higher value results in less noise, however it might be reduced if the LCR stop and gives an error complaining that your PC is not fast enough.

Returns The sample frequency

Return type Float

`get_scope_coupling_name_list()`
Get a list with the different coupling options

Returns A list containing the coupling options

Return type Int

`get_scope_range_list()`
Get a list of the different gains of the scope that can be used.

Returns The current index

Return type Int

`get_settings_dict(`*filename*`)`
Get the settings in the object formatted as a dictionary object.

Returns A dict with the settings in this object

Return type Dictionary

`get_signal_scope_coupling()`
Get the index of the coupling of the channel of the scope that is used to measure the signal

Returns 0 for AC, 1 for DC

Return type Int

`get_signal_scope_range_index()`
Get which element of the scope range list is selected for the signal channel

Returns The current index

Return type Int

`get_signal_scope_range_value()`
Get the voltage range of the channel of the scope that is used to measure the signal

Returns The range as a float. i.e. 0.2 or 2

Return type Float

`get_sub_block_freq()`
The number of sub blocks per second that need to be processed. This equavalent to the number of FFT's that are being taken per second.

Returns The number of sub-blocks per second

Return type Int

`get_sub_block_size()`
Get the size of the blocks on which the fft is performed. Since the time it takes to do an fft increases with a power of 1.4 with the amount of points, a higher number of sub-blocks will reduce the computation load. A higher number sub-blocks also moves the frequency at which noise at the edge of each block will appear. A higher number of sub-blocks makes very lower frequency measurements more computationally intensive though.

Returns The number of samples per sub-block

Return type Int

`get_sub_blocks()`
Get in how many blocks the data retrieved from the scope is split up before doing the fft. Since the time it takes to do an fft increases with a power of 1.4 with the amount of points, a higher number of sub-blocks will reduce the computation load. A higher number sub-blocks also moves the frequency at which noise at the edge of each block will appear. A higher number of sub-blocks makes very lower frequency measurements more computationally intensive though.

Returns The number of sub-blocks

Return type Int

`get_sub_blocks_list()`
get a list of options to which the sub-blocks can be set.

Returns A list with the possible sub-block settings

Return type List of Ints

`get_time_plot_points()`
Get the number of points that are plotted in the time plots. Setting this to a to high value will make things slow, settings this to a too high value will cause aliasing.

Returns The number of points

Return type Int

`get_update_frequency()`
Get how often data is retrieved from the scope, the gui is updated and how often the value obtained using `TiePieLCR_api.TiePieLCR_api.get_impedance()` is updated. Only reduce this if your computer is really really slow..

Returns The update frequency

Return type Float

`impedance_format = 0`

`impedance_format_label1 = ['X', 'Rp', 'Rs', 'Z']`

`impedance_format_label2 = ['Y', 'Cp', 'Cs', 'Phi']`

`impedance_format_unit1 = ['', '', '', '']`

`impedance_format_unit2 = ['', 'F', 'F', 'rad']`

`impedance_formats = ['XY', 'RpCp', 'RsCs', 'ZPhi']` `inst = 1`

`integration_time = 1`

`static lcm(L)`

`load_settings(filename)`
Loads a settings file and applies the settings to this object

Parameters `filename` (*String*) – The filename

Returns True if the file exists, False otherwise

Return type Boolean

`max_sample_frequency = 6250000.0`

`multisine_update_required = False`

`offset_bandwidth = 20`

`offset_integration_time = 1`

`optimise_crest()`

`output_block_size = 0`

`output_oversample_ratio = 2`

`plot_periods = 3`

`plot_points = 500`

`plot_time = 5`

`protection_C = [0, 0, 3.9e-12, 3.9e-12, 0, 0]`

`protection_R = [0, 0, 2610.0, 2610.0, 0, 0]`

`real_time_mode = False`

`reference_C = [0, 4e-12, 4e-12, 3.9e-10, 4e-12, 4e-12, 0]`

`reference_R = [1, 2610.0, 200000.0, 90000000.0, 10000000000.0, 2700.0, 1]`

`reference_gain_list = [1, -0.0003831417624521073, -5e-06, -3.9e-10, -3.9e-12, 0.0003831417624521073, 1]`

`reference_name_list = ['None', 'Lcur 370 A/V', 'Lcur 5 A/V', 'Lcur 390 pC/V', 'Lcur 3.9 pC/V', 'HcurI', 'HcurV']`

`reference_offset_unit_list = ['', 'A', 'A', 'A', 'A', 'A', 'V']`

`reference_setting = 1`

`reference_unit_list = ['', 'A', 'A', 'C', 'C', 'A', 'V']`

`reference_update_required = False`

`reset()`
When the settings are loaded in the TiePieLCR parts interface of the interface might be reset. By running this function before any changes are made to the settings object, only the parts that really need to be reset are reset.

Returns Nothing

Return type None

`restart_required = False`

sample_frequency = 6250000.0

save_memory = 10000000.0

save_settings(*filename*)

Saves a settings file and with the settings of this object

Parameters **filename** (*String*) – The filename

Returns Nothing

Return type none

scope_auto_ranging = [True, True]

scope_coupling_name_list = ['AC', 'DC']

scope_couplings = [1, 1]

scope_couplings_list = [2, 1]

scope_range_list = [0.2, 0.4, 0.8, 2, 4]

scope_range_name_list = ['0.2', '0.4', '0.8', '2', '4']

scope_ranges = [2, 2]

set_LCR_gain(*new_gain*)

Set the instrumentation amplifier gain to one of the options in the list of possible gains. The list of possible gain can be obtained using `TiePieLCR_settings.TiePieLCR_settings.get_gain_name_list()`

Parameters **new_gain** (*Int*) – The index in the list

Returns Nothing

Return type None

set_amplitude(*amplitude*)

Sets the maximum amplitude of the multisine that is used as an excitation signal.

Parameters **amplitude** (*Float*) – The maximum amplitude

Returns False if the total excitation signals get's too large, True otherwise

Return type Boolean

set_demodulation_bandwidth(*bandwidth*)

Set the bandwidth with which the current and the voltage are demodulated and how fast the bottom two graphs in the interface will respond.

Parameters **bandwidth** (*Float*) – The bandwidth

Returns A dictionary that contains the keyword 'error' with an error when an error occured and is empty otherwise

Return type Dict

set_demodulation_params(*freq_list*, *tiepie_list*)

set_impedance_format(*value*)

Set the complex impedance measured by the LCR can be represented in different ways:

- XY: As a complex and an imaginary part
- RpCp: As a capacitor and a resistor in parallel
- RsCs: As a capacitor and a reisistor in series
- ZPhi: As an absolute value and a phase

Parameters **`value`** (*String*) – The used format as a string i.e. 'XY' or 'RpCp'

Returns Nothing

Return type None

`set_integration_time`(*integration_time*)
Set the time over which the demodulation signals will be avaraged to compute the offset displayed in the interface and the offset obtain using `TiePieLCR_api.TiePieLCR_api.get_impedance()`.

Parameters **`integration_time`** (*Float*) – The amount of time

Returns Nothing

Return type None

`set_maximum_plot_frequency`(*frequency*)
Set the minimum frequency that is shown in the frequency plots on the top right.

Returns True if larger as the current minimum frequency, False otherwise.

Return type Boolean

`set_minimum_plot_frequency`(*frequency*)
Set the minimum frequency that is shown in the frequency plots on the top right.

Returns True if smaller as the current maximum frequency, False otherwise.

Return type Boolean

`set_multisine`(*freq_list*, *weight_list*)
Sets the frequencies and their weights in the multisine that is used as an excitation signal. The weights are defined relative to the maximum amplitude of the signal.

Parameters

- **`freq_list`** (*List of floats*) – A list of frequencies to be used in the multisine
- **`weight_list`** (*List of weights*) – A list of weight to be used in the multisine. The weights are relative to the maximum amplitude of the signal as defined by *`TiePieLCR_settings.TiePieLCR_settings.set_amplitude()`*.

Returns The sample frequency

Return type Float

`set_offset`(*offset*)
Sets the offset of the multisine that is used as an excitation signal.

Parameters **`amplitude`** (*Float*) – The offset

Returns False if the total excitation signals get's too large, True otherwise

Return type Boolean

`set_offset_bandwidth`(*new_offset_bandwidth*)
Set the bandwidth with which the offset is calculated and what will be the bandwidth of the offset signals in the mat file.

Parameters **`bandwidth`** (*Float*) – The bandwidth

Returns Nothing

Return type None

`set_offset_integration_time`(*new_offset_integration*)
Over how much time the the offset signals are averaged to come to the displayed impedances.

Parameters **bandwidth** (*Float*) – The amount of time

Returns Nothing

Return type None

set_plot_periods(*periods*)
Set the number of repetitions that are shown in the time plot in the left top of the interface.

Parameters **periods** (*float*) – The number of periods

Returns Nothing

Return type None

set_plot_time(*plot_time*)
Set the amount of time shown on the x-axis of the bottom two graphs.

Parameters **plot_time** (*Float*) – The amount of time

Returns Nothing

Return type None

set_reference(*new_reference*)
Set the transimpedance amplifier to one of the options in the list of possible options. The list of possible transimpedance amplifiers can be obtained using *TiePieLCR_settings.TiePieLCR_settings.get_reference_name_list()*

Parameters **new_gain** (*Int*) – The index in the list

Returns Nothing

Return type None

set_reference_scope_coupling(*coupling*)
Set the index of the coupling of the channel of the scope that is used to measure the reference

Parameters **coupling** (*Int*) – Index in the list returned by *TiePieLCR_settings.TiePieLCR_settings.get_scope_coupling_name_list()*

Returns Nothing

Return type none

set_reference_scope_range(*range*)
Set which element of the scope range list is selected for the reference channel. The scope range list can be obtained using *TiePieLCR_settings.TiePieLCR_settings.get_scope_range_list()*.

Returns Nothing

Return type None

set_sample_frequency(*value*)
Set the sampling frequency of the scope. A higher value results in less noise, however it might be reduced if the LCR stop and gives an error complaining that your PC is not fast enough.

The sampling frequency should be smaller as 6250000 Hz and be an integer multiple of the number of sub blocks.

Returns A dictionary that contains the keyword 'error' when an error occured and is empty otherwise

Return type Dict

set_signal_scope_coupling(*coupling*)
Set the index of the coupling of the channel of the scope that is used to measure the signal

Parameters **coupling** (*Int*) – Index in the list returned by *TiePieLCR_settings.TiePieLCR_settings.get_scope_coupling_name_list()*

Returns Nothing

Return type none

set_signal_scope_range(*range*)

Set which element of the scope range list is selected for the signal channel. The scope range list can be obtained using *TiePieLCR_settings.TiePieLCR_settings.get_scope_range_list()*.

Returns Nothing

Return type None

set_sub_blocks(*sub_block_index*)

Sets in how many blocks the data retrieved from the scope is split up before doing the fft. Since the time it takes to do an fft increases with a power of 1.4 with the amount of points, a higher number of sub-blocks will reduce the computation load. A higher number sub-blocks also moves the frequency at which noise at the edge of each block will appear. A higher number of sub-blocks makes very lower frequency measurements more computationally intensive though.

Parameters **sub_block_index** (*Int*) – Number in the list obtained through *TiePieLCR_settings.TiePieLCR_settings.get_sub_blocks_list()* that should be used

Returns A dictionary that contains the keyword 'error' with an error when an error occured and is empty otherwise

Return type Dict

set_update_freq(*update_freq_index*)

Set how often data is retrieved from the scope, the gui is updated and how often the value obtained using `TiePieLCR_api.TiePieLCR_api.get_impedance()` is updated. Only reduce this if your computer is really really slow..

The update sample frequency divided by the update frequency should be an integer multiple of the number of sub blocks.

Parameters **update_freq_index** (*Int*) – Frequency in the list obtained through `TiePieLCR_settings.TiePieLCR_settings.get_update_freq_list()` that should be used

Returns A dictionary that contains the keyword 'error' with an error when an error occured and is empty otherwise

Return type Dict

settings_dict = {}

side_lob_n = 7

sub_blocks_list = [1, 2, 5, 10, 20, 50]

sub_blocks_name_list = ['1', '2', '5', '10', '20', '50']

sub_blocks_setting = 1

update_freq_list = [1, 2, 5, 10, 25]

update_freq_name_list = ['1 Hz', '2 Hz', '5 Hz', '10 Hz', '25 Hz']

update_freq_setting = 3

version = 'V1.0.0'

CHAPTER

SIX

IMPEDANCE CLASS

class impedance.**impedance**

Bases: `object`

error1 = 0

error2 = 0

ref_clipped = False

ref_offset = 0

set_clipping(*ref_clipping*, *sig_clipping*)

Set wether or not the reference and signal clipped (=reached the maximum voltage that can be measured) during the integration time.

Parameters

- **ref_clipping** (*Boolean*) – The reference clipped
- **sig_clipping** (*Booelan*) – The signal clipped

Returns Nothing

Return type None

set_errors(*new_error1*, *new_error2*)

Set the impedance values of the object. What these values represents is determined by `tiepieLCR_settings.tiepieLCR_settings.get_impedance_format()`

Parameters

- **new_error1** (*Complex double*) – The standard error in the first impedance value
- **new_error2** (*Complex double*) – The standard error in the second impedance value

Returns Nothing

Return type None

set_offsets(*new_ref_offset*, *new_sig_offset*)

Set the offset values of the object.

Parameters

- **new_ref_offset** (*Double*) – The offset in the reference
- **new_sig_offset** (*Double*) – The offset in the signal

Returns Nothing

Return type None

`set_timestamp`(*stamp*)
set the timestamp of the point in time the impedance was measured. This always is the timestamp of the last sample of the integration time.

Parameters **`stamp`** (*`Float`*) – The timestamp

Returns Nothing

Return type None

`set_values`(*new_value1*, *new_value2*)
Set the impedance values of the object. What these values represents is determined by `tiepieLCR_settings.tiepieLCR_settings.get_impedance_format()`

Parameters

- **`new_value1`** (*`Complex double`*) – The first impedance value.
- **`new_value2`** (*`Complex double`*) – The second impedance value.

Returns Nothing

Return type None

`sig_clipped = False`

`sig_offset = 0`

`timestamp = 0`

`valid = False`

`value1 = 0`

`value2 = 0`

PYTHON MODULE INDEX