fmcw Documentation

Alexandre Bondoux

Oct 18, 2019

Contents:

1 Table Of Contents: **3**

1.1 History of the project 3

1.2 fmcw package 3

2 Indices and tables **15**

Python Module Index **17**

Index **19**

/!DOCUMENTATION STILL UNDER CONSTRUCTION /!

This library provides high level access to the FMCW3 radar. It provides the necessary building blocks to automatically handle configuration of the FPGA, reception of the data and post-processing. With reasonable settings, most users should even be able to be able to display a few plots in real time.

If you have any questions, feel free to join the fmcwworkspace slack.

CHAPTER 1

Table Of Contents:

1.1 History of the project

The fmcw project started a few years ago as Henrik Forsten attempted to build a radar with readily accessible components. His blog is the best source of technical information on the hardware.

The fmcw package aims at generalizing the python code he open sourced on GitHub and provide the user with a lot more features. By lowering the threshold required to get into building a radar, the hope is to grow a community of passionate radar makers.

1.2 fmcw package

Here is the documentation of all the modules in the fmcw package.

1.2.1 fmcw.adc module

class fmcw.adc.**ADF4158**
Bases: object

find_reg(*reg*)
Finds register by name.

Parameters **reg** –

Returns

freq_to_regs(*fstart*, *fpd_freq*, *bw*, *length*, *delay*)
Set up the ADC.

Parameters

- **fstart** – Initial chirp frequency?
- **fpd_freq** –

- **bw** – [Hz] bandwidth
- **length** –
- **delay** – Delay to account for the start of the mixer.

Returns

write_value(***kw*)
Write value to register, doesn't update the device.

Parameters **kw** –

Returns

1.2.2 fmcw.camera module

class fmcw.camera.**camera**(*flag_camera_ready*, *flag_reading_data*, *s*)
Bases: multiprocessing.context.Process

This class describes the object that will read the camera as a separate subprocess. The start/stop is regulated by an Event object.

run()
Yes, importing all these modules from within a function is gettho, but I have not found a way around it yet. import cv2 will start displaying the feed with Qt5 (my default), which then appears to be "taken" for all the other plots being display. Therefore, the next backend is used for the other plots - TkAgg - and my package was not written to support something else than TkAgg.

Returns

1.2.3 fmcw.display module

fmcw.display.**import_csv**(*path*, *timestamp*, *s*)

fmcw.display.**import_settings**(*path*, *timestamp*)

fmcw.display.**plot_angle**(*t*, *d*, *fxdb*, *angles_masked*, *clim*, *max_range*, *time_stamp*, *method=''*, *show_plot=False*)
TO DO: ACTUALIZE THE ARGUMENTS WITH THE NEW METHOD Plot the angular data for a bunch of sweeps.

Parameters

- **t** –
- **d** –
- **fxdb** –
- **angles_masked** –
- **clim** –
- **max_range** –
- **time_stamp** –
- **method** –
- **show_plot** –

Returns

fmcw.display.**plot_if_spectrum**(*d*, *ch*, *sweep_number*, *w*, *fir_gain*, *adc_bits*, *time_stamp*, *show_plot=False*)

TO DO: NEEDS AN UPDATE TO REDUCE THE ARGUMENT COUNT VIA THE USE OF THE SETTINGS DICTIONARY Plot the IF Fourier spectrum. Useful to see how much noise there is in the data and where.

Parameters

- **d** – [m] distance bins TO DO: why is it here?
- **ch** – Dictionary containing the data
- **sweep_number** – sweep to plot
- **w** – Window to use when processing the data
- **fir_gain** – TO DO: will be superseeded by the settings dictionary
- **adc_bits** – TO DO: will be superseeded by the settings dictionary
- **time_stamp** – [bool] Save the data?
- **show_plot** – [bool] Show the plot?

Returns

fmcw.display.**plot_if_time_domain**(*fig_if*, *t*, *ch*, *sweep_number*, *s*, *ylim*, *time_stamp*, *show_plot=False*)

TO DO: ACTUALIZE THE ARGUMENTS WITH THE NEW METHOD Plot the IF data for a bunch of sweeps.

Parameters

- **fig_if** –
- **t** –
- **ch** –
- **sweep_number** –
- **s** –
- **ylim** –
- **time_stamp** –
- **show_plot** –

Returns

fmcw.display.**plot_range_time**(*t*, *im*, *s*, *time_stamp=''*)

TO DO: ACTUALIZE THE ARGUMENTS WITH THE NEW METHOD Range time plot of a bunch of sweeps.

Parameters

- **t** –
- **meshgrid_data** –
- **m** –
- **time_stamp** –
- **show_plot** –

Returns

1.2.4 fmcw.ftdi module

class fmcw.ftdi.**FPGA**(*ADC*, *encoding='latin1'*)
Bases: object

Creates the FTDI object that handles the communication with the FPGA.

clear_adc(*oe1=False*, *oe2=False*, *shdn1=False*, *shdn2=False*)
Create a packet signaling the ADC pins to clear on the FPGA.

Parameters

- **oe1** – Clear Output Enable 1
- **oe2** – Clear Output Enable 2
- **shdn1** – Clear Shutdown 1
- **shdn2** – Clear Shutdown 2

Returns Packet to be encapsulated

clear_buffer()
Clear some buffer.

Returns Packet to be encapsulated

clear_gpio(*led=False*, *pa_off=False*, *mix_enbl=False*, *adf_ce=False*)
Create a packet signaling the GPIO pins to clear on the FPGA.

Parameters

- **led** – Clear the LED
- **pa_off** – Clear pa_off
- **mix_enbl** – Disable mixer
- **adf_ce** – Clear the adf_ce

Returns Packet to be encapsulated

close()
Close the FTDI object.

Returns

send_packet(*x*, *cmd*)
Add a header to a packet, encode it and write to FTDI.

Parameters

- **x** – data
- **cmd** – type of packet

Returns write to FTDI

set_adc(*oe1=False*, *oe2=False*, *shdn1=False*, *shdn2=False*)
Create a packet signaling the ADC pins to set on the FPGA.

Parameters

- **oe1** – Set Output Enable 1
- **oe2** – Set Output Enable 2
- **shdn1** – Set Shutdown 1

- **shdn2** – Set Shutdown 2

Returns Packet to be encapsulated

set_channels(*a=True*, *b=True*)
WARNING: ONLY 2 CHANNELS SUPPORTED Set the channels to be activated (only two supported).

Parameters

- **a** – State of channel a
- **b** – State of channel b

Returns Packet to be encapsulated

set_downsampler(*enable=True*, *quarter=False*)
WARNING: THE FPGA CODE REQUIRES IT TO BE ENABLED. TO DO: ALLOW THE USER TO DEACTIVATE IT Set the downsampler.

Parameters

- **enable** – Turn it on
- **quarter** – Divide the sampling rate by another factor of 2

Returns Packet to be encapsulated

set_gpio(*led=False*, *pa_off=False*, *mix_enbl=False*, *adf_ce=False*)
Create a packet signaling the GPIO pins to set on the FPGA.

Parameters

- **led** – Set the LED
- **pa_off** – Set the pa_off
- **mix_enbl** – Enable the mixer
- **adf_ce** – Set the adf_ce

Returns Packet to be encapsulated

set_sweep(*fstart*, *bw*, *length*, *delay*)
Set sweep parameters.

Parameters

- **fstart** – [Hz] Start frequency of the chirp
- **bw** – [Hz] Bandwidth to use
- **length** – [s] Duration of the sweep
- **delay** – [s] Delay between two sweeps

Returns real delay between two sweeps (just informational)

write_decimate(*decimate*)
Create a packet to configure the decimation factor at the FPGA level.

Parameters **decimate** – Number of sweeps to skip. 0 means no sweeps are skipped.

Returns Packet to be encapsulated

write_pa_off_timer(*length*)
Convert the duration pa_off_timer to a number of ADC clock cycles.

Parameters **length** – number of clock cycles that represent the pa_off_timer

Returns Packet to be encapsulated

write_pll()
Call for the configuration of all the registers.

Returns void

write_pll_reg(*n*)
Create a packet to configure a PLL register.

Parameters **n** – Configuration parameter

Returns Packet to be encapsulated

write_sweep_delay(*length*)
Convert the duration between two sweeps (sweep delay) to a number of ADC clock cycles.

Parameters **length** – number of clock cycles that represent the duration between two sweeps

Returns Packet to be encapsulated

write_sweep_timer(*length*)
Convert the duration of the sweep to a number of ADC clock cycles.

Parameters **length** – number of clock cycles that represent the duration of a sweep

Returns Packet to be encapsulated

class fmcw.ftdi.**Writer**(*filename*, *queue*, *encoding='latin1'*, *timeout=0.5*)
Bases: threading.Thread

DEPRECATED Legacy Writer thread used to write to the binary log file.

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

1.2.5 fmcw.preprocessing module

class fmcw.preprocessing.**fpga_reader**(*flag_reading_data*, *connection*, *s*)
Bases: multiprocessing.context.Process

This class describes the object that will read the FPGA. By inheriting from Process, it can be launched as a subprocess.

close()
Close the Process object.

This method releases resources held by the Process object. It is an error to call this method if the child process is still running.

run()
Main routine polling the USB bus. It is not perfect, but >95% of the frames are valid with this configuration.

Returns void

1.2.6 fmcw.postprocessing module

class fmcw.postprocessing.**Writer**(*queue*, *s*, *encoding='latin1'*)
Bases: threading.Thread

Writer object that writes data to file. Created as a separate thread fed from a queue, so it's not blocking. Nothing special about it. Comes in twpo flavors: - Writer for binary files - Writer for csv files

run()
Process the data from the queue and write it to file.

Returns void

class fmcw.postprocessing.**angle_animation**(*tfd_angles*, *s*, *method='angle'*, *blit=False*)
Bases: object

update_plot(*fxdb*, *time_stamp*)
Dynamic refresh of the angular plot. A lot of work has been put in reducing the time necessary to refresh a plot. There must be some possible improvements, especially by messing with the backend directly. TO DO: less data points could be plotted as an entire sweep is likely to contain more points than pixels.

Parameters

- **fxdb** – Angular data to plot
- **time_stamp** – Timestamp for the current sweep

Returns

class fmcw.postprocessing.**angle_display**(*tfd_angles*, *s*, *data_accessible*, *new_sweep_angle*, *sweep_to_display*, *time_stamp*)
Bases: multiprocessing.context.Process

Sub-process to display the angular information coming from both receivers.

run()
Angle sub-process loop.

Returns

fmcw.postprocessing.**butter_highpass**(*cutoff*, *fs*, *order=4*)
User friendly wrapper for a highpass scipy.signal.butter.

Parameters

- **cutoff** – cutoff frequency
- **fs** – sampling frequency
- **order** – order of the Butterworth filter

Returns scipy butter objects

fmcw.postprocessing.**butter_highpass_filter**(*data*, *cutoff*, *fs*, *order=4*)
Filter data with a highpass scipy.signal.butter.

Parameters

- **data** – Data to filter
- **cutoff** – Cutoff frequency
- **fs** – Sampling frequency
- **order** – Order of the Butterworth filter

Returns Filtered data

fmcw.postprocessing.**calculate_angle_plot**(*sweeps*, *s*, *tfd_angles*)
Perform the data processing to calculate the angular location of objects in a single sweep. The goal is to plot that result afterward, not to process multiple sweeps.

Parameters

- **sweeps** – Data from which the angle position will be calculated
- **s** – Settings dictionary
- **tfd_angles** – Tuple containing all the bins important for the plotting

Returns fxdb

fmcw.postprocessing.**calculate_if_data**(*sweeps*, *s*)
Convert the raw data to a differential voltage level. Note that the data is cast from int16 to float64.

Parameters

- **sweeps** – Sweeps to consider
- **s** – Settings dictionary

Returns Voltage is returned as a dict with each key being a channel.

fmcw.postprocessing.**calculate_range_time**(*ch*, *s*, *single_sweep=-1*)
Take a single sweep and calculate the distances of all signals. All the channels are averaged in a single virtual channel. While this is not super good practice, it is mostly okay given how far the objects are in comparison to the distance between antennas.

Parameters

- **ch** – dict containing the sweep data for each channel
- **s** – Settings dictionary
- **single_sweep** – Sweep to select in the dictionary in case there are actually multiple of them. To be removed.

Returns im, nb_sweeps, max_range_index

fmcw.postprocessing.**compare_ndarrays**(*a*, *b*)
Check if two arrays are equivalent or not with additional details Helper function written to find quickly why two arrays are not equal element wise.

Parameters

- **a** – Array 1
- **b** – Array 2

Returns Void. An exception is raised if a difference between the two arrays have been found.

fmcw.postprocessing.**create_bases**(*s*)
Create the x axis data for all sorts of plots. This will speed up the display of the plots by caching it and limiting the amount of data to be redrawn.

Parameters **s** – Settings dictionary

Returns time, frequency, distance, angle bins

fmcw.postprocessing.**f_to_d**(*f*, *s*)
Converts frequency bins to distance bins based on ADC settings.

Parameters

- **f** – Frequency bins

- **s** – Settings dictionary

Returns Distance bins

fmcw.postprocessing.**find_start**(*f*, *start*, *s*)

Find a valid start header in a binary file by looking for two valid headers separated by the proper length of data. Given the simplicity of the system, it is not possible to guarantee that this data is "legit" as valid headers could be coming from random data. However, it is very unlikely.

Parameters

- **f** – File handle
- **start** – Start signal to look for
- **s** – Settings dictionary

Returns The current file.seek() index at which the valid data starts and the corresponding frame number. It is coded on a single byte, so expect it to roll over after 255 is reached.

fmcw.postprocessing.**find_start_batch**(*data*, *s*, *initial_index=0*)

Find the starting index of the first valid batch of sweep data and its corresponding header. :param data: Batch of data coming from the FPGA via the USB port.

Parameters

- **s** – Settings dictionary
- **initial_index** – 0 if reading a new batch, non zero if finding the next valid sweep within a batch

Returns Starting index of a sweep data, header of that sweep

class fmcw.postprocessing.**if_display**(*tfd_angles*, *s*, *data_accessible*, *new_sweep_if*, *sweep_to_display*, *time_stamp*)

Bases: multiprocessing.context.Process

Sub-process for displaying the IF (Intermediate Frequency) data. These raw values coming out of the ADC (after FPGA filtering) are (almost) what make up the sweeps. There is a little bit of post-processing but not much.

run()

IF process loop

Returns

class fmcw.postprocessing.**if_time_domain_animation**(*tfd_angles*, *s*, *grid=False*, *blit=False*)

Bases: object

update_plot(*if_data*, *time_stamp*)

Dynamic refresh of the IF plot. A lot of work has been put in reducing the time necessary to refresh a plot. There must be some possible improvements, especially by messing with the backend directly. TO DO: less data points could be plotted as an entire sweep is likely to contain more points than pixels available to it on the screen.

Parameters

- **if_data** – processed IF data from a sweep
- **time_stamp** – Timestamp for the current sweep

Returns

fmcw.postprocessing.**import_data**(*f*, *start*, *first_frame*, *s*, *samples*, *verbose=False*)

Import the data from a binary file. This was the source inspiration for process_batch, which is more up to date and deal with real time data. As a result, this might not be fully up to date.

Parameters

- **f** – File handle
- **start** – Start signal for the headers
- **first_frame** – Get the current frame number read from find_start
- **s** – Settings dictionary
- **samples** – Legacy argument, useless
- **verbose** – Print a lot more info

Returns

fmcw.postprocessing.**move_figure**(*f*, *number*)

Move a figure to position (x, y) of the screen determined by the figure "number". Only 3 positions supported. DO NOT REALY ON THIS FUNCTION. CANNOT BE GENERALIZED TO OTHER USE CASES THAN WHAT IT WAS DESIGNED FOR. Basically, only used it with Qt5Agg. Did not try other backends and the code is not complete for it. They are slower than Qt when I tried, so not relevant. All units are px.

Parameters

- **f** – Figure handle
- **number** – Figure number. Only handles 3 different positions on screen, all 3 horizontal.

Returns Void

fmcw.postprocessing.**process_batch**(*rest*, *data*, *s*, *next_header*, *counter_decimation*, *sweep_count*, *verbose=False*)

Main function to process incoming batches of data from the FPGA. The goal is to find valid sweeps in the data. Main challenges are that the start of the data might come from the end of a previous sweep, there might be some dropped byte in some sweeps due to latency from the OS vs real time FPGA, and a last sweep that is incomplete and has to be merged with the next batch.

Parameters

- **rest** – End of the previous batch that was not long enough to constitute a whole sweep.
- **data** – New batch of USB data from the FPGA
- **s** – Settings dictionary
- **next_header** – Expected header of the next sweep
- **counter_decimation** – Rolling counter, keeps track of software decimation across batches
- **sweep_count** – Global number of valid, post decimation sweeps that have been found
- **verbose** – A lot of extra info will be displayed

Returns batch_ch, next_header, rest, sweep_count, counter_decimation

fmcw.postprocessing.**r4_normalize**(*x*, *d*, *e=1.5*)

Not sure what this does. Used when processing the angle data.

Parameters

- **x** –
- **d** –

- **e** –

Returns

class fmcw.postprocessing.**range_time_animation**(*s*, *max_range_index*, *blit=False*)

Bases: object

update_plot(*im*, *time_stamp*, *sweeps_skipped*)

Dynamic refresh of the Range time plot A lot of work has been put in reducing the time necessary to refresh a plot. There must be some possible improvements, especially by messing with the backend directly. TO DO: less data points could be plotted as an entire sweep is likely to contain more points than pixels available to it on the screen.

Parameters

- **im** – range time data
- **time_stamp** – Timestamp for the current sweep
- **sweeps_skipped** – Important here to duplicate the current sweep as many times as sweeps we skipped

Returns

class fmcw.postprocessing.**range_time_display**(*tfd_angles*, *s*, *data_accessible*, *new_sweep_range_time*, *sweep_to_display*, *time_stamp*)

Bases: multiprocessing.context.Process

run()

Range time sub-process loop.

Returns

fmcw.postprocessing.**read_settings**(*f*, *encoding=None*)

Reads the first line of a file and evaluates it as python code. Used when reading the binary log as the first line contains the settings dictionary.

Parameters

- **f** – File handle
- **encoding** –

Returns Settings dictionary from string evaluated as python code

fmcw.postprocessing.**subtract_background**(*channel_data*, *w*, *data*)

DEPRECATED? Subtract the mean to a list of sweeps and multiply the result by the weights w. One thing to note, is that sweeps full of zeros (coming from corrupted usb data) are left invariant.

Parameters

- **channel_data** – dict of channels containing the sweep data as numpy arrays
- **w** – weights to apply to the array of sweeps
- **data** – Not sure

Returns Updated channel_data

fmcw.postprocessing.**subtract_clutter**(*channel_data*, *w*, *data*, *clutter_averaging=1*)

DEPRECATED? Subtract to a sweep the average of the previous clutter_averaging sweeps. It's some kind of moving average. The goal is to perform motion detection a lot more easily.

Parameters

- **channel_data** – dict of channels containing the sweep data as numpy arrays

- **w** – weights to apply to the array of sweeps
- **data** – Not sure
- **clutter_averaging** – Number of previous sweeps to average before subtracting them to the current one.

Returns

fmcw.postprocessing.**twos_comp**(*val*, *bits*)

Compute the 2's complement of int value val.

Parameters

- **val** – Bytes to complement
- **bits** –

Returns 2's complement of int value val

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

fmcw.adc, 3
fmcw.camera, 4
fmcw.display, 4
fmcw.ftdi, 6
fmcw.postprocessing, 9
fmcw.preprocessing, 8

Index

A

ADF4158 (*class in fmcw.adc*), 3
angle_animation (*class in fmcw.postprocessing*), 9
angle_display (*class in fmcw.postprocessing*), 9

B

butter_highpass() (*in module fmcw.postprocessing*), 9
butter_highpass_filter() (*in module fmcw.postprocessing*), 9

C

calculate_angle_plot() (*in module fmcw.postprocessing*), 9
calculate_if_data() (*in module fmcw.postprocessing*), 10
calculate_range_time() (*in module fmcw.postprocessing*), 10
camera (*class in fmcw.camera*), 4
clear_adc() (*fmcw.ftdi.FPGA method*), 6
clear_buffer() (*fmcw.ftdi.FPGA method*), 6
clear_gpio() (*fmcw.ftdi.FPGA method*), 6
close() (*fmcw.ftdi.FPGA method*), 6
close() (*fmcw.preprocessing.fpga_reader method*), 8
compare_ndarrays() (*in module fmcw.postprocessing*), 10
create_bases() (*in module fmcw.postprocessing*), 10

F

f_to_d() (*in module fmcw.postprocessing*), 10
find_reg() (*fmcw.adc.ADF4158 method*), 3
find_start() (*in module fmcw.postprocessing*), 11
find_start_batch() (*in module fmcw.postprocessing*), 11
fmcw.adc (*module*), 3
fmcw.camera (*module*), 4
fmcw.display (*module*), 4
fmcw.ftdi (*module*), 6
fmcw.postprocessing (*module*), 9
fmcw.preprocessing (*module*), 8
FPGA (*class in fmcw.ftdi*), 6
fpga_reader (*class in fmcw.preprocessing*), 8
freq_to_regs() (*fmcw.adc.ADF4158 method*), 3

I

if_display (*class in fmcw.postprocessing*), 11
if_time_domain_animation (*class in fmcw.postprocessing*), 11
import_csv() (*in module fmcw.display*), 4
import_data() (*in module fmcw.postprocessing*), 11
import_settings() (*in module fmcw.display*), 4

M

move_figure() (*in module fmcw.postprocessing*), 12

P

plot_angle() (*in module fmcw.display*), 4
plot_if_spectrum() (*in module fmcw.display*), 4
plot_if_time_domain() (*in module fmcw.display*), 5
plot_range_time() (*in module fmcw.display*), 5
process_batch() (*in module fmcw.postprocessing*), 12

R

r4_normalize() (*in module fmcw.postprocessing*), 12
range_time_animation (*class in fmcw.postprocessing*), 13
range_time_display (*class in fmcw.postprocessing*), 13
read_settings() (*in module fmcw.postprocessing*), 13
run() (*fmcw.camera.camera method*), 4
run() (*fmcw.ftdi.Writer method*), 8
run() (*fmcw.postprocessing.angle_display method*), 9
run() (*fmcw.postprocessing.if_display method*), 11

run() (*fmcw.postprocessing.range_time_display method*), 13
run() (*fmcw.postprocessing.Writer method*), 9
run() (*fmcw.preprocessing.fpga_reader method*), 8

S

send_packet() (*fmcw.ftdi.FPGA method*), 6
set_adc() (*fmcw.ftdi.FPGA method*), 6
set_channels() (*fmcw.ftdi.FPGA method*), 7
set_downsampler() (*fmcw.ftdi.FPGA method*), 7
set_gpio() (*fmcw.ftdi.FPGA method*), 7
set_sweep() (*fmcw.ftdi.FPGA method*), 7
subtract_background() (*in module fmcw.postprocessing*), 13
subtract_clutter() (*in module fmcw.postprocessing*), 13

T

twos_comp() (*in module fmcw.postprocessing*), 14

U

update_plot() (*fmcw.postprocessing.angle_animation method*), 9
update_plot() (*fmcw.postprocessing.if_time_domain_animation method*), 11
update_plot() (*fmcw.postprocessing.range_time_animation method*), 13

W

write_decimate() (*fmcw.ftdi.FPGA method*), 7
write_pa_off_timer() (*fmcw.ftdi.FPGA method*), 7
write_pll() (*fmcw.ftdi.FPGA method*), 8
write_pll_reg() (*fmcw.ftdi.FPGA method*), 8
write_sweep_delay() (*fmcw.ftdi.FPGA method*), 8
write_sweep_timer() (*fmcw.ftdi.FPGA method*), 8
write_value() (*fmcw.adc.ADF4158 method*), 4
Writer (*class in fmcw.ftdi*), 8
Writer (*class in fmcw.postprocessing*), 9