LWA352 SNAP2 F-Engine Documentation

Release 1.0.0

Jack Hickish

Apr 11, 2021

CONTENTS:

Contents:

CHAPTER ONE

INSTALLATION

The LWA 352 F-Engine pipeline is available at https://github.com/realtimeradio/caltech-lwa. Follow the following instructions to download and install the pipeline.

Specify the build directory by defining the `BUILDDIR` environment variable, eg:

```
export BUILDDIR=~/src/
mkdir -p $BUILDDIR
```

1.1 Get the Source Code

Clone the repository and its dependencies with:

```
# Clone the main repository
cd $BUILDDIR
git clone https://github.com/realtimeradio/caltech-lwa
# Clone relevant submodules
cd caltech-lwa
git submodule init
git submodule update
```

1.2 Install Prerequisites

1.2.1 Firmware Requirements

The LWA-253 F-Engine firmware can be built with the CASPER toolflow, and was designed using the following software stack:

- Ubuntu 18.04.0 LTS (64-bit)
- MATLAB R2019a
- Simulink R2019a
- MATLAB Fixed-Point Designer Toolbox R2019a
- Xilinx Vivado HLx 2019.1.3
- Python 3.6.9

It is *strongly* recommended that the same software versions be used to rebuild the design.

CHAPTER

TWO

CONTROL INTERFACE

2.1 Overview

A Python class `Snap2Fengine` is provided to encapsulate control of individual blocks in the firmware DSP pipeline. The structure of the software interface aims to mirror the hierarchy of the firmware modules, through the use of multiple `Block` class instances, each of which encapsulates control of a single module in the firmware pipeline.

In testing, and interactive debugging, the `Snap2Fengine` class provides an easy way to probe board status for a SNAP2 board on the local network.

In order to integrate with the larger LWA352 control framework, control and monitoring of multiple F-Engines can also be carried out through the passing of JSON-encoded messages through an `etcd`[1] key-value store. This mechanismi, shown in Fig. 2.1, utilizes the Python class `Snap2FengineEtcdClient` to translate commands and responses between the `etcd` format and the underlying `Snap2Fengine` method calls.

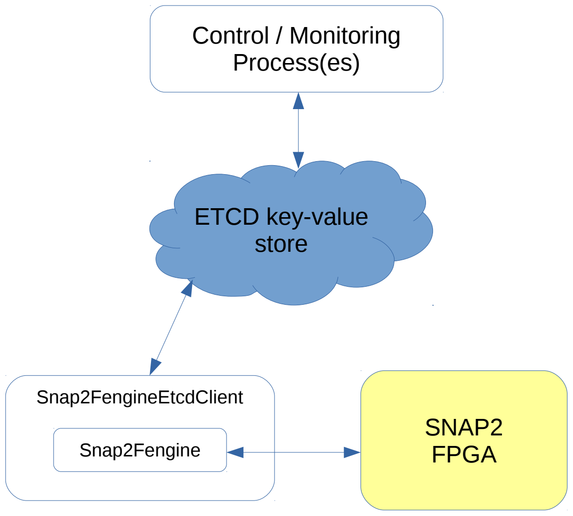

Fig. 2.1: The top-level control architecture.

[1] See etcd.io

2.2 `SNAP2Fengine` Python Interface

2.2.1 Top-Level Control

class `lwa_f.snap2_fengine.`**`Snap2Fengine`**(*host*, *logger=None*)

`configure_output`(*base_ant*, *n_chans_per_packet*, *chans*, *ips*, *ports=None*, *ants=None*)

2.2.2 FPGA Control

class `lwa_f.blocks.fpga.`**`Fpga`**(*host*, *name*, *logger=None*)

`get_status`()
Get FPGA stats. returns: Dictionary of stats

`is_programmed`()
Lazy check to see if a board is programmed. Check for the "version_version" register. If it exists, the board is deemed programmed.

Returns True if programmed, False otherwise.

Return type bool

2.2.3 Timing Control

class `lwa_f.blocks.sync.`**`Sync`**(*host*, *name*, *logger=None*)

`arm_noise`()
Arm noise generator resets

`arm_sync`()
Arm sync pulse generator.

`count_ext`()
Returns Number of external sync pulses received.

`count_int`()
Returns Number of internal sync pulses received.

`get_error_count`()
Returns count of number of sync errors

`get_latency`()
Returns Number of FPGA clock ticks between sync transmission and reception

`get_status`()
Get a dictionary of status values, with optional warning of error flags. To be overridden by individual blocks

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs, defined on a per-block basis. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary should be as defined in *error_levels.py*.

`initialize`(*read_only=False*)
Initialize this block. Set sync period to 0.

load_telescope_time(*tt*, *software_load=False*)
Load a new starting value into the telescope time counter on the next sync.

tt (int) : Telescope time to load software_load (bool) : If True, immediately load via a software trigger

period()
Returns the number of FPGA clock ticks between the last two external sync pulses.

reset_error_count()
Reset internal error counter to 0.

reset_telescope_time()
Reset the telescope time counter to 0 immediately.

sw_sync()
Issue a software sync pulse

uptime()
Returns uptime in FPGA clock ticks. Resolution is 2**32 (21 seconds at 200 MHz)

wait_for_sync()
Block until a sync has been received.

2.2.4 ADC Control

class lwa_f.blocks.adc.**Adc**(*host*, *name*, *logger=None*)
Instantiate a control interface for an ADC block.

Parameters

- **host** (*casperfpga.CasperFpga*) – CasperFpga interface for host.
- **name** (*str*) – Name of block in Simulink hierarchy.
- **logger** (*logging.Logger*) – Logger instance to which log messages should be emitted.

calibrate(*use_ramp=False*)
Compute and set all ADC data lane input delays to their optimal values. After this call, the ADCs are left in test mode.

Parameters use_ramp (*bool*) – If True, after calibration, use the ramp test pattern to verify the ADC->FPGA link is functioning correctly. If False, perform this verification with the same constant test value used for the calibration procedure.

Returns True if the calibration procedure succeeded. False otherwise.

Return type bool

get_snapshot(*fmc*, *signed=False*, *trigger=True*)
Read a snapshot of data from all ADC channels on a single FMC card. Return data without interleaving ADC cores.

Parameters

- **fmc** (*int*) – Which FMC port (0 or 1) to read.
- **signed** (*bool*) – If True, return data interpretted as signed 2's complement integers. If False, return data as unsigned integers.

- **trigger** (*bool*) – If True, trigger a new simultaneous capture of data from all channels. If False, read existing data capture. Grabbing data without a new trigger may be useful if you wish to read channels from a second FMC port which were captured simultaneously with another port.

Returns Array of captured data with dimensions [ADC_CHIPS_PER_FMC, ADC_LANES, TIME_SAMPLES]. Data from ADC lanes representing the same analog input are _not_ interleaved. Data from ADC lanes n,n+1 are associated with the same analog input.

Return type numpy.array

get_snapshot_interleaved(*fmc*, *signed=False*, *trigger=True*)

Read a snapshot of data from all ADC channels on a single FMC card. Return data with ADC cores interleaved.

Parameters

- **fmc** (*int*) – Which FMC port (0 or 1) to read.
- **signed** (*bool*) – If True, return data interpretted as signed 2's complement integers. If False, return data as unsigned integers.
- **trigger** (*bool*) – If True, trigger a new simultaneous capture of data from all channels. If False, read existing data capture. Grabbing data without a new trigger may be useful if you wish to read channels from a second FMC port which were captured simultaneously with another port.

Returns numpy array of captured data with dimensions [ADC_CHANNELS_PER_FMC, TIME_SAMPLES].

Return type numpy.ndarray

initialize(*read_only=False*, *clocksource=1*)

Initialize connected ADC boards.

Parameters

- **read_only** (*bool*) – If True, don't do anything which would affect a running system. If False, train ADC->FPGA data links.
- **clocksource** (*int*) – Which ADC board (0 or 1) on an FMC card should serve as the source of the clocks. Note that while this parameter is set for boards on all FMC cards, only the FMC card selected as the clock source at Simulink compile-time will be used for clocking.

Returns True if initialization was successful. False otherwise.

mmcm_is_locked()

Read the ADC control register to determine if the clock PLLs are locked.

Returns True if the ADC clocks are locked. False otherwise.

Return type bool

print_sweep(*errs*, *best_delays=None*, *step_size=4*)

Print, using ASCII, the valid data capture eye as a function of delay setting. Delays are printed such that one row represents a sweep of delays for a single ADC data lane. Each column in the row is `X` if data contained errors at this delay, `-` if no errors were detected at this delay, and `|`, if this delay is considered the best setting in the sweep range.

Parameters

- **errs** (*list*) – Array of error counts with dimensions [DELAY_TRIALS, ADC_CHIPS, DATA_LANES_PER_ADC_CHIP] such as that returned by `_get_errs_by_delay`.

- **best_delays** (*list*) – Array of best delays, with shape [ADC_CHIPS, DATA_LANES_PER_ADC_CHIP], such as that returned by `_get_best_delays`. These delays are marked with an ASCII `|`.
- **step_size** (*int*) – Number of IDELAY tap steps between delay trials.

reset()
Toggle the ADC reset input.

set_delays(*adc*, *delays*)
Set IDELAY tap values for all ADC data lanes on an FMC port.

Parameters

- **adc** (*casperfpga.ads5296.AD5296*) – ADS5296 object associated with an FMC ADC interface.
- **delays** (*list*) – Array of delays to load, with shape [ADC_CHIPS, DATA_LANES_PER_ADC_CHIP], such as that returned by `_get_best_delays`.

sync()
Toggle the ADC sync input.

2.2.5 Input Control

class lwa_f.blocks.input.**Input**(*host*, *name*, *n_streams=64*, *n_bits=10*, *logger=None*)
Instantiate a control interface for an Input block.

Parameters

- **host** (*casperfpga.CasperFpga*) – CasperFpga interface for host.
- **name** (*str*) – Name of block in Simulink hierarchy.
- **logger** (*logging.Logger*) – Logger instance to which log messages should be emitted.
- **n_streams** (*int*) – Number of independent streams which may be delayed
- **n_bits** (*int*) – Number of bits per ADC sample.

get_all_histograms()
Get histograms for all antpols, summing over all interleaving Input:

antpol (int): Antpol number (zero-indexed)

Returns:

vals, hist vals (numpy array): histogram bin centers hist (numpy array): histogram data

get_bit_stats()
Get the mean, RMS, and mean powers of all ADC streams.

Returns (means, powers, rmss) tuple. Each member of the tuple is an array with `self.n_streams` elements.

Rval (numpy.ndarray, numpy.ndarray, numpy.ndarray)

get_histogram(*input*, *sum_cores=True*)
Get a histogram for an ADC input. Inputs:

input (int): ADC input from which to get data. sum_cores (Boolean): If True, compute one histogram from both A & B ADC cores. If False, compute separate histograms.

Returns:

If sum_cores is True:

vals, hist vals (numpy array): histogram bin centers hist (numpy array): histogram data

If sum_cores is False:

vals, hist_a, hist_b vals (numpy array): histogram bin centers hist_a (numpy array): histogram data for "A" cores hist_b (numpy array): histogram data for "B" cores

`get_status`()

Get a dictionary of status values, with optional warning of error flags. To be overridden by individual blocks

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs, defined on a per-block basis. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary should be as defined in *error_levels.py*.

`get_switch_positions`()

Get the positions of the input switches.

Returns List of switch positions. Entry `n` contains the position of the switch associated with ADC input `n`. Switch positions are "noise" (internal digital noise generators), "adc" (digitized ADC stream), or "zero" (constant 0).

Return type list of str

`initialize`(*read_only=False*)

Initialize the block.

Parameters **`read_only`** (*`bool`*) – If True, do nothing. If False, set the input multiplexers to ADC data and enable statistic computation.

`show_histogram_plot`()

A helper method for plotting multiple histograms without importing your own pyplot. eg.

for i in range(Input.n_streams): Input.plot_histogram(i)

Input.show_histogram_plot()

`use_adc`(*stream=None*)

Switch input to ADC.

Parameters **`stream`** (*`int or None`*) – Which stream to switch. If None, switch all.

`use_noise`(*stream=None*)

Switch input to internal noise source.

Parameters **`stream`** (*`int or None`*) – Which stream to switch. If None, switch all.

`use_zero`(*stream=None*)

Switch input to zeros.

Parameters **`stream`** (*`int or None`*) – Which stream to switch. If None, switch all.

2.2.6 Noise Generator Control

class `lwa_f.blocks.noisegen.`**`NoiseGen`**(*host*, *name*, *n_noise=5*, *n_outputs=64*, *logger=None*)

Noise Generator controller

This block controls a digital noise source, which can generate multiple independent channels of gaussian noise. These channels can be assigned to multiple outputs of this block, to create correlated or uncorrelated noise streams.

`assign_output`(*output*, *noise*)

Assign an output channel with a given noise stream.

output [int] The index of the output stream to be assigned.

noise [int] The index of the noise stream to assign to *output*. Note that each noise generator core generates two independent streams, so *noise* can be in range(0, 2*self.n_noise)

`get_output_assignment`(*output*)

Get the index of the noise stream assigned to *output*

output [int] The index of the output stream to query.

noise [int] The index of the noise stream to assign to *output*. Note that each noise generator core generates two independent streams, so *noise* can be in range(0, 2*self.n_noise)

`get_seed`(*n*)

Get the seed of noise generatr *n*

`get_status`()

Get a dictionary of status values, with optional warning of error flags. To be overridden by individual blocks

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs, defined on a per-block basis. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary should be as defined in *error_levels.py*.

`initialize`(*read_only=False*)

Individual blocks should override this method to configure themselves appropriately

Parameters **`read_only`** (*`bool`*) – If False, initialize blocks in a way that might change the configuration of running hardware. If True, read runtime info from blocks, but don't change anything.

`set_seed`(*n*, *seed*)

Set the seed of noise generator *n*

2.2.7 Delay Control

class `lwa_f.blocks.delay.`**`Delay`**(*host*, *name*, *n_streams=64*, *logger=None*)

Instantiate a control interface for a Delay block.

Parameters

- **`host`** (*`casperfpga.CasperFpga`*) – CasperFpga interface for host.
- **`name`** (*`str`*) – Name of block in Simulink hierarchy.
- **`logger`** (*`logging.Logger`*) – Logger instance to which log messages should be emitted.
- **`n_streams`** (*`int`*) – Number of independent streams which may be delayed

MIN_DELAY = 5
minimum delay allowed

get_delay(*stream*)
Get the current delay for a given input.

Parameters **stream** (*int*) – Which ADC input index to query

Returns Currently loaded delay, in ADC samples

Return type int

get_max_delay()
Query the firmware to get the maximum delay it supports.

Returns Maximum supported delay, in ADC samples

Return type int

get_status()
Get status and error flag dictionaries.

Status keys:

- delay<n>: Currently loaded delay for ADC input index n.
- max_delay: The maximum delay supported by the firmware.
- min_delay: The minimum delay supported by the firmware.

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary are as defined in *error_levels.py* and indicate that values in the status dictionary are outside normal ranges.

initialize(*read_only=False*)
Initialize all delays.

Parameters **read_only** (*bool*) – If True, do nothing. If False, initialize all delays to the minimum allowed value.

set_delay(*stream*, *delay*)
Set the delay for a given input stream.

Parameters

- **stream** (*int*) – ADC stream index to which delay should be applied.
- **delay** (*int*) – Number of ADC clock cycles delay to load.

2.2.8 PFB Control

class lwa_f.blocks.pfb.**Pfb**(*host*, *name*, *logger=None*)

get_status()
Get a dictionary of status values, with optional warning of error flags. To be overridden by individual blocks

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs, defined on a per-block basis. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary should be as defined in *error_levels.py*.

initialize(*read_only=False*)

Individual blocks should override this method to configure themselves appropriately

Parameters **read_only** (*bool*) – If False, initialize blocks in a way that might change the configuration of running hardware. If True, read runtime info from blocks, but don't change anything.

2.2.9 Auto-correlation Control

class lwa_f.blocks.autocorr.**AutoCorr**(*host*, *name*, *acc_len=32768*, *logger=None*, *n_chans=4096*, *n_pols=64*, *n_parallel_streams=8*, *n_cores=4*, *use_mux=True*)

Instantiate a control interface for an Auto-Correlation block.

Parameters

- **host** (*casperfpga.CasperFpga*) – CasperFpga interface for host.
- **name** (*str*) – Name of block in Simulink hierarchy.
- **logger** (*logging.Logger*) – Logger instance to which log messages should be emitted.
- **acc_len** (*int*) – Accumulation length initialization value, in spectra.
- **n_chans** (*int*) – Number of frequency channels.
- **n_pols** (*int*) – Number of individual data streams.
- **n_parallel_streams** (*int*) – Number of streams processed by the firmware module in parallel.
- **n_cores** (*int*) – Number of accumulation cores in firmware design.
- **use_mux** (*bool*) – If True, only one core is instantiated and a multiplexer is used to switch different inputs into it. If False, multiple cores are instantiated simultaneously in firmware.

get_acc_len()

Get the currently loaded accumulation length in units of spectra.

Returns Current accumulation length

Return type int

get_new_spectra(*core=0*)

Get a new average power spectra.

Parameters **core** (*int*) – If using multiplexing, read data for this core. If not using multiplexing, read data from all cores.

Returns Float32 array of dimensions [POLARIZATION, FREQUENCY CHANNEL] containing autocorrelations with accumulation length divided out.

Return type numpy.array

get_status()

Get status and error flag dictionaries.

Status keys:

- acc_len : Currently loaded accumulation length

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary are as defined in *error_levels.py* and indicate that values in the status dictionary are outside normal ranges.

`initialize`(*read_only=False*)
Initialize the block, setting (or reading) the accumulation length.

Parameters **`read_only`** (`bool`) – If False, set the accumulation length to the value provided when this block was instantiated. If True, use whatever accumulation length is currently loaded.

`plot_spectra`(*core=0*, *db=True*, *show=True*)
Plot the spectra of all polarizations in a single core, with accumulation length divided out

Parameters

- **`core`** (`int`) – If using multiplexing, read data for this core. If not using multiplexing, read data from all cores.
- **`db`** (`bool`) – If True, plot 10log10(power). Else, plot linear.
- **`show`** (`bool`) – If True, call matplotlib's *show* after plotting

Returns matplotlib.Figure

`set_acc_len`(*acc_len*)
Set the number of spectra to accumulate.

Parameters **`acc_len`** (`int`) – Number of spectra to accumulate

2.2.10 Correlation Control

class `lwa_f.blocks.corr.`**`Corr`**(*host*, *name*, *acc_len=1024*, *logger=None*, *n_chans=1024*)
Instantiate a control interface for a Correlation block.

Parameters

- **`host`** (`casperfpga.CasperFpga`) – CasperFpga interface for host.
- **`name`** (`str`) – Name of block in Simulink hierarchy.
- **`logger`** (`logging.Logger`) – Logger instance to which log messages should be emitted.
- **`acc_len`** (`int`) – Accumulation length initialization value, in spectra.
- **`n_chans`** (`int`) – Number of frequency channels in the correlation output.

`get_acc_len`()
Get the currently loaded accumulation length in units of spectra.

Returns Current accumulation length

Return type int

`get_new_corr`(*pol1*, *pol2*, *flush_vacc=True*)
Get a new correlation.

Parameters

- **`pol1`** (`int`) – First (unconjugated) polarization index.
- **`pol2`** (`int`) – Second (conjugated) polarization index.

- **flush_vacc** (*bool*) – If True, throw away the first accumulation after setting the input selection registers. This is good practice the first time a new polarization pair is read.

Returns Complex-valued cross-correlation spectra of *pol1* and *pol2* with accumulation length divided out.

Return type numpy.array

get_status()
Get status and error flag dictionaries.

Status keys:

- acc_len : Currently loaded accumulation length

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary are as defined in *error_levels.py* and indicate that values in the status dictionary are outside normal ranges.

initialize(*read_only=False*)
Initialize the block, setting (or reading) the accumulation length.

Parameters **read_only** (*bool*) – If False, set the accumulation length to the value provided when this block was instantiated. If True, use whatever accumulation length is currently loaded.

plot_corr(*pol1*, *pol2*, *show=False*)
Plot a correlation spectra.

Parameters

- **pol1** (*int*) – First (unconjugated) polarization index.
- **pol2** (*int*) – Second (conjugated) polarization index.
- **show** (*bool*) – If True, call matplotlib's *show* after plotting

Returns matplotlib.Figure

set_acc_len(*acc_len*)
Set the number of spectra to accumulate.

Parameters **acc_len** (*int*) – Number of spectra to accumulate

2.2.11 Post-FFT Test Vector Control

class lwa_f.blocks.eqtvg.**EqTvg**(*host*, *name*, *n_streams=64*, *n_chans=4096*, *logger=None*)
Instantiate a control interface for a post-equalization test vector generator block.

Parameters

- **host** (*casperfpga.CasperFpga*) – CasperFpga interface for host.
- **name** (*str*) – Name of block in Simulink hierarchy.
- **logger** (*logging.Logger*) – Logger instance to which log messages should be emitted.
- **n_streams** (*int*) – Number of independent streams which may be delayed
- **n_chans** (*int*) – Number of frequency channels.

get_status()

Get status and error flag dictionaries.

Status keys:

- tvg_enabled: Currently state of test vector generator. `True` if the generator is enabled, else `False`.

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary are as defined in *error_levels.py* and indicate that values in the status dictionary are outside normal ranges.

initialize(*read_only=False*)

Initialize the block.

Parameters **read_only** (*bool*) – If True, do nothing. If False, load frequency-ramp test vectors, but disable the test vector generator.

read_stream_tvg(*stream*)

Read the test vector loaded to an ADC stream.

Parameters **stream** (*int*) – Index of stream from which test vectors should be read.

Returns Test vector array, as loaded to the FPGA in 8-bit unsigned integer representation.

Return type numpy.ndarray

tvg_disable()

Disable the test vector generator

tvg_enable()

Enable the test vector generator.

tvg_is_enabled()

Query the current test vector generator state.

Returns True if the test vector generator is enabled, else False.

Return type bool

write_const_per_stream()

Write a constant to all the channels of a stream, with stream *i* taking the value *i*.

write_freq_ramp()

Write a frequency ramp to the test vector that is repeated for all ADC inputs. Data are wrapped to fit into 8 bits. I.e., the test vector value for channel 257 takes the value `1`.

write_stream_tvg(*stream*, *test_vector*)

Write a test vector pattern to a single signal stream.

Parameters

- **stream** (*int*) – Index of stream to which test vectors should be loaded.
- **test_vector** (*list or numpy.ndarray*) – *self.n_chans*-element test vector. Values should be representable in 8-bit unsigned integer format. Data are loaded such that the most-significant 4 bits of the test_vectors are interpretted as the 2's complement 4-bit real sample data. The least-significant 4 bits of the test vectors are interpretted as the 2's complement 4-bit imaginary sample data.

2.2.12 Equalization Control

class `lwa_f.blocks.eq.`**`Eq`**(*host*, *name*, *n_streams=64*, *n_coeffs=512*, *logger=None*)

Instantiate a control interface for an Equalization block.

Parameters

- **`host`** (*`casperfpga.CasperFpga`*) – CasperFpga interface for host.
- **`name`** (*`str`*) – Name of block in Simulink hierarchy.
- **`logger`** (*`logging.Logger`*) – Logger instance to which log messages should be emitted.
- **`n_streams`** (*`int`*) – Number of independent streams which may be delayed
- **`n_coeffs`** (*`int`*) – Number of coefficients per input stream. Coefficients are shared among neighbouring frequency channels.

`clip_count`()

Get the total number of times any samples have clipped, since last sync.

Returns Clip count.

Return type int

`get_coeffs`(*stream*)

Get the coefficients currently loaded. Reads the actual coefficients from the board.

Parameters **`stream`** (*`int`*) – ADC stream index to query.

Returns (coeffs, binary_point). `coeffs` is an array of `self.n_coeffs` integer coefficients currently being applied. `binary_point` is the position of the binary point wy which these integers are scaled on the FPGA.

Return type (numpy.ndarray, int)

`get_status`()

Get status and error flag dictionaries.

Status keys:

- clip_count: Number of clip events in the last sync period.
- width: Bit width of coefficients
- binary_point: Binary point position of coefficients
- coefficients<`n`>: The currently loaded, integer-valued coefficients for ADC stream `n`.

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary are as defined in *error_levels.py* and indicate that values in the status dictionary are outside normal ranges.

`initialize`(*read_only=False*)

Initialize block.

Parameters **`read_only`** (*`bool`*) – If False, set all coefficients to some nominally sane value. Currently, this is 100.0. If True, do nothing.

`set_coeffs`(*stream*, *coeffs*)

Set the coefficients for a data stream. Clipping and saturation will be applied before loading.

Parameters

- **stream** (*int*) – ADC stream index to which coefficients should be applied.
- **coeffs** (*list or numpy.ndarray*) – Array of coefficients to load. This should be of length `self.n_coeffs`, else an AssertionError will be raised.

2.2.13 Channel Selection Control

class `lwa_f.blocks.chanreorder.`**ChanReorder**(*host*, *name*, *n_chans=1024*, *logger=None*)
Instantiate a control interface for a Channel Reorder block.

Parameters

- **host** (*casperfpga.CasperFpga*) – CasperFpga interface for host.
- **name** (*str*) – Name of block in Simulink hierarchy.
- **logger** (*logging.Logger*) – Logger instance to which log messages should be emitted.
- **n_chans** (*int*) – Number of channels in the system.

initialize(*read_only=False*)
Initialize the block.

Parameters **read_only** (*bool*) – If True, this method is a no-op. If False, initialize the block with the identity map. I.e., map channel *n* to channel *n*.

n_parallel_chans = 8
Number of channels per reorder word

read_reorder(*raw=False*)
Read the currently loaded reorder map.

Parameters **raw** (*bool*) – If True, return the map as loaded onto the FPGA. If False, return the resulting channel map – i.e., the channel ordering being output by this F-engine.

Returns The reorder map currently loaded.

Return type list

set_channel_order(*order*)
Re-order channels so that they are sent with the order in the specified map.

There are various requirements of the map which must be met.

- Every integer multiple of *self.n_parallel_chans* of the map must start on an integer multiple of *n_parallel_chans*. Eg., for *n_parallel_chans = 8 order[0] = 16* is acceptable. *order[0] = 4* is not.
- Blocks of *n_parallel_channels* must be consecutive. Eg., if *n_parallel_chans=8*, *order[16:24] = [0,1,2,3,4,5,6,7]* is acceptable. *order[16:24] = [0,1,2,3,8,9,10,11]* is not.
- The provided map must be *self.n_chans* elements long.

Parameters **order** (*list of int*) – The order to which data should be mapped. I.e., if *order[0] = 16*, then the first channel out of the F-engine will be channel 16. The order map should meet the above criteria. A ValueError exception will be raised if they are not.

2.2.14 Packetization Control

class `lwa_f.blocks.packetizer.`**`Packetizer`**(*host*, *name*, *n_chans=4096*, *n_pols=64*, *sample_rate_mhz=200.0*, *logger=None*)

The packetizer block allows dynamic definition of packet sizes and contents. In firmware, it is a simple block which allows insertion of header entries and EOFs at any point in the incoming data stream. It is up to the user to configure this block such that it behaves in a reasonable manner – i.e.

- Output data rate does not overflow the downstream Ethernet core
- Packets have a reasonable size
- EOFs and headers are correctly placed.

`get_packet_info`(*n_pkt_chans*, *occupation=0.95*, *chan_block_size=8*)

Get the packet boundaries for packets with payload sizes *n_bytes*.

n_pkt_chans [int] The number of channels per packet.

occupation [float] The maximum allowed throughput capacity of the underlying link. The calculation does not include application or protocol overhead, so must necessarily be < 1.

chan_block_size [int] The granularity with which we can start packets. I.e., packets must start on an n*`chan_block` boundary.

packet_starts, packet_payloads, channel_indices

packet_starts [list of ints] The word indexes where packets start – i.e., where headers should be written. For example, a value [0, 1024, 2048, ...] indicates that headers should be written into underlying brams at addresses 0, 1024, etc.

packet_payloads [list of range()] The range of indices where this packet's payload falls. Eg: [range(1,257), range(1025,1281), range(2049,2305), ... etc] These indices should be marked valid, and the last given an EOF.

channel_indices [list of range()] The range of channel indices this packet will send. Eg: [range(1,129), range(1025,1153), range(2049,2177), ... etc] Channels to be sent should be re-indexed so that they fall into these ranges.

`write_config`(*packet_starts*, *packet_payloads*, *channel_indices*, *ant_indices*, *dest_ips*, *dest_ports*, *print_config=False*)

Write the packetizer configuration BRAMs with appropriate entries.

packet_starts [list of ints] Word-indices which are the first entry of a packet and should be populated with headers (see *get_packet_info()*)

packet_payloads [list of range()s] Word-indices which are data payloads, and should be mared as valid (see *get_packet_info()*)

channel_indices [list of ints] Header entries for the channel field of each packet to be sent

ant_indices [list of ints] Header entries for the antenna field of each packet to be sent

dest_ips [list of str] IP addresses for each packet to be sent.

dest_ports [list of int] UDP destination ports for each packet to be sent.

print [bool] If True, print config for debugging

All parameters should have identical lengths.

2.2.15 Ethernet Output Control

class `lwa_f.blocks.eth.`**`Eth`**(*host*, *name*, *logger=None*)

Instantiate a control interface for a 40 GbE block.

Parameters

- **`host`** (*`casperfpga.CasperFpga`*) – CasperFpga interface for host.
- **`name`** (*`str`*) – Name of block in Simulink hierarchy.
- **`logger`** (*`logging.Logger`*) – Logger instance to which log messages should be emitted.

`add_arp_entry`(*ip*, *mac*)

Set a single arp entry.

Parameters

- **`ip`** (*`int`*) – The last octet of the IP address matching the given MAC.
- **`mac`** (*`int`*) – The MAC address to be loaded to the ARP cache.

`disable_tx`()

Disable Ethernet transmission.

`enable_tx`()

Enable Ethernet transmission.

`get_status`()

Get a dictionary of status values, with optional warning of error flags. To be overridden by individual blocks

Returns (status_dict, flags_dict) tuple. *status_dict* is a dictionary of status key-value pairs, defined on a per-block basis. flags_dict is a dictionary with all, or a sub-set, of the keys in *status_dict*. The values held in this dictionary should be as defined in *error_levels.py*.

`initialize`(*read_only=False*)

Individual blocks should override this method to configure themselves appropriately

Parameters **`read_only`** (*`bool`*) – If False, initialize blocks in a way that might change the configuration of running hardware. If True, read runtime info from blocks, but don't change anything.

`reset`()

Disable, then reset the 40 GbE core. It must be enabled with `enable_tx` before traffic will be transmitted.

`set_arp_table`(*macs*)

Set the ARP table with a list of MAC addresses.

Parameters **`macs`** (*`list of int`*) – MAC addresses to be loaded into the ARP table. These should be the form of a list of integers, such that the ``n``th list entry contains the MAC address of the device with IP *XXX.XXX.XXX.n*.

`status_reset`()

Reset all status counters.

2.3 `etcd` Interface

The `etcd` F-Engine interface provides a mechanism to control multiple SNAP2 boards running F-Engine firmware via the passing of messages through an `etcd` key-value store.

In order to use the `etcd` control interface, a daemon `Snap2FengineEtcdClient` instance is required to be running on a server with network access to the SNAP2 hardware being controlled.

2.3.1 Key Organization

For an F-Engine running on a SNAP2 `hostname`, there are three relevant `etcd` key paths.

Command

The command key is:

`/cmd/snap/<hostname>/command`

Writing messages to this key results in the execution of `Snap2Fengine` command methods.

Response

Each command written to the command key will elicit a response published to the key:

`/resp/snap/<hostname>/response`

The response message will indicate the success of failure of the command, and may contain a command-dependent data payload. For example, a spectra, or snapshot of ADC samples.

Monitor

In addition to the Command/Response control protocol, general telemetry relating to pipeline element `blockname` can be read from the keys beneath the path:

`/mon/snap/<hostname>/<blockname>`

Such keys are indended to be continuously updated on a ~1 second time cadence, and contain low data-volume status information. For example, FPGA temperature, ADC RMS, number of transmitted Ethernet packets, and similar.

2.3.2 Command/Response Protocol

The Command/Response protocol is designed to be a simple interface to the underlying `Snap2Fengine` control class. It will naturally extend as the control class functionality is expanded.

A simple example of a control client is implemented in the `Snap2FengineEtcdControl` class, which is used for internal testing.

Command Format

Commands sent to the command key are JSON-encoded dictionaries, and should have the following fields:

Field	Type	Description
se-quence_id	integer	A unique integer associated with this command, used to identify the command's re-sponse
timestamp	float	The UNIX time when this command was issued
command	string	Command Name
block	string	Firmware block name to which command applies
kwargs	dictio-nary	Dictionary of arguments required by the `block.command` method

Allowed values for ``**block**`` are any of the keys in the `Snap2Fengine` blocks attribute. I.e.:

```
from lwa_f import snap2_fengine
f = snap2_fengine.Snap2Fengine('snap2-rev2-11')
for block in sorted(f.blocks.keys()): print(block)
adc
autocorr
corr
delay
eq
eq_tvg
eth
input
noise
packetizer
pfb
reorder
sync
```

Allowed values for ``**command**`` are any of the methods which can be called against `Snap2Fengine.blocks[block]`. For example, for the `delay` block, allowed commands are:

- `get_max_delay`
- `set_delay`
- `get_delay`
- `initialize`
- `get_status`

All blocks are instances of the generic `Block` class, and thus it is also possible to call parent class methods such as `read_uint` and `write_uint`. These directly manipulate FPGA registers, and should be used with caution.

The ``**kwargs**`` field should contain any arguments required by the command method being called. For example, the Fengine `delay` block's `set_delay` method requires a `stream` argument (to select which of the 64 SNAP2 data streams is being manipulated) and a `delay` argument (to set the delay for this stream).

As such, in order to set the delay of data stream `5` to `100` adc samples, a command should be issued with:

Field	Value
`block`	`"delay"`
`command`	`"set_delay"`
`kwargs`	`{"stream": 5, "delay": 100}`

An example of a valid command JSON string, issued with the above parameters at UNIX time 1618060712.60 and with `sequence_id=1` is:

```
'{"block": "delay", "timestamp": 1618060712.6, "kwargs": {"delay": 100,
"stream": 5}, "command": "set_delay", "sequence_id": 1}'
```

Consult the `Snap2Fengine` API details for a list of commands and their arguments.

Response Format

Every command sent elicits the writing of JSON-encoded dictionary to the response key. This dictionary has the following fields:

Field	Type	Description
sequence_id	integer	An integer matching the `sequence_id` field of the command string to which this is a response
timestamp	float	The UNIX time when this response was issued
status	string	The string "normal" if the corresponding command was processed without error, or "error" if it was not.
response	command-dependent	The response of the command method, as determined by the command API. If a method would usually return a numpy array, when using the `etcd` interface the response will be a list. In the event that the status field is `"error"`, The response field will contain an error message string

Not all `Snap2Fengine` methods return values, in whice case the response field is `null`. The previous command example (setting a delay) results in the underlying API call `Snap2Fengine.blocks['delay'].set_delay(5, 100)` which returns `None`. The response to the example command (assuming processing the command took 0.2 milliseconds) is thus:

Field	Value
`sequence_id`	`1`
`timestamp`	`1618060712.8`
`status`	`"normal"`
`response`	`null`

or, in JSON-encoded form:

```
'{"sequence_id": 1, "timestamp": 1618060712.8, "status": "normal",
"response": null}'
```

If the response `status` field is "error", common `response` error messages, and their meanings are:

"JSON decode error"	Command string could not be JSON-decoded.
"Sequence ID not integer"	Sequence ID was not provided in the command string or decoded to a non-integer value.
"Bad command format"	Received command did not comply with formatting specifications. E.g. was missing a required field such as `block` or `command`.
"Command invalid"	Received command doesn't exist in the `Snap2Fengine` API, or is prohibited for `etcd` access.
"Wrong block"	`block` field of the command decoded to a block which doesn't exist.
"Command arguments invalid"	`kwargs` key contained missing, or unexpected keys.
"Command failed"	The underlying `Snap2Fengine` API call raised an exception.

In the event that a command fails, more information is available in the `Snap2FengineEtcdClient` daemon logs.

Monitoring Interface

autocorr/acc_len	int	Accumulation length, in spectra, of the internal autocorrelation module.
corr/acc_len	int	Accumulation length, in spectra, of the internal correlation module.
delay/`n`/delay	int	Delay of each of the `n` data streams.
delay/maxdelay	int	The maximum delay supported by the firmware.
eq/binary_point	int	The position of the EQ coefficient binary point
eq/width	int	The bit width of the EQ coefficients
eq/clip_count	int	The number of clips when requantizing spectra to 4-bit. This counter resets every [TODO: w
eq/`n`/coefficients	list (int)	The currently loaded coefficients, in integer form (i.e., interpretted with all bits above the bin
eq_tvg/tvg_enabled	bool	True if the post-FFT test vector generator is enabled. False otherwise.
eth/tx_ctr	int	Running count of number of packets transmitted.
eth/tx_err	int	Running count of number of packet errors detected.
eth/tx_full	int	Running count of number of transmission buffer overflow events.
eth/tx_vld	int	Running count of number of 256-bit words transmitted.
feng/flash_firmware	string	The current .fpg bitstream file stored in flash memory
feng/flash_firmware_md5	int	The MD5 checksum of the .fpg bitstream stored in flash
feng/host	string	The hostname of the SNAP2 board
feng/programmed	bool	True if the board appears to be running DSP firmware. False otherwise.
feng/serial	string	A notional "serial number" of this hardware. Not yet implemented.
feng/sw_version	string	The software version of the `lwa_f` python library currently in use
feng/sys_mon	string	"reporting" if the current firmware has a working system monitor module. "not reporting" if
feng/temp	float	FPGA junction temperature, in degrees C, reported by system monitor (if available)
feng/vccaux	float	Voltage of the VCCAUX FPGA power rail reported by system monitor (if available)
feng/vccbram	float	Voltage of the VCCBRAM FPGA power rail reported by system monitor (if available)
feng/vccint	float	Voltage of the VCCINT FPGA power rail reported by system monitor (if available)
input/`n`/mean	float	Mean of ADC sample values for input ADC `n`.
input/`n`/rms	float	RMS of ADC sample values for input ADC `n`.
input/`n`/power	float	Mean of squares of ADC sample values for input ADC `n`.
input/`n`/switch_position	string	Switch position for input data stream `n`. `noise` for internal noise generators, `adc` for analog
noise/noise_core00_seed	int	Seed value for internal noise generator 0.
noise/noise_core01_seed	int	Seed value for internal noise generator 1.
noise/noise_core02_seed	int	Seed value for internal noise generator 2.
noise/`n`/output_assignment	int	Noise source (0 - 5) currently assigned to data stream `n`.
pfb/fft_shift	int	Currently loaded FFT shift schedule.
pfb/overflow_count	int	Running count of FFT overflow events.
sync/ext_count	int	Running count of number of external sync pulses received since FPGA was programmed.
sync/int_count	int	Running count of number of internal sync pulses received since FPGA was programmed.
sync/period_fpga_clks	int	Detected period of external sync pulses in units of FPGA clock cycles.
sync/uptime_secs	int	Number of seconds since FPGA was last programmed

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX