SOUK MKID Readout

Release souk_mkid_readout-ae29423:souk_mkid_readout-0.0.1-8305bd3d-dirty

Jack Hickish

Feb 23, 2023

CONTENTS:

CHAPTER

ONE

INSTALLATION

The SOUK MKID Readout pipeline firmware and software is available at https://github.com/realtimeradio/souk-firmware. Follow the instructions here 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/souk-firmware
# Clone relevant submodules
cd souk-firmware
git submodule init
git submodule update
```

1.2 Install Prerequisites

1.2.1 Firmware Requirements

The SOUK MKID Readout firmware can be built with the CASPER toolflow, and was designed using the following software stack:

- Ubuntu 18.04.0 LTS (64-bit)
- MATLAB R2021a
- Simulink R2021a
- MATLAB Fixed-Point Designer Toolbox R2021a
- Xilinx Vivado HLx 2021.2
- Python 3.8

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

CHAPTER

TWO

F-ENGINE SYSTEM OVERVIEW

2.1 Overview

A block diagram of the readout system – which is also the top-level of the Simulink source code for the firmware – is shown in Fig. 2.1.

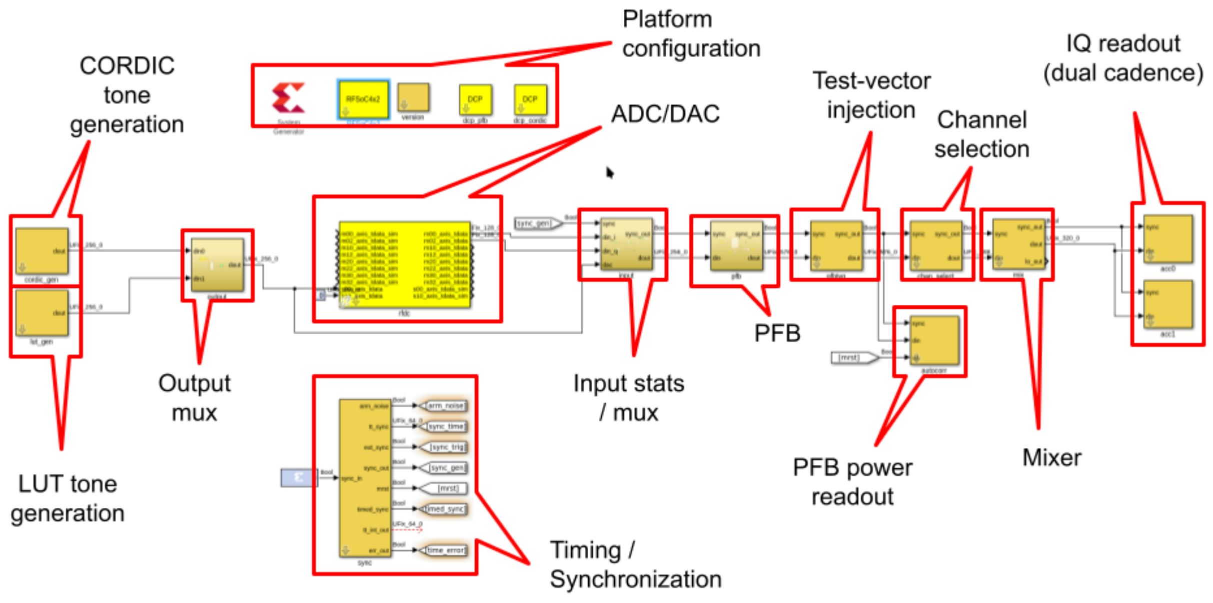

Fig. 2.1: Top-level Simulink diagram.

2.1.1 Initialization

The functionality of individual blocks is described below. However, in order to simply get the firmware into a basic working state the following process should be followed:

1. Program the FPGA
2. Initialize all blocks in the system
3. Trigger master reset and timing synchronization event.

In a multi-board system, the process of synchronizing a board can be relatively involved. For testing purposes, using single board, a simple software reset can be used in place of a hardware timing signal to perform an artificial synchronization. A software reset is automatically issued as part of system initialization.

The following commands bring the F-engine firmware into a functional state, suitable for testing. See Section 3 for a full software API description

```
# Import the SNAP2 F-Engine library
from souk_mkid_readout import SoukFirmwareReadout

# Instantiate an SoukFirmwareReadout instance, connecting to a board with
# hostname 'my_zcu111'
f = SoukFirmwareReadout('my_zcu111', config_file='my_config_file.yaml')

# Program a board
f.program() # Load whatever firmware was listed in the config file

# Initialize all the firmware blocks
# and issue a global software reset
f.initialize()
```

2.1.2 Block Descriptions

Each block in the firmware design can be controlled using an API described in Section 3.

CHAPTER THREE

CONTROL INTERFACE

3.1 Overview

A Python class `SoukMkidReadout` 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 `SoukMkidReadout` class provides an easy way to probe board status for a RFSoC board on the local network.

3.2 `SoukMkidReadout` Python Interface

The `SoukMkidReadout` class can be instantiated and used to control a single RFSoC board running LWA's F-Engine firmware. An example is below:

```python
# Import the RFSoC F-Engine library
from souk_mkid_readout import SoukMkidReadout

# Instantiate a SoukMkidReadout instance to a board with
# hostname 'my_zcu111'
f = SoukMkidReadout('my_zcu111', configfile='my_config.yaml')

# Program a board (if it is not already programmed)
# and initialize all the firmware blocks
if not f.fpga.is_programmed():
  f.program() # Load whatever firmware is in flash
  # Initialize firmware blocks
  f.initialize()

# Blocks are available as items in the SoukMkidReadout `blocks`
# dictionary, or can be accessed directly as attributes
# of the SoukMkidReadout.

# Print available block names
print(sorted(f.blocks.keys()))
# Returns:
# ['rfdc', 'input', 'autocorr', 'pfb', 'pfbtvg', 'chanreorder', 'mix',
# 'gen_lut', 'gen_cordic', 'output', 'accumulator0', 'accumulator1']
```

(continues on next page)

(continued from previous page)

```
# Grab some ADC data from the ADC
adc_data = f.input.get_adc_snapshot()
```

Details of the methods provided by individual blocks are given in the next section.

3.2.1 Top-Level Control

The Top-level `SoukMkidReadout` instance can be used to perform high-level control of the firmware, such as programming and de-programming FPGA boards. It can also be used to apply configurations which affect multiple firmware subsystems, such as configuring channel selection and packet destination.

Finally, a `SoukMkidReadout` instance can be used to initialize, or get status from, all underlying firmware modules.

`class souk_mkid_readout.`**`SoukMkidReadout`**`(`*host*, *fpgfile=None*, *configfile=None*, *logger=None*`)`

A control class for SOUK MKID Readout firmware on a single board

Parameters

- **`host`** (*`str`*) – Hostname/IP address of FPGA board
- **`fpgfile`** (*`str`*) – Path to .fpg file running on the board
- **`configfile`** (*`str`*) – Path to configuration YAML file for system
- **`logger`** (*`logging.Logger`*) – Logger instance to which log messages should be emitted.

`configfile`

configuration YAML file

`fpgfile`

fpgfile currently in use

`get_status_all``()`

Call the `get_status` methods of all blocks in `self.blocks`. If the FPGA is not programmed with F-engine firmware, will only return basic FPGA status.

Returns

(status_dict, flags_dict) tuple. Each is a dictionary, keyed by the names of the blocks in `self.blocks`. These dictionaries contain, respectively, the status and flags returned by the `get_status` calls of each of this F-Engine's blocks.

`hostname`

hostname of FPGA board

`initialize``(`*read_only=False*`)`

Call the `` `initialize `` methods of all underlying blocks, then optionally issue a software global reset.

Parameters

`read_only` (*`bool`*) – If True, call the underlying initialization methods in a read_only manner, and skip software reset.

`is_connected``()`

Returns

True if there is a working connection to a board. False otherwise.

Return type

bool

`logger`

Python Logger instance

`print_status_all`(*use_color=True*, *ignore_ok=False*)

Print the status returned by `get_status` for all blocks in the system. If the FPGA is not programmed with F-engine firmware, will only print basic FPGA status.

Parameters

- **use_color** (*bool*) – If True, highlight values with colors based on error codes.
- **ignore_ok** (*bool*) – If True, only print status values which are outside the normal range.

`program`(*fpgfile=None*)

Program an .fpg file to an FPGA.

Parameters

fpgfile (*str*) – The .fpg file to be loaded. Should be a path to a valid .fpg file. If None is given, *self.fpgfile* will be loaded. If this is None, RuntimeError is raised

3.2.2 FPGA Control

The `FPGA` control interface allows gathering of FPGA statistics such as temperature and voltage levels. Its methods are functional regardless of whether the FPGA is programmed with an LWA F-Engine firmware design.

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

Instantiate a control interface for top-level FPGA control.

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.

`get_build_time`()

Read the UNIX time at which the current firmware was built.

Return build_time

Seconds since the UNIX epoch at which the running firmware was built.

Rtype int

`get_connected_antname`()

Fetch the connected antenna name.

Return self.antname

The name of the connected antennna.

Rtype str

`get_firmware_type`()

Read the firmware type register and return the contents as an integer.

Return type

Firmware type

Rtype str

`get_firmware_version()`

Read the firmware version register and return the contents as a string.

Return version
major_version.minor_version.revision.bugfix

Rtype str

`get_fpga_clock()`

Estimate the FPGA clock, by polling the `sys_clkcounter` register.

Returns
Estimated FPGA clock in MHz

Return type
float

`get_status()`

Get status and error flag dictionaries.

Status keys:

- programmed (bool) : `True` if FPGA appears to be running DSP firmware. `False` otherwise, and flagged as a warning.
- flash_firmware (str) : The name of the firmware file currently loaded in flash memory.
- flash_firmware_md5 (str) : The MD5 checksum of the firmware file currently loaded in flash memory.
- timestamp (str) : The current time, as an ISO format string.
- host (str) : The host name of this board.
- antname (str) : The name of the antenna connected to this board.
- sw_version (str) : The version string of the control software package. Flagged as warning if the version indicates a build against a dirty git repository.
- fw_version (str): The version string of the currently running firmware. Available only if the board is programmed.
- fw_type (int): The firmware type ID of the currently running firmware. Available only if the board is programmed.
- fw_build_time (int): The build time of the firmware, as an ISO format string. Available only if the board is programmed.
- sys_mon (str) : `'reporting'` if the current firmware has a functioning system monitor module. Otherwise `'not reporting'`, flagged as an error.
- temp (float) : FPGA junction temperature, in degrees C. (Only reported is system monitor is available). Flagged as a warning if outside the recommended operating conditions. Flagged as an error if outside the absolute maximum ratings. See DS892.
- vccaux (float) : Voltage of the VCCAUX FPGA power rail. (Only reported is system monitor is available). Flagged as a warning if outside the recommended operating conditions. Flagged as an error if outside the absolute maximum ratings. See DS892.
- vccbram (float) : Voltage of the VCCBRAM FPGA power rail. (Only reported is system monitor is available). Flagged as a warning if outside the recommended operating conditions. Flagged as an error if outside the absolute maximum ratings. See DS892.

- vccint (float) : Voltage of the VCCINT FPGA power rail. (Only reported is system monitor is available). Flagged as a warning if outside the recommended operating conditions. Flagged as an error if outside the absolute maximum ratings. See DS892.

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.

`is_programmed()`

Check to see if a board is programmed. If the Katcp command *listdev* fails, assume that it isn't

Returns
True if programmed, False otherwise.

Return type
bool

`set_connected_antname(`*antname*`)`

Set the connected antenna name.

Parameters
antname (*str*) – The antenna name.

3.2.3 Timing Control

The Sync control interface provides an interface to configure and monitor the multi-RFSoC timing distribution system.

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

The Sync block controls internal timing signals.

Parameters

- **host** (*casperfpga.CasperFpga*) – CasperFpga interface for host.
- **name** (*str*) – Name of block in Simulink hierarchy.
- **clk_hz** (*int*) – The FPGA clock rate at which the DSP fabric runs, in Hz.
- **logger** (*logging.Logger*) – Logger instance to which log messages should be emitted.

`arm_noise()`

Arm noise generator resets.

`arm_sync(`*wait=True*`)`

Arm sync pulse generator, which passes sync pulses to the design DSP.

Parameters
wait (*bool*) – If True, wait for a sync to pass before returning.

`count_ext()`

Returns
Number of external sync pulses received.

Rtype int

`get_error_count()`

Returns
Number of sync errors.

Rtype int

`get_latency()`

Returns
Number of FPGA clock ticks between sync transmission and reception

Rtype int

`get_status()`

Get status and error flag dictionaries.

Status keys:

- uptime_fpga_clks (int) : Number of FPGA clock ticks (= ADC clock ticks) since the FPGA was last programmed.
- period_fpga_clks (int) : Number of FPGA clock ticks (= ADC clock ticks) between the last two internal sync pulses.
- ext_count (int) : The number of external sync pulses since the FPGA was last programmed.
- int_count (int) : The number of internal sync pulses since the FPGA was last programmed.

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.

`get_tt_of_ext_sync()`

Get the internal TT at which the last sync pulse arrived.

Returns
(tt, sync_number). `tt` is the internal TT of the last sync. `sync_number` is the sync pulse count corresponding to this TT.

Rtype int

`get_tt_of_sync()`

Get the internal TT of the last system sync event.

Returns
tt. The internal TT of the last sync.

Rtype int

`initialize(read_only=False)`

Initialize block.

Parameters
`read_only` (*bool*) – If False, initialize system control register to 0 and reset error counters. If True, do nothing.

`load_internal_time`(*tt*, *software_load=False*)

Load a new starting value into the _internal_ telescope time counter on the next sync.

Parameters

- **`tt`** (*`int`*) – Telescope time to load
- **`software_load`** (*`bool`*) – If True, immediately load via a software trigger. Else load on the next external sync pulse arrival.

`period`()

Returns
The number of FPGA clock ticks between the last two external sync pulses.

Rtype int

`reset_error_count`()

Reset internal error counter to 0.

`sw_sync`()

Issue a sync pulse from software. This will only do anything if appropriate arming commands have been made in advance.

`update_internal_time`(*fs_hz=None*, *offset_ns=0.0*, *sync_clock_factor=1*)

Arm sync trigger receivers, having loaded an appropriate telescope time.

Parameters

- **`fs_hz`** (*`int`*) – The FPGA DSP clock rate, in Hz. Used to set the telescope time counter. If None is provided, self.clk_hz will be used.
- **`offset_ns`** (*`float`*) – Nanoseconds offset to add to the time loaded into the internal telescope time counter.

Returns
next_sync_clocks: The value of the TT counter at the arrival of the next sync pulse. Or, *None*, if the TT counter was loaded late.

Rtype int

`uptime`()

Returns
Time in FPGA clock ticks since the FPGA was last programmed. Resolution is 2**32 (21 seconds at 200 MHz)

Return type
int

`wait_for_sync`()

Block until a sync has been received.

3.2.4 RFDC Control

The `Rfdc` control interface allows control of the RFSoC's ADCs and DACs.

`class souk_mkid_readout.blocks.rfdc.`**`Rfdc`**(*host*, *name*, *logger=None*, *lmkfile=None*, *lmxfile=None*)

Instantiate a control interface for an RFDC firmware 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.
- **`lmkfile`** (*`str`*) – LMK configuration file to load to board's PLL chip
- **`lmxfile`** (*`str`*) – LMX configuration file to load to board's PLL chip

`get_status`()

Get status and error flag dictionaries.

Status keys:

- lmkfile (str) : The name of the LMK configuration file being used.
- lmxfile (str) : The name of the LMX configuration file being used.

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*)

Parameters

`read_only` (*`bool`*) – If False, initialize the RFDC core and PLL chips. If True, do nothing.

3.2.5 Input Control

`class souk_mkid_readout.blocks.input.`**`Input`**(*host*, *name*, *logger=None*)

`disable_loopback`()

Set pipeline to feed pipeline from ADC inputs

`enable_loopback`()

Set pipeline to internally loop-back DAC stream into ADC.

`get_adc_snapshot`()

Get an ADC snapshot.

Returns

numpy array of complex valued ADC samples

Return type

numpy.ndarray

`get_status()`

Get status and error flag dictionaries.

Status keys:

- loopback (book) : True is system is in internal loopback mode. If True this is flagged with "WARNING".

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*)

Parameters

read_only (*bool*) – If False, disable loopback mode. If True, do nothing.

`loopback_enabled()`

Get the current loopback state.

Returns

True if internal loopback is enabled. False otherwise.

Return type

bool

`plot_adc_snapshot`(*nsamples=None*)

Plot an ADC snapshot.

Parameters

nsamples (*int*) – If provided, only plot this many samples

`plot_adc_spectrum`(*db=False*)

Plot a power spectrum of the ADC input stream using a simple FFT.

Parameters

db (*bool*) – If True, plot in dBs, else linear.

3.2.6 PFB Control

`class souk_mkid_readout.blocks.pfb.Pfb`(*host*, *name*, *logger=None*, *fftshift=4294967295*)

`get_fftshift()`

Get the currently applied FFT shift schedule. The returned value takes into account any hardcoding of the shift settings by firmware.

Returns

Shift schedule

Return type

int

`get_overflow_count()`

Get the total number of FFT overflow events, since the last statistics reset.

Returns

Number of overflows

Return type
int

`get_status()`

Get status and error flag dictionaries.

Status keys:

- overflow_count (int) : Number of FFT overflow events since last statistics reset. Any non-zero value is flagged with "WARNING".
- fftshift (str) : Currently loaded FFT shift schedule, formatted as a binary string, prefixed with "0b".

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*)

Parameters
read_only (*bool*) – If False, set the FFT shift to the default value, and reset the overflow count. If True, do nothing.

`reset_overflow_count()`

Reset overflow counter.

`set_fftshift`(*shift*)

Set the FFT shift schedule.

Parameters
shift (*int*) – Shift schedule to be applied.

3.2.7 PFB TVG Control

class souk_mkid_readout.blocks.pfbtvg.**PfbTvg**(*host*, *name*, *n_inputs=2*, *n_chans=4096*, *n_serial_inputs=1*, *n_rams=2*, *n_samples_per_word=4*, *sample_format='h'*, *logger=None*)

Instantiate a control interface for a post-PFB 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_inputs** (*int*) – Number of independent inputs which may be emulated
- **n_serial_inputs** (*int*) – Number of independent inputs sharing a data bus
- **n_rams** (*int*) – Number of independent bram blocks per input
- **n_samples_per_word** (*int*) – Number of complex samples per word in RAM
- **n_chans** (*int*) – Number of frequency channels.
- **sample_format** (*str*) – Struct type code (eg. 'h' for 16-bit signed) for each of the real/imag parts of the TVG data samples.

`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_input_tvg`(*input*)

Read the test vector loaded to an ADC input.

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

Returns
Test vector array

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_input()`

Write a constant to all the channels of a input, with input *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_input_tvg`(*input*, *test_vector*)

Write a test vector pattern to a single signal input.

Parameters

- **input** (*int*) – Index of input to which test vectors should be loaded.

- `test_vector` (*list or numpy.ndarray*) – *self.n_chans*-element test vector. Values should be representable in 16-bit integer format, and may be complex.

3.2.8 Auto-correlation Control

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

Instantiate a control interface for an Auto-Correlation block. This provides auto-correlation spectra of post-FFT data.

In order to save FPGA resourece, the auto-correlation block may use a single correlation core to compute the auto-correlation of a subset of the total number of ADC channels at any given time. This is the case when the block is instantiated with `n_cores > 1` and `use_mux=True`. In this case, auto-correlation spectra are captured `n_signals / n_cores` channels at a time.

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_signals** (*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.

Variables

n_signals_per_block – Number of signal streams handled by a single correlation core.

get_acc_cnt()

Get the current accumulation count.

Return count
Current accumulation count

Rtype count
int

get_acc_len()

Get the currently loaded accumulation length in units of spectra.

Returns
Current accumulation length

Return type
int

`get_new_spectra`(*signal_block=0*, *flush_vacc='auto'*, *filter_ksize=None*, *return_list=False*)

Get a new average power spectra.

Parameters

- `signal_block` (*int*) – If using multiplexing, read data for this signal block. If not using multiplexing, this parameter does nothing, and data from all inputs will be returned. When multiplexing, Each call will return data for inputs `self.n_signals_per_block x signal_block` to `self.n_signals_per_block x (signal_block+1) - 1`.
- `flush_vacc` (*Bool or string*) – If `True`, throw away a spectra before grabbing a valid one. This can be useful if the upstream analog settings may have changed during the last integration. If `False`, return the first spectra available. If `'auto'` perform a flush if the input multiplexer has changed positions.
- `filter_ksize` (*int*) – If not None, apply a spectral median filter with this kernel size. The kernet size should be odd.
- `return_list` (*Bool*) – If True, return a list else numpy.array

Returns

Float32 2D list of dimensions [POLARIZATION, FREQUENCY CHANNEL] containing autocorrelations with accumulation length divided out.

Return type

list

`get_status`()

Get status and error flag dictionaries.

Status keys:

- acc_len (int) : Currently loaded accumulation length in number of spectra.

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_all_spectra`(*db=True*, *show=True*, *filter_ksize=None*, *adc_srate_mhz=None*)

Plot the spectra of all signals, with accumulation length divided out

Parameters

- `db` (*bool*) – If True, plot 10log10(power). Else, plot linear.
- `show` (*bool*) – If True, call matplotlib's *show* after plotting
- `filter_ksize` (*int*) – If not None, apply a spectral median filter with this kernel size. The kernet size should be odd.
- `adc_srate_mhz` (*float*) – ADC sample rate in MHz. If provided, plot with an appropriate frequency scale on the X-axis.

Returns
matplotlib.Figure

`plot_spectra`(*signal_block=0*, *db=True*, *show=True*, *filter_ksize=None*, *adc_srate_mhz=None*)

Plot the spectra of all signals in a single signal_block, with accumulation length divided out

Parameters

- **signal_block** (*int*) – If using multiplexing, plot data for this signal block. If not using multiplexing, this parameter does nothing, and data from all inputs will be plotted. When multiplexing, Each call will plot data for inputs `self.n_signals_per_block x signal_block` to `self.n_signals_per_block x (signal_block+1) - 1`.
- **db** (*bool*) – If True, plot 10log10(power). Else, plot linear.
- **show** (*bool*) – If True, call matplotlib's *show* after plotting
- **filter_ksize** (*int*) – If not None, apply a spectral median filter with this kernel size. The kernet size should be odd.
- **adc_srate_mhz** (*float*) – ADC sample rate in MHz. If provided, plot with an appropriate frequency scale on the X-axis.

Returns
matplotlib.Figure

`set_acc_len`(*acc_len*)

Set the number of spectra to accumulate.

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

3.2.9 Channel Sub-Select Control

class `souk_mkid_readout.blocks.chanreorder.ChanReorder`(*host*, *name*, *n_chans_in=4096*, *n_chans_out=2048*, *n_parallel_chans_in=4*, *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_in** (*int*) – Number of channels input to the reorder
- **n_chans_out** (*int*) – Number of channels output to the reorder
- **n_parallel_chans_in** (*int*) – Number of channels handled in parallel at the input

`get_channel_outmap`()

Read the currently loaded reorder map.

Returns
The reorder map currently loaded. Entry *i* in this map is the channel number which emerges in the `i`th output position.

Return type
list

`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*.

`set_channel_outmap`(*outmap*)

Remap the channels such that the channel outmap[i] emerges out of the reorder map in position i.

The provided map must be *self.n_chans_out* elements long, else *ValueError* is raised

Parameters

`outmap` (*list of int*) – The outmap to which data should be mapped. I.e., if *outmap[0] = 16*, then the first channel out of the reordr block will be channel 16.

3.2.10 Mixer Control

`class souk_mkid_readout.blocks.mixer.Mixer`(*host*, *name*, *n_chans=4096*, *n_parallel_chans=4*, *phase_bp=31*, *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 this block processes
- `n_parallel_chans` (*int*) – Number of channels this block processes in parallel
- `phase_bp` (*int*) – Number of phase fractional bits

`disable_power_mode`()

Use phase rotation, rather than power.

`enable_power_mode`()

Instead of applying a phase rotation to the data streams, calculate their power.

`get_phase_offset`(*chan*)

Get the currently loaded phase increment being applied to channel *chan*.

Returns

The phase increment being applied to channel *chan* on each successive sample, in radians.

Return type

float

`initialize`(*read_only=False*)

Initialize the block.

Parameters

`read_only` (*bool*) – If True, this method is a no-op. If False, set this block to phase rotate mode, but initialize with each channel having zero phase increment.

`is_power_mode()`

Get the current block mode.

Returns
True if the block is calculating power, False if it is applying phase rotation.

Return type
bool

`set_phase_step`(*chan*, *phase*)

Set the phase increment to apply on each successive sample for channel *chan*.

Parameters

- **chan** (*int*) – The channel index to which this phase-rate should be applied
- **phase** (*float*) – The phase increment to be added each successive sample in units of radians.

3.2.11 Accumulator Control

class `souk_mkid_readout.blocks.accumulator.Accumulator`(*host*, *name*, *acc_len=32768*, *logger=None*, *n_chans=4096*, *n_parallel_chans=8*)

Instantiate a control interface for an Auto-Correlation block. This provides auto-correlation spectra of post-FFT data.

In order to save FPGA resourece, the auto-correlation block may use a single correlation core to compute the auto-correlation of a subset of the total number of ADC channels at any given time. This is the case when the block is instantiated with `n_cores > 1` and `use_mux=True`. In this case, auto-correlation spectra are captured `n_signals / n_cores` channels at a time.

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_parallel_chans** (*int*) – Number of chans processed by the firmware module in parallel.

`get_acc_cnt()`

Get the current accumulation count.

Return count
Current accumulation count

Rtype count
int

`get_acc_len()`

Get the currently loaded accumulation length in units of spectra.

Returns
Current accumulation length

Return type
int

get_new_spectra()

Wait for a new accumulation to be ready then read it.

Returns
Array of *self.n_chans* complex-values.

Return type
numpy.ndarray

get_status()

Get status and error flag dictionaries.

Status keys:

- acc_len (int) : Currently loaded accumulation length in number of spectra.

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(*power=True*, *db=True*, *show=True*)

Plot the spectra of all signals in a single signal_block, with accumulation length divided out

Parameters

- **power** (*bool*) – If True, plot power, else plot complex
- **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

3.2.12 Generator

class souk_mkid_readout.blocks.generator.**Generator**(*host*, *name*, *logger=None*)

initialize(*read_only=False*)

Parameters
read_only (*bool*) – If True, do nothing. If False, reset phase and generator contents

`reset_phase()`

Reset the phase of the output(s).

`set_cordic_output(n, p)`

Set CORDIC output *n* to increment by phase *p* every sample.

Parameters

- **n** (*int*) – Which generator to target.
- **p** (*float*) – phase increment, in units of radians

`set_lut_output(n, x)`

Set LUT output *n* to sample array *x*.

Parameters

- **n** (*int*) – Which generator to target.
- **x** (*list or numpy.array*) – Array (or list) of complex sample values

`set_output_freq(n, freq_mhz, sample_rate_mhz=5000.0, amplitude=1.0)`

Set an output to a CW tone at a specific frequency.

Parameters

- **n** (*int*) – Which generator to target
- **freq_mhz** (*float*) – Output frequency, in MHz
- **sample_rate_mhz** (*float*) – DAC sample rate, in MHz
- **amplitude** (*float*) – Set the output of amplitude of the CW signal. Only applicable for LUT-based generators

3.2.13 Output

`class souk_mkid_readout.blocks.output.Output(host, name, logger=None)`

`get_mode()`

Get the current output mode.

Returns
string describing output mode, eg. "CORDIC"

Rval
str

`get_status()`

Get status and error flag dictionaries.

Status keys:

- mode (str) : 'CORDIC' or 'LUT'

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*)

Parameters
`read_only` (*bool*) – If True, do nothing. If False, initialize to LUT mode.

`use_cordic()`

Set output pipeline to use CORDIC generators

`use_lut()`

Set output pipeline to use LUT generators

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX