LWA Correlator

Release 1.0.0

Jack Hickish

Sep 06, 2021

CONTENTS:

CHAPTER

ONE

INSTALLATION

The LWA 352 Correlator/Beamformer pipeline is available at https://github.com/realtimeradio/caltech-bifrost-dsp. 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-bifrost-dsp
# Clone relevant submodules
cd caltech-bifrost-dsp
git submodule init
git submodule update
```

1.2 Install Prerequisites

The following libraries should be installed via the Ubuntu package manager:

```
apt install exuberant-ctags build-essential autoconf libtool exuberant-ctags libhwloc-
↪dev python3-venv
```

The following 3rd party libraries must also be obtained and installed:

1.2.1 CUDA

CUDA can be installed as follows:

```
# Make a directory for the cuda source
mkdir -p $BUILDDIR/cuda
cd $BUILDDIR/cuda
# Download the CUDA installer
wget http://developer.download.nvidia.com/compute/cuda/11.0.2/local_installers/cuda_
↪11.0.2_450.51.05_linux.run

# blacklist nouveau drivers before installing nvidia drivers
sudo su
echo "blacklist nouveau" > /etc/modprobe.d/blacklist-nouveau.conf
echo "options nouveau modeset=0" >> /etc/modprobe.d/blacklist-nouveau.conf
update-initramfs -u
exit

# reboot the machine and nouveau drivers [hopefully] won't start
# reboot now
```

After rebooting, install the CUDA libraries

```
cd $BUILDDIR/cuda
sudo sh cuda_11.0.2_450.51.05_linux.run
# Add CUDA executables to $PATH
echo "export PATH=/usr/local/cuda/bin:${PATH}" >> ~/.bashrc
source ~/.bashrc
```

This CUDA install script will take a minute to unzip and run the installer. If it fails, log messages can be found in `/var/log/nvidia-installer.log` and `/var/log/cuda-installer.log`.

1.2.2 IB Verbs

The LWA pipeline uses Infiniband Verbs for fast UDP packet capture. The recommended version is 4.9 LTS. This can be obtained from https://www.mellanox.com/support/mlnx-ofed-matrix?mtag=linux_sw_drivers

1.2.3 xGPU

xGPU is submoduled in the main pipeline repository, to ensure version compatibility. Install with:

```
cd $BUILDDIR/caltech-bifrost-dsp
./install_xgpu
```

1.2.4 Bifrost

Bifrost is submoduled in the main pipeline repository, to ensure version compatibility.

The version provided requires Python >=3.5. It is recommended that the bifrost package is installed within a Python version environment.

To install bifrost:

```
cd $BUILDDIR/caltech-bifrost-dsp/bifrost
make
make install
```

1.3 Install the Pipeline

After installing the prerequisites above, the LWA pipeline can be installed with

```
cd $BUILDDIR/caltech-lwa-dsp/pipeline
# Be sure to run the installation in the
# appropriate python environment!
python setup.py install
```

Author Jack Hickish

CHAPTER TWO

INTRODUCTION

The LWA-352 X-Engine processing system performs correlation, beamforming, and triggered voltage recording for the LWA-352 array. This document outlines the hardware (Section [sec:hardware]) and software (Section [sec:software]) which makes up the X-Engine, and details the user control interface (Section [sec:api]).

CHAPTER THREE

HARDWARE

The LWA-352 X-engine system comprises 9 1U dual-socket Silicon Mechanics *Rackform R353.v7* servers, each hosting a pair of Nvidia GPUs and solid state memory buffers. Hardware specifications are given in Table [tab:hardware].

Hardware	Model	Notes
Server	Supermicro 1029GQ-TRT	Re-branded as Silicon Mechanics Rackform R353.v7
Motherboard	Supermicro X11 DCQ	
CPU	dual Intel Xeon Scalable Silver 4210R	2.4 GHz, 10 core, 100W TDP
RAM	768 GB PC4-23400	12 x 64 GB; 2933 MHz DDR4; ECC RDIMM
NIC	Mellanox MCX515A-GCAT	ConnectX-5 EN MCX515A-GCAT (1x QSFP28); PCIe 3.0x16
NVMe Controllers	Asus Hyper M.2 X16 Card V2	2 cards per server
NVMe Memory	8TB Samsung 970 Evo Plus	8 x 1 TB
GPU	Nvidia RTX 2080Ti	2 cards per server

CHAPTER

FOUR

PIPELINE

The pipeline is launched using the `lwa352-pipeline.py` script. This has the following options:

```
usage: lwa352-pipeline.py [-h] [-l LOGFILE] [-v] [--nchan NCHAN]
                          [--fakesource] [--nodata] [--testdatain TESTDATAIN]
                          [--testdatacorr TESTDATACORR]
                          [--testdatacorr_acc_len TESTDATACORR_ACC_LEN]
                          [-a CORR_ACC_LEN] [--nocorr] [--nobeamform]
                          [--nogpu] [--ibverbs] [-G GPU] [-P PIPELINEID]
                          [-C CORES] [-q] [--testcorr] [--useetcd]
                          [--pycorrout] [--etcdhost ETCDHOST] [--ip IP]
                          [--port PORT] [--bufgbytes BUFGBYTES]
                          [--cor_npipeline COR_NPIPELINE]
                          [--target_throughput TARGET_THROUGHPUT]

LWA352-OVRO Correlator-Beamformer Pipeline

optional arguments:
  -h, --help            show this help message and exit
  -l LOGFILE, --logfile LOGFILE
                        Specify log file (default: None)
  -v, --verbose         Increase verbosity (default: 0)
  --nchan NCHAN         Number of frequency channels in the pipeline (default:
                        192)
  --fakesource          Use a dummy source for testing (default: False)
  --nodata              Don't generate data in the dummy source (faster)
                        (default: False)
  --testdatain TESTDATAIN
                        Path to input test data file (default: None)
  --testdatacorr TESTDATACORR
                        Path to correlator output test data file (default:
                        None)
  --testdatacorr_acc_len TESTDATACORR_ACC_LEN
                        Number of accumulations per sample in correlator test
                        data file (default: 2400)
  -a CORR_ACC_LEN, --corr_acc_len CORR_ACC_LEN
                        Number of accumulations to start accumulating in the
                        slow correlator (default: 240000)
  --nocorr              Don't use correlation threads (default: False)
  --nobeamform          Don't use beamforming threads (default: False)
  --nogpu               Don't use any GPU threads (default: False)
  --ibverbs             Use IB verbs for packet capture (default: False)
  -G GPU, --gpu GPU     Which GPU device to use (default: 0)
  -P PIPELINEID, --pipelineid PIPELINEID
                        Pipeline ID. Useful if you are running multiple
```

(continues on next page)

(continued from previous page)

```
                        pipelines on a single machine (default: 0)
  -C CORES, --cores CORES
                        Comma-separated list of CPU cores to use (default:
                        0,1,2,3,4,5,6,7)
  -q, --quiet           Decrease verbosity (default: 0)
  --testcorr            Compare the GPU correlation with CPU. SLOW!! (default:
                        False)
  --useetcd             Use etcd control/monitoring server (default: False)
  --pycorrout           Don't use CORR output packets, use custom format
                        (default: False)
  --etcdhost ETCDHOST   Host serving etcd functionality (default: etcdhost)
  --ip IP               IP address to which to bind (default: 100.100.100.101)
  --port PORT           UDP port to which to bind (default: 10000)
  --bufgbytes BUFGBYTES
                        Number of GBytes to buffer for transient buffering
                        (default: 4)
  --cor_npipeline COR_NPIPELINE
                        Number of pipelines per COR packet output stream
                        (default: 2)
  --target_throughput TARGET_THROUGHPUT
                        Target throughput when using --fakesource (default:
                        1000.0)
```

CHAPTER

FIVE

SOFTWARE

The LWA-352 pipeline comprises 13 independent processes, briefly described below.

1. `capture`: Receive F-engine packets and correctly arrange in buffers for downstream processing. Monitor and record missing packets and network performance statistics.
2. `copy1`: Transfer blocks of data from the capture buffer to a deep transient buffer.
3. `copy2`: Transfer blocks of data from CPU to GPU, for high-performance computation.
4. `triggered_dump`: Process software triggers to copy deep-buffered data to disk.
5. `corr`: Correlate data using the `xGPU` library and accumulate for short (~100ms) durations.
6. `corr_subsel`: Down-select a sub-set of the complete visibility matrices.
7. `corr_output_part`: Output subselected visibilities as UDP/IP streams.
8. `corr_acc`: Further accumulate correlation output to ~second durations.
9. `corr_output_full`: Output full, accumulated visibility matrices.
10. `beamform`: Form multiple voltage beams.
11. `beamform_vlbi_output`: Package and transmit voltage beam(s) for VLBI purposes.
12. `beamform_sum_beams`: Form integrated power-spectra for multiple beams.
13. `beamform_output`: Output accumulated power beams.

5.1 High-Level Parameters

- `NBEAM`: Number of dual-polarization beams to form.
- `CHAN_PER_PACKET`: Number of frequency channels in an F-engine packet. This should be chosen to match the F-engine configuration.
- `NPIPELINE`: The total number of pipelines in the multi-server correlator system.
- `NSNAP`: The number of SNAP2 boards in the total F-engine system.
- `GSIZE`: "Gulp size" – the number of samples processed per batch by a processing block.
- `NETGSIZE`: Number of time samples buffered in a capture buffer block.
- `NET_NGULP`: Number of `NETGSIZE` buffers comprising a single buffer "time slot". Data capture always begins on a time boundary of `NETGSIZE*NET_NGULP`, and downstream processes should begin processing on boundaries consistent with this.

5.2 Bifrost Block Description

The bifrost pipelining framework divides streams of data into *sequences*, each of which has a header describing the stream. Different processing blocks act to perform operations on streams, and may modify sequences and their headers.

Here we summarize the bifrost blocks in the LWA-352 pipeline, and the headers they require (for input sequences) and provide (for output sequences).

5.2.1 `capture`

class Capture(*log*, *fs_hz=196000000*, *chan_bw_hz=23925.78125*, *input_to_ant=None*, *nstand=352*, *npol=2*, *system_nchan=2912*, **args*, ***kwargs*)

Functionality

This block receives UDP/IP data from an Ethernet network and writes it to a bifrost memory buffer.

New Sequence Condition

This block starts a new sequence each time the incoming packet stream timestamp changes in an unexpected way. For example, if a large block of timestamps are missed a news sequence will be started. Or, if the incoming timestamps decrease (which might happen if the upstream transmitters are reset) a new sequence is started.

Input Header Requirements

This block is a bifrost source, and thus has no input header requirements.

Output Headers

Output header fields are as follows:

Field	Format	Units	Description
`time_tag`	int		Arbirary integer, incremented with each new sequence.
`sync_time`	int	UNIX seconds	Synchronization time (corresponding to spectrum sequence number 0)
`seq0`	int		Spectra number for the first sample in this sequence
`chan0`	int		Channel index of the first channel in this sequence
`nchan`	int		Number of channels in the sequence
`system_nchan`	int		The total number of channels in the system (i.e., the number of channels across all pipelines)
`fs_hz`	double	Hz	Sampling frequency of ADCs
`sfreq`	double	Hz	Center frequency of first channel in the sequence
`bw_hz`	int	Hz	Bandwidth of the sequence
`nstand`	int		Number of stands (antennas) in the sequence
`npol`	int		Number of polarizations per stand in the sequence
`complex`	bool		True if the data are complex, False otherwise
`nbit`	int		Number of bits per sample (or per real/imag part if the samples are complex)
`input_to_ant`	list[int]		List of input to stand/pol mappings with dimensions `[nstand x npol, 2]`. E.g. if entry `N` of this list has value `[S, P]` then the `N`-th correlator input is stand `S`, polarization `P`.
`ant_to_input`	list[ints]		List of stand/pol to correlator input number mappings with dimensions `[nstand, npol]`. E.g. if entry `[S,P]` of this list has value `N` then stand `S`, polarization `P` of the array is the `N`-th correlator input

Data Buffers

Input data buffer: None

Output data buffer: Complex 4-bit data with dimensions (slowest to fastest) `Time x Freq x Stand x Polarization`

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **fs_hz** (*int*) – Sampling frequency, in Hz, of the upstream ADCs
- **chan_bw_hz** (*float*) – Bandwidth of a single frequency channel in Hz.
- **nstand** (*int*) – Number of stands in the array.
- **npol** (*int*) – Number of polarizations per antenna stand.
- **input_to_ant** – An map of correlator input to station / polarization. Provided as an `[nstand x npol, 2]` array such that if `input_to_ant[i] == [S,P]` then the i-th correlator input is stand `S`, polarization `P`.

Keyword Arguments

Parameters

- **fmt** (*string*) – The string identifier of the packet format to be received. E.g "snap2".
- **sock** (*bifrost.udp_socket.UDPSocket*) – Input UDP socket on which to receive.
- **ring** (*bifrost.ring.Ring*) – bifrost output data ring
- **core** (*int*) – CPU core to which this block should be bound.
- **nsrc** (*int*) – Number of packet sources. This might mean the number of boards transmitting packets, or, in the case that it takes multiple packets from each board to send a complete set of data, this could be a multiple of the number of source boards.
- **src0** (*int*) – The first source to transmit to this block.
- **system_nchan** (*int*) – The total number of channels in the complete, multi-pipeline system. This is only used to set sequence headers for downstream packet header generators which require this information.
- **max_payload_size** (*int*) – The maximum payload size, in bytes, of the UDP packets to be received.
- **buffer_ntime** (*int*) – The number of time samples to be buffered into the output data ring buffer before it is marked full.
- **utc_start** (*datetime.datetime*) – ?The time at which the block should begin receiving. Set to datetime.datetime.now() to start immediately.
- **ibverbs** (*Bool*) – Boolean parameter which, if true, will cause this block to use an Infiniband Verbs packet receiver. If false, or not provided, a standard UDP socket will be used.

5.2.2 copy

class Copy(*log*, *iring*, *oring*, *ntime_gulp=2500*, *buffer_multiplier=1*, *guarantee=True*, *core=- 1*, *nbyte_per_time=129536*, *gpu=- 1*, *buf_size_gbytes=None*)

Functionality

This block copies data from one bifrost buffer to another. The buffers may be in CPU or GPU space.

New Sequence Condition

This block has no new sequence condition. It will output a new sequence only when the upstream sequence changes.

Input Header Requirements

This block has no input header requirements.

Output Headers

This block copies input headers downstream, adding none of its own.

Data Buffers

Input Data Buffer: A bifrost ring buffer of at least `nbyte_per_time x ntime_gulp` bytes size.

Output Data Buffer: A bifrost ring buffer whose size is set by the `buffer_multiplier` and/or `buf_size_gbytes` parameters.

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring
- **oring** (*bifrost.ring.Ring*) – bifrost output data ring
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **gpu** (*int*) – GPU device ID to which this block should copy to/from. A value of -1 indicates no binding. This parameter need not be provided if neither input nor output data rings are allocated in GPU (or GPU-pinned) memory.
- **ntime_gulp** (*int*) – Number of time samples to copy with each gulp.
- **nbyte_per_time** (*int*) – Number of bytes per time sample. The total number of bytes read with each gulp is `nbyte_per_time x ntime_gulp`.
- **buffer_multiplier** (*int*) – The block will set the output buffer size to be `4 x buffer_multiplier` times the size of a single data gulp. If `buf_size_gbytes` is also provided then the output memory block size is required to be a mulitple of `buffer_multiplier x ntime_gulp x nbyte_per_time` bytes.
- **buf_size_gbytes** (*int*) – If provided, attempt to set the output buffer size to `buf_size_gbytes` Gigabytes (10**9 Bytes). Round down the buffer such that the chosen size is the largest integer multiple of `buffer_multiplier x ntime_gulp x nbyte_per_time` which is less than `10**9 x buf_size_gbytes`. Note that bifrost (seems to) round up buffer sizes to an integer power of two number of bytes, so be careful about accidentally allocating more memory than you have!

5.2.3 triggered_dump

class TriggeredDump(*log*, *iring*, *ntime_gulp=2500*, *ntime_per_file=1000000*, *guarantee=True*, *core=-1*, *nbyte_per_time=135168*, *etcd_client=None*, *dump_path='/tmp'*)

Functionality

This block writes data from an input bifrost ring buffer to disk, when triggered.

New Sequence Condition

This block is a bifrost sink, and generates no output sequences.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
`seq 0`	int		Spectra number for the first sample in the input sequence

The field `seq` if provided by the upstream block will be overwritten.

Output Headers

This block is a bifrost sink, and generates no output headers. Headers provided by the upstream block are written to this block's data files, with the exception of the `seq` field, which is overwritten.

Data Buffers

Input Data Buffer: A bifrost ring buffer of at least `nbyte_per_time x ntime_gulp` bytes size.

Output Data Buffer: None

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **ntime_gulp** (*int*) – Number of time samples to copy with each gulp.
- **nbyte_per_time** (*int*) – Number of bytes per time sample. The total number of bytes read with each gulp is `nbyte_per_time x ntime_gulp`.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.
- **dump_path** – Root path to directory where dumped data should be stored. This parameter can be overridden by runtime control commands.
- **ntime_per_file** (*int*) – Number of time samples of data to write to each output file. This parameter can be overridden by runtime control commands.

Runtime Control and Monitoring

This block accepts the following command fields:

Field	Format	Units	Description
`command`	string		Commands: `Trigger`: Begin capturing data ASAP `Abort`: Abort a capture currently in progress and delete its data `Stop`: Stop a capture currently in progress
`ntime_per_sample`	int	samples	Number of time samples to capture in each file.
`nfile`	int		Number of files to capture per trigger event.
`dump_path`	str		Root path to directory where data should be stored.

Output Data Format

When triggered, this block will output a series of `nfile` data files, each containing `ntime_per_file` time samples of data.

File names begin `lwa-dump-` and are suffixed by the trigger time as a floating-point UNIX timestamp with two decimal points of precision, a file extension ".tbf", and a file number ".N" where N runs from 0 to `nfile - 1`. For example: `lwa-dump-1607434049.77.tbf.0`.

The files have the following structure:

- The first 4 bytes contains a little-endian 32-bit integer, `hsize`, describing the number of bytes of subsequent header data.
- The following 4 bytes contains a little-endian 32-bit integer, `hblock_size` describing the size of header block which precedes the data payload in the file.
- Bytes `8` to `8 + hsize` contain the json-encoded header of the bifrost sequence which is contained in the payload of this file, with an additional `seq` keyword, which indicates the spectra number of the first spectra in this data file.
- Bytes `hblock_size` to `EOF` contain data in the shape and format of the input bifrost data buffer.

Output data files can thus be read with:

```
import struct
with open(FILENAME, "rb") as fh:
    hsize = struct.unpack('<I', fh.read(4))
    hblock_size = struct.unpack('<I', fh.read(4))
    header = json.loads(fh.read(hsize))
    fh.seek(hblock_size)
    data = fh.read() # read to EOF
```

5.2.4 corr

class Corr(*log*, *iring*, *oring*, *ntime_gulp=2500*, *guarantee=True*, *core=- 1*, *nchan=192*, *npol=2*, *nstand=352*, *acc_len=2400*, *gpu=- 1*, *test=False*, *etcd_client=None*, *autostartat=0*, *ant_to_input=None*)

Functionality

This block reads data from a GPU-side bifrost ring buffer and feeds it to xGPU for correlation, outputing results to another GPU-side buffer.

New Sequence Condition

This block starts a new sequence each time a new integration configuration is loaded or the upstream sequence changes.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
`seq0`	int		Spectra number for the first sample in the input sequence

Output Headers

This block passes headers from the upstream block with the following modifications:

Field	Format	Units	Description
`seq0`	int		Spectra number for the *first* sample in the integrated output
`acc_len`	int		Number of spectra integrated into each output sample by this block
`ant_to_input`	list of ints		This header is removed from the sequence
`input_to_ant`	list of ints		This header is removed from the sequence

Data Buffers

Input Data Buffer: A GPU-side bifrost ring buffer of 4+4 bit complex data in order: `time x channel x stand x polarization`.

Each gulp of the input buffer reads `ntime_gulp` samples, which should match *both* the xGPU compile-time parameters `NTIME` and `NTIME_PIPE`.

Output Data Buffer: A GPU-side bifrost ring buffer of 32+32 bit complex integer data. This buffer is in the xGPU triangular matrix order: `time x channel x complexity x baseline`.

The output buffer is written in single accumulation blocks (an integration of `acc_len` input time samples).

Instantiation

Parameters

- **`log`** (*`logging.Logger`*) – Logging object to which runtime messages should be emitted.
- **`iring`** (*`bifrost.ring.Ring`*) – bifrost input data ring. This should be on the GPU.
- **`oring`** (*`bifrost.ring.Ring`*) – bifrost output data ring. This should be on the GPU.
- **`guarantee`** (*`Bool`*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **`core`** (*`int`*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **`gpu`** (*`int`*) – GPU device which this block should target. A value of -1 indicates no binding
- **`ntime_gulp`** (*`int`*) – Number of time samples to copy with each gulp.
- **`nchan`** (*`int`*) – Number of frequency channels per time sample. This should match the xGPU `NFREQUENCY` compile-time parameter.
- **`nstand`** (*`int`*) – Number of stands per time sample. This should match the xGPU `NSTATION` compile-time parameter.
- **`npol`** (*`int`*) – Number of polarizations per stand. This should match the xGPU `NPOL` compile-time parameter.

- **acc_len** (*int*) – Accumulation length per output buffer write. This should be an integer multiple of the input gulp size `ntime_gulp`. This parameter can be updated at runtime.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.
- **test** (*Bool*) – If True, run a CPU correlator in parallel with xGPU and verify the output. Beware, the (Python!) CPU correlator is *very* slow.
- **autostartat** (*int*) – The start time at which the correlator should automatically being correlating without intervention of the runtime control system. Use the value `-1` to cause integration to being on the next gulp.
- **ant_to_input** (*nstand x npol list of ints*) – an [nstand, npol] list of input IDs used to map stand/polarization `S`, `P` to a correlator input. This allows the block to pass this information to downstream processors. *This functionality is currently unused*

Runtime Control and Monitoring

This block accepts the following command fields:

Field	Format	Units	Description
`acc_len`	int	samples	Number of samples to accumulate. This should be a multiple of `ntime_gulp`
`start_time`	int	samples	The desired first time sample in an accumulation. This should be a multiple of `ntime_gulp`, and should be related to GPS time through external knowledge of the spectra count origin (aka SNAP *sync time*). The special value `-1` can be used to force an immediate start of the correlator on the next input gulp.

5.2.5 `corr_acc`

class CorrAcc(*log*, *iring*, *oring*, *guarantee=True*, *core=- 1*, *nchan=192*, *npol=2*, *nstand=352*, *acc_len=24000*, *gpu=- 1*, *etcd_client=None*, *autostartat=0*)

Functionality

This block reads data from a GPU-side bifrost ring buffer and accumulates it in an internal GPU buffer. The accumulated data are then copied to an output ring buffer.

New Sequence Condition

This block starts a new sequence each time a new integration configuration is loaded or the upstream sequence changes.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
`seq0`	int		Spectra number for the first sample in the input sequence
`acc_len`	int		Number of spectra integrated into each output sample by the upstream processing

Output Headers

This block passes headers from the upstream block with the following modifications:

Field	Format	Units	Description
`seq0`	int		Spectra number for the *first* sample in the integrated output
`acc_len`	int		Total number of spectra integrated into each output sample by this block, incorporating any upstream processing
`upstream_acc_len`	int		Number of spectra already integrated by upstream processing

Data Buffers

Input Data Buffer: A GPU-side bifrost ring buffer of 32+32 bit complex integer data. The input buffer is read in gulps of `nchan * (nstand//2+1)*(nstand//4)*npol*npol*4*2` 32-bit words, which is the appropriate size if this block is fed by an upstream `Corr` block.

Note that if the upstream block is ``Corr``, the complexity axis of the input buffer is not the fastest changing.

Output Data Buffer: A bifrost ring buffer of 32+32 bit complex integer data of the same ordering and dimensionality as the input buffer.

The output buffer is written in single accumulation blocks (an integration of `acc_len` input vectors).

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **oring** (*bifrost.ring.Ring*) – bifrost output data ring. This should be on the GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **gpu** (*int*) – GPU device which this block should target. A value of -1 indicates no binding
- **nchan** (*int*) – Number of frequency channels per time sample.
- **nstand** (*int*) – Number of stands per time sample.
- **npol** (*int*) – Number of polarizations per stand.
- **acc_len** (*int*) – Accumulation length per output buffer write. This should be an integer multiple of any upstream accumulation. This parameter can be updated at runtime.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.
- **autostartat** (*int*) – The start time at which the correlator should automatically being correlating without intervention of the runtime control system. Use the value `-1` to cause integration to being on the next gulp.

Runtime Control and Monitoring

This block accepts the following command fields:

Field	Format	Units	Description
acc_len	int	samples	Number of samples to accumulate. This should be a multiple of any upstream accumulation performed by other blocks. I.e., it should be an integer multiple of a sequences acc_len header entry.
start_time	int	samples	The desired first time sample in an accumulation. This should be compatible with the accumulation length and start time of upstream blocks. I.e. it should be offset from the input sequence header's seq0 value by an integer multiple of the input sequence header's acc_len value

5.2.6 corr_output_full

class CorrOutputFull(*log*, *iring*, *guarantee=True*, *core=- 1*, *nchan=192*, *npol=2*, *nstand=352*, *etcd_client=None*, *dest_port=10000*, *checkfile=None*, *checkfile_acc_len=1*, *antpol_to_bl=None*, *bl_is_conj=None*, *use_cor_fmt=True*, *npipeline=1*)

Functionality

Output an xGPU-spec visibility buffer as a stream of UDP packets.

New Sequence Condition

This block is a bifrost sink, and generates no downstream sequences.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
seq0	int		Spectra number for the first sample in the input sequence
acc_len	int		Number of spectra integrated into each output sample by upstream processing
nchan	int		The number of frequency channels in the input visibility matrices
chan0	int		The index of the first frequency channel in the input visibility matrices
npol	int		The number of polarizations per stand in the input visibility matrices
bw_hz	double	Hz	Bandwidth of the input visibility matrices.
fs_hz	int	Hz	ADC sample rate. Only required if use_cor_fmt=True
sfreq	double	Hz	Center frequency of the first channel in the input visibility matrices. Only required if use_cor_fmt=False.

Optional header fields, which describe the input xGPU buffer contents. If not supplied as headers, these should be provided as keyword arguments when this block is instantiated.

Field	Format	Units	Description
`ant_to_ bl_id`	list of int		A 4D list of integers, with dimensions `[nstand, nstand, npol, npol]` which maps the correlation of `stand0, pol0` with `stand1, pol1` to visibility index `[stand0, stand1, pol0, pol1]`
`bl_is_conj`	list of bool		A 4D list of boolean values, with dimensions `[nstand, nstand, npol, npol]` which indicates if the correlation of `stand0, pol0` with `stand1, pol1` has the first (`stand0, pol0`) or second (`stand1, pol1`) input conjugated. If `bl_id_conj[stand0, stand1, pol0, pol1]` has the value `True`, then `stand0, pol0` is the conjugated input.

Output Headers

This is a bifrost sink block, and provides no data to an output ring.

Data Buffers

Input Data Buffer: A CPU-side bifrost ring buffer of 32+32 bit complex integer data. This input buffer is read in gulps of `nchan * (nstand//2+1)*(nstand//4)*npol*npol*4*2` 32-bit words, which is the size of an xGPU visibility matrix.

Output Data Buffer: This block has no output data buffer.

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **nchan** (*int*) – Number of frequency channels per time sample.
- **nstand** (*int*) – Number of stands per time sample.
- **npol** (*int*) – Number of polarizations per stand.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.
- **dest_port** (*int*) – Default destination port for UDP data. Can be overriden with the runtime control interface.
- **use_cor_fmt** (*Bool*) – If True, use the LWA `COR` packet output format. Otherwise use a custom format. See *Output Format*, below.
- **antpol_to_bl** (*4D list of int*) – Map of antenna/polarization visibility intputs to xGPU output indices. See optional sequence header entry `ant_to_bl_id`. If not provided, this map should be available as a bifrost sequence header.
- **bl_is_conj** (*4D list of bool*) – Map of visibility index to conjugation convention. See optional sequence header entry `bl_is_conj`. If not provided, this map should be available as a bifrost sequence header.

- **checkfile** (*str*) – Path to a data file containing the expected correlator output. If provided, the data input to this block will be checked against this file. The data file should contain binary numpy.complex-format data in order `time x nchan x nstand x nstand x npol x npol`. Each entry in this data file should represent an expected visibility which has been integrated for `checkfile_acc_len` samples. This file can be generated with this package's `make_golden_inputs.py` script.
- **checkfile_acc_len** (*int*) – The number of integrations which have gone into each time slice of the provided `checkfile`. For a check to be run, the accumulation length (and accumulation starts) of data input to this block should be a multiple of `checkfile_acc_len`.
- **npipeline** (*int*) – The number of pipelines in the system, as written to output COR packets. This may or may not be the same as the number of actual pipelines present. The one-index of this block's pipeline is computed from sequence header values as `((chan0 // nchan) % npipeline) + 1`.

Runtime Control and Monitoring

Field	Format	Units	Description
`dest_ip`	string		Destination IP for transmitted packets, in dotted-quad format. Eg. `"10.0.0.1"`. Use `"0.0.0.0"` to skip sending packets
`dest_file`	string		If not "", overrides `dest_ip` and causes the output data to be written to the supplied file
`dest_port`	int		UDP port to which packets should be transmitted.
`max_mbps`	int	Mbits/s	The maximum output data rate to allow before throttling. Set to `-1` to send as fast as possible.

Output Data Format

Each packet from the correlator contains data from multiple channels for a single, dual-polarization baseline. There are two possible output formats depending on the value of `use_cor_fmt` with which this block is instantiated.

If `use_cor_fmt=True`, this block outputs packets conforming to the LWA-SV "COR" spec (though with integer rather than floating point data). This format comprises a stream of UDP packets, each with a 32 byte header defined as follows:

```
struct cor {
  uint32_t  sync_word;
  uint8_t   id;
  uint24_t  frame_number;
  uint32_t  secs_count;
  int16_t   freq_count;
  int16_t   cor_gain;
  int64_t   time_tag;
  int32_t   cor_navg;
  int16_t   stand_i;
  int16_t   stand_j;
  int32_t   data[nchans, npols, npols, 2];
};
```

Packet fields are as follows:

Field	Format	Units	Description
sync_word	uint32		Mark 5C magic number, `0xDEC0DE5C`
id	uint8		Mark 5C ID, used to identify COR packet, `0x02`
frame_number	uint24		Mark 5C frame number. Unused.
secs_count	uint32		Mark 5C seconds since 1970-01-01 00:00:00 UTC. Unused.
freq_count	int16		zero-indexed frequency channel ID of the first channel in the packet.
cor_gain	int16		Right bitshift used for gain compensation. Unused.
time_tag	int64	ADC sample period	Central sampling time since 1970-01-01 00:00:00 UTC.
cor_navg	int16	TODO: sub-slots doesn't work	Integration time.
stand_i	int16		1-indexed stand number of the unconjugated stand.
stand_j	int16		1-indexed stand number of the conjugated stand.
data	int32*		The data payload. Data for the visibility of antennas at stand_i and stand_j, with stand_j conjugated. Data are a multidimensional array of 32-bit integers, with dimensions `[nchans, npols, npols, 2]`. The first axis is frequency channel. The second axis is the polarization of the antenna at stand_i. The second axis is the polarization of the antenna at stand_j.| The fourth axis is complexity, with index 0 the real part of the visibility, and index 1 the imaginary part.

If `use_cor_fmt=False`, this block outputs a stream of UDP packets, with each comprising a 56 byte header followed by a payload of signed 32-bit integers. The packet definition is as follows:

```
struct corr_output_full_packet {
  uint64_t  sync_time;
  uint64_t  spectra_id;
  double    bw_hz;
  double    sfreq_hz;
  uint32_t  acc_len;
  uint32_t  nchans;
  uint32_t  chan0;
  uint32_t  npols;
  uint32_t  stand0;
  uint32_t  stand1;
  int32_t   data[npols, npols, nchans, 2];
};
```

Packet fields are as follows:

Field	Format	Units	Description
sync_time	uint64	UNIX seconds	The sync time to which spectra IDs are referenced.
spectra_id	int		The spectrum number for the first spectra which contributed to this packet's integration.
bw_hz	double (binary64)	Hz	The total bandwidth of data in this packet
sfreq_hz	double (binary64)	Hz	The center frequency of the first channel of data in this packet
acc_len	uint32		The number of spectra integrated in this packet
nchans	uint32		The number of frequency channels in this packet. For LWA-352 this is 184
chan0	uint32		The index of the first frequency channel in this packet
npols	uint32		The number of polarizations of data in this packet. For LWA-352, this is 2.
stand0	uint32		The index of the first antenna stand in this packet's visibility.
stand1	uint32		The index of the second antenna stand in this packet's visibility.
data	int32*		The data payload. Data for the visibility of antennas at stand0 and stand1, with stand1 conjugated. Data are a multidimensional array of 32-bit integers, with dimensions [`npols`, `npols`, `nchans`, 2]. The first axis is the polarization of the antenna at stand0. The second axis is the polarization of the antenna at stand1. The third axis is frequency channel. The fourth axis is complexity, with index 0 the real part of the visibility, and index 1 the imaginary part.

5.2.7 `corrsubsel`

class CorrSubsel(*log*, *iring*, *oring*, *guarantee=True*, *core=- 1*, *etcd_client=None*, *nchan=192*, *npol=2*, *nstand=352*, *nchan_sum=4*, *gpu=- 1*, *antpol_to_bl=None*, *bl_is_conj=None*)

Functionality

This block selects individual visibilities from an xGPU buffer, averages them in frequency, and rearranges them into a sane format.

New Sequence Condition

This block starts a new sequence each time a new baseline selection is loaded or if the upstream sequence changes.

Input Header Requirements

This block requires the following headers from upstream:

Field	Format	Units	Description
seq0	int	•	Spectra number for the first sample in the input sequence
acc_len	int	•	Number of spectra accumulated per input data sample
nchan	int	•	Number of channels in the input sequence
bw_hz	double	Hz	Bandwith of the input sequence
sfreq	double	Hz	Center frequency of first channel in the input sequence

Output Headers

This block copies headers from upstream with the following modifications:

Field	Format	Units	Description
seq0	int	•	Spectra number for the first sample in the output sequence. This may diverge from the input sequence `seq0`.
nchan	int	•	Number of frequency channels in the output data stream, after any integration performed by this block
nvis	int	•	Number of visibilities in the output data stream
nchan_sum	int	•	Number of frequency channels summed by this block
baselines	list of ints	•	A list of output stand/pols, with dimensions `[nvis, 2, 2]`. E.g. if entry [V] of this list has value `[[N_0, P_0], [N_1, P_1]]` then the `V`-th entry in the output data array is the correlation of stand `N_0`, polarization `P_0` with stand `N_1`, polarization `P_1`
bw_hz	double	Hz	Bandwith of the output sequence, after averaging
sfreq	double	Hz	Center frequency of first channel in the output sequence, after averaging

Data Buffers

Input Data Buffer: A GPU-side bifrost ring buffer of 32+32 bit complex integer data. The input buffer is read in gulps of `nchan * (nstand//2+1)*(nstand//4)*npol*npol*4*2` 32-bit words, which is the appropriate size if this block is fed by an upstream `Corr` block.

Note that if the upstream block is ``Corr``, the complexity axis of the input buffer is not the fastest changing.

Output Data Buffer: A bifrost ring buffer of 32+32 bit complex integer data. The output buffer may be in GPU or CPU memory, and has dimensions `time x frequency channel x visibility x complexity`. The output buffer is written in blocks of `nchan // nchan_sum x nvis_out[=4704]` 64-bit words.

Instantiation

Parameters

- **`log`** (*`logging.Logger`*) – Logging object to which runtime messages should be emitted.

- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **oring** (*bifrost.ring.Ring*) – bifrost output data ring. This may be on the CPU or GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **nchan** (*int*) – Number of frequency channels per time sample.
- **nstand** (*int*) – Number of stands per time sample.
- **npol** (*int*) – Number of polarizations per stand.
- **nchan_sum** (*int*) – Number of frequency channels to sum together when generating output.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.
- **antpol_to_bl** (*4D list of int*) – Map of antenna/polarization visibility intputs to xGPU output indices. See optional sequence header entry `ant_to_bl_id`.
- **bl_is_conj** (*4D list of bool*) – Map of visibility index to conjugation convention. See optional sequence header entry `bl_is_conj`.

Runtime Control and Monitoring

Field	Format	Units	Description
`baselines`	3D list of int		A list of baselines for subselection. This field should be provided as a multidimensional list with dimensions `[nvis, 2, 2]`. The first axis runs over the 4704 baselines which may be selected. The second index is 0 for the first (unconjugated) input selected and 1 for the second (conjugated) input selected. The third axis is 0 for stand number, and 1 for polarization number.

Example

To set the baseline subsection to choose:

- visibility 0: the autocorrelation of antenna 0, polarization 0
- visibility 1: the cross correlation of antenna 5, polarization 1 with antenna 6, polarization 0

use:

```
subsel = [ [[0,0], [0,0]], [[5,1], [6,0]], ... ]
```

Note that the uploaded selection list must always have 4704 entries.

5.2.8 corr_output_part

class CorrOutputPart(*log*, *iring*, *use_cor_fmt=False*, *guarantee=True*, *core=- 1*, *etcd_client=None*, *dest_port=10001*, *nvis_per_packet=16*, *npipeline=1*)

Functionality

Output a block of visibilities as a UDP data stream

New Sequence Condition

This block is a bifrost sink, and generates no downstream sequences.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
`seq0`	int		Spectra number for the first sample in the input sequence
`acc_len`	int		Number of spectra integrated into each output sample by upstream processing
`nchan`	int		The number of frequency channels in the input visibility matrices
`nchan_sum`	int		The number of frequency channels summed by upstream processing
`chan0`	int		The index of the first frequency channel in the input visibility matrices
`npol`	int		The number of polarizations per stand in the input visibility matrices
`bw_hz`	double	Hz	Bandwidth of the input visibility matrices.
`sfreq`	double	Hz	Center frequency of the first channel in the input visibility matrices. Only required if `use_cor_fmt=False`.
`fs_hz`	int	Hz	ADC sample rate. Only required if `use_cor_fmt=True`
`nvis`	int	•	Number of visibilities in the output data stream
`baselines`	list of ints	•	A list of output stand/pols, with dimensions `[nvis, 2, 2]`. E.g. if entry [V] of this list has value `[[N_0, P_0], [N_1, P_1]]` then the `V`-th entry in the output data array is the correlation of stand `N_0`, polarization `P_0` with stand `N_1`, polarization `P_1`

Output Headers

This is a bifrost sink block, and provides no data to an output ring.

Data Buffers

Input Data Buffer: A CPU-side bifrost ring buffer with 32+32 bit complex integer data in order `time x channel x visibility x complexity` This input buffer is read in gulps of `nchan x nvis` words, each 8 bytes in size.

Output Data Buffer: This block has no output data buffer.

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.

- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.
- **dest_port** (*int*) – Default destination port for UDP data. Can be overriden with the runtime control interface.
- **use_cor_fmt** (*Bool*) – If True, use the LWA `COR` packet output format. Otherwise use a custom format. See *Output Format*, below. Currently, only `use_cor_fmt=False` is supported.
- **nvis_per_packet** (*int*) – Number of visibilities to pack into a single UDP packet, if using the custom format (i.e., if `use_cor_fmt=False`). If using the COR format, this parameter has no effect.
- **npipeline** (*int*) – The number of pipelines in the system, as written to output COR packets. This may or may not be the same as the number of actual pipelines present. The one-index of this block's pipeline is computed from sequence header values as `((chan0 // nchan) % npipeline) + 1`.

Runtime Control and Monitoring

Field	Format	Units	Description
`dest_ip`	string		Destination IP for transmitted packets, in dotted-quad format. Eg. `"10.0.0.1"`. Use `"0.0.0.0"` to skip sending packets
`dest_port`	int		UDP port to which packets should be transmitted.

Output Data Format

If `use_cor_fmt=True`, this block outputs packets conforming to the LWA-SV "COR" spec (though with integer rather than floating point data). This format comprises a stream of UDP packets, each with a 32 byte header defined as follows:

```
struct cor {
  uint32_t  sync_word;
  uint8_t   id;
  uint24_t  frame_number;
  uint32_t  secs_count;
  int16_t   freq_count;
  int16_t   cor_gain;
  int64_t   time_tag;
  int32_t   cor_navg;
  int16_t   stand_i;
  int16_t   stand_j;
  int32_t   data[nchans, npols, npols, 2];
};
```

Packet fields are as follows:

Field	Format	Units	Description
sync_word	uint32		Mark 5C magic number, 0xDEC0DE5C
id	uint8		Mark 5C ID, used to identify COR packet, 0x02
frame_number	uint24		Mark 5C frame number. Unused.
secs_count	uint32		Mark 5C seconds since 1970-01-01 00:00:00 UTC. Unused.
freq_count	int16		zero-indexed frequency channel ID of the first channel in the packet.
cor_gain	int16		Right bitshift used for gain compensation. Unused.
time_tag	int64	ADC sample period	Central sampling time since 1970-01-01 00:00:00 UTC.
cor_navg	int16	TODO: sub-slots doesn't work	Integration time.
stand_i	int16		1-indexed stand number of the unconjugated stand.
stand_j	int16		1-indexed stand number of the conjugated stand.
data	int32*		The data payload. Data for the visibility of antennas at stand_i and stand_j, with stand_j conjugated. Data are a multidimensional array of 32-bit integers, with dimensions `[nchans, npols, npols, 2]`. The first axis is frequency channel. The second axis is the polarization of the antenna at stand_i. The second axis is the polarization of the antenna at stand_j.| The fourth axis is complexity, with index 0 the real part of the visibility, and index 1 the imaginary part.

If `use_cor_fmt=False`:

Each packet from the correlator contains data from multiple channels for multiple single-polarization baselines. There are two possible output formats depending on the value of `use_cor_fmt` with which this block is instantiated.

If `use_cor_fmt=False`, this block outputs a stream of UDP packets, with each comprising a 56 byte header followed by a payload of signed 32-bit integers. The packet definition is as follows:

```
struct corr_output_partial_packet {
  uint64_t   sync_time;
  uint64_t   spectra_id;
  double     bw_hz;
  double     sfreq_hz;
  uint32_t   acc_len;
  uint32_t   nvis;
  uint32_t   nchans;
  uint32_t   chan0;
  uint32_t   baselines[nvis, 2, 2];
  int32_t    data[nvis, nchans, 2];
};
```

Packet fields are as follows:

Field	Format	Units	Description
sync_time	uint64	UNIX seconds	The sync time to which spectra IDs are referenced.
spectra_id	int	•	The spectrum number for the first spectra which contributed to this packet's integration.
bw_hz	double (binary64)	Hz	The total bandwidth of data in this packet
sfreq_hz	double (binary64)	Hz	The center frequency of the first channel of data in this packet
acc_len	uint32	•	The number of spectra integrated in this packet
nvis	uint32	•	The number of single polarization visibilities present in this packet.
nchans	uint32	•	The number of frequency channels in this packet. For LWA-352 this is 184
chan0	uint32	•	The index of the first frequency channel in this packet
baselines	uint32*	•	An array containing the stand and polarization indices of the multiple visibilities present in this packet. This entry has dimensions [`nvis`, 2, 2]. The first index runs over the number of visibilities within this packet. The second index is 0 for the first (unconjugated) visibility input and 1 for the second (conjugated) antenna input. The third index is zero for stand number, and 1 for polarization number.
data	int32*	•	The data payload. Data for the visibility of antennas at stand0 and stand1, with stand1 conjugated. Data are a multidimensional array of 32-bit integers, with dimensions [`nvis`, `nchans`, 2]. The first axis runs over the multiple visibilities in this packet. Each index can be associated with a physical antenna using the `baselines` field. The second axis is frequency channel. The third axis is complexity, with index 0 the real part of the visibility, and index 1 the imaginary part.

5.2.9 `beamform`

class Beamform(*log*, *iring*, *oring*, *nchan=256*, *nbeam=1*, *ninput=704*, *ntime_gulp=2500*, *ntime_sum=None*, *guarantee=True*, *core=- 1*, *gpu=- 1*, *etcd_client=None*)

Functionality

This block reads data from a GPU-side bifrost ring buffer and feeds it to to a beamformer, generating either voltage or (integrated) power beams.

New Sequence Condition

This block starts a new sequence each time the upstream sequence changes.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
seq0	int		Spectra number for the first sample in the input sequence
nchan	int		Number of channels in the sequence
sfreq	double	Hz	Center frequency of first channel in the sequence
bw_hz	int	Hz	Bandwidth of the sequence
nstand	int		Number of stands (antennas) in the sequence
npol	int		Number of polarizations per stand in the sequence

Output Headers

This block passes headers from the upstream block with the following modifications:

Field	Format	Units	Description
nstand	int		Number of beams in the sequence
nbeam	int		Number of beams in the sequence
nbit	int		Number of bits per output sample. This block sets this value to 32
npol	int		Number of polarizations per beam. This block sets this value to 1.
complex	Bool		This block sets this entry to `True`, indicating that the data out of this block are complex, with real and imaginary parts each of width `nbit`

Data Buffers

Input Data Buffer: A GPU-side bifrost ring buffer of 4+4 bit complex data in order: `time x channel x input`.

Typically, the `input` axis is composed of `stand x polarization`, but this block does not assume this is the case.

Each gulp of the input buffer reads `ntime_gulp` samples, I.e `ntime_gulp x nchan x ninput` bytes.

Output Data Buffer: A GPU-side bifrost ring buffer of 32+32 bit complex floating-point data containing beamformed data. With `ntime_sum=None`, this is complex beamformer data with dimensionality `time x channel x beams x complexity`. This output buffer is written in blocks of `ntime_gulp` samples, I.e. `ntime_gulp x nchan x nbeam x 8` bytes.

With `ntime_sum != None`, this block will generate dynamic power spectra rather than voltages. This mode is experimental. See `bifrost/beamform.h` for details.

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **oring** (*bifrost.ring.Ring*) – bifrost output data ring. This should be on the GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.

- **gpu** (*int*) – GPU device which this block should target. A value of -1 indicates no binding
- **ntime_gulp** (*int*) – Number of time samples to copy with each gulp.
- **nchan** (*int*) – Number of frequency channels per time sample. This should match the sequence header `nchan` value, else an AssertionError is raised.
- **ninput** – Number of inputs per time sample. This should match the sequence headers `nstand x npol`, else an AssertionError is raised.
- **ntime_sum** (*int*) – Set to `None` to generate voltage beams. Set to an integer value >=0 to generate accumulated dynamic spectra. Values other than None are *experimental*.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.

Runtime Control and Monitoring

This block accepts the following command fields:

Field	Format	Units	Description
`delays`	2D list of float	ns	An `nbeam x ninput` element list of geometric delays, in nanoseconds.
`gains`	2D list of complex32)		A two dimensional list of calibration gains with shape `nchan x ninput`
`load_sample`	int	sample	**NOT YET IMPLEMENTED** Sample number on which the supplied delays should be loaded. If this field is absent, new delays will be loaded as soon as possible.

5.2.10 `beamform_sum_beams`

class BeamformSumBeams(*log*, *iring*, *oring*, *nchan=256*, *ntime_gulp=2500*, *ntime_sum=24*, *guarantee=True*, *core=- 1*, *gpu=- 1*, *etcd_client=None*)

Functionality

This block reads beamformed voltage data from a GPU-side ring buffer and generates integrated power spectra.

New Sequence Condition

This block starts a new sequence each time the upstream sequence changes.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
`seq0`	int		Spectra number for the first sample in the input sequence
`nchan`	int		Number of channels in the sequence
`nstand`	int		Number of stands (antennas) in the sequence
`nbeam`	int		Number of single polarization beams in the sequence

Output Headers

This block passes headers from the upstream block with the following modifications:

Field	Format	Units	Description
nbeam	int		Number of dual polatization beams in the sequence. Equal to half the input header nbeam
nbit	int		Number of bits per output sample. This block sets this value to 32
npol	int		Number of polarizations per beam. This block sets this value to 2.
complex	Bool		This block sets this entry to True, indicating that the data out of this block are complex, with real and imaginary parts each of width nbit
acc_len	int		Number of spectra integrated into each output sample by this block

Data Buffers

Input Data Buffer: A GPU-side bifrost ring buffer of 32+32 bit complex floating-point data containing beam-formed voltages. The input buffer has dimensionality (slowest varying to fastest varying) `time x channel x beams x complexity`. The number of beams should be even.

Each gulp of the input buffer reads `ntime_gulp` samples, I.e `ntime_gulp x nchan x nbeam x 8` bytes.

This block considers beam indices `0, 2, 4, ..., nbeam-2` to be `X` polarized, and beam indices `1, 3, 5, ..., nbeam-1` to be `Y` polarized. As such, `nbeam/2` output beams are generated by this block, with 4 polarization products each.

Output Data Buffer: A CPU- or GPU-side bifrost ring buffer of 32 bit, floating-point, integrated, beam powers. Data has dimensionality `time x channel x beams x beam-element`.

`channel` runs from 0 to `nchan`.

`beam` runs from 0 to the output `nbeam-1` (equivalent to the input `nbeam/2 - 1`).

`beam-element` runs from 0 to 3 with the following mapping:

- index 0: The accumulated power of a beam's `X` polarization
- index 1: The accumulated power of a beam's `Y` polarization
- index 2: The accumulated real part of a beam's `X x conj(Y)` cross-power
- index 3: The accumulated imaginary part of a beam's `X x conj(Y)` cross-power

This block sums over `ntime_sum` input time samples, thus writing `ntime_gulp / ntime_sum` output samples for every `ntime_gulp` samples read.

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **oring** (*bifrost.ring.Ring*) – bifrost output data ring. This should be on the GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **gpu** (*int*) – GPU device which this block should target. A value of -1 indicates no binding

- **ntime_gulp** (*int*) – Number of time samples to copy with each gulp.
- **nchan** (*int*) – Number of frequency channels per time sample. This should match the sequence header `nchan` value, else an AssertionError is raised.
- **ntime_sum** (*int*) – The number of time sample which this block should integrate. `ntime_gulp` should be an integer multiple of `ntime_sum`, else an AssertionError is raised.

Runtime Control and Monitoring

This block has no runtime control keys. It is completely configured at instantiation time.

5.2.11 beamform_output

class BeamformOutput(*log*, *iring*, *guarantee=True*, *core=- 1*, *etcd_client=None*, *dest_port=10000*, *ntime_gulp=480*)

Functionality

This block reads beamformed power data from a CPU-side buffer and transmits as a stream of UDP packets.

New Sequence Condition

This block is a bifrost sink, and generates no downstream sequences.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
`seq0`	int		Spectra number for the first sample in the input sequence
`nchan`	int		The number of frequency channels in the input data buffer
`chan0`	int		The index of the first frequency channel in the input data buffer
`nbeam`	int		The number of beams in the input data buffer
`nbit`	int		The number of bits (per real/imag part) of the input data samples. Must be 32
`complex`	Bool		True indicates that the input samples are complex, with real and imaginary parts both having `nbit` bits. Must be True.
`system_ncha n`	int		The total number of frequency channels in the multi-pipeline system. Must be a multiple of `nchan`.
`npol`	int		The number of polarizations in the input data buffer. Must be 1
`fs_hz`	int	Hz	ADC sample rate.

Output Headers

This is a bifrost sink block, and provides no data to an output ring.

Data Buffers

Input Data Buffer: A CPU-side bifrost ring buffer of 32 bit, floating-point, integrated, beam powers. Data has dimensionality `time x channel x beams x beam-element`.

`channel` runs from 0 to `nchan`.

`beam` runs from 0 to the output `nbeam-1` (equivalent to the input `nbeam/2 - 1`).

`beam-element` runs from 0 to 3 with the following mapping:

- index 0: The accumulated power of a beam's `X` polarization
- index 1: The accumulated power of a beam's `Y` polarization
- index 2: The accumulated real part of a beam's `X x conj(Y)` cross-power
- index 3: The accumulated imaginary part of a beam's `X * conj(Y)` cross-power

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **ntime_gulp** (*int*) – Number of time samples to copy with each gulp.
- **dest_port** (*int*) – Default destination port for UDP data. Can be overriden with the runtime control interface.

Runtime Control and Monitoring

This block reads the following control keys

Field	Format	Units	Description
`dest_ip`	list of string		Destination IP addresses for transmitted packets, in dotted-quad format. Eg. `"10.0.0.1"`. Use `"0.0.0.0"` to skip sending packets. Beam `i` is sent to `dest_ip[i % len(dest_ip)]`.
`dest_port`	list of int		UDP port to which packets should be transmitted. Beam `i` is sent to `dest_port[i] % len(dest_port)`

Output Data Format

Each packet output contains a single time sample of data from multiple channels and a single power beam. The output data format complies with bifrost's built-in "PBEAM" spec.

This format comprises a stream of UDP packets, each with a 18 byte header defined as follows:

```
struct ibeam {
    uint8_t  server; // 1-indexed
    uint8_t  beam;   // 1-indexed
    uint8_t  gbe;    // AKA "tuning"
    uint8_t  nchan;
    uint8_t  nbeam;
    uint8_t  nserver;
    uint16_t navg;   // Number of raw spectra averaged
    uint16_t chan0;  // First channel index in a packet
    uint64_t seq;    // 1-indexed: Really?
    float    data[nchan, nbeam, 4]; // Channel x Beam x Beam Element x 32-bit␣
↪float
};
```

Packet fields are as follows:

Field	Format	Units	Description
server	uint8		One-based "pipeline number". Pipeline 1 processes the first `nchan` channels, pipeline `p` processes the `p`-th `nchan` channels. Pipline ID counts through each channel block, and then multiple beams in the system. Eg, if the system has 512 channels, 256 channels per packet, and 3 beams, `server` runs from 1 to 6.
beam	uint8		One-based "beam number".
gbe	uint8		AKA "tuning". Set to 0.
nchan	uint8		Number of frequency channels in this packet
nbeam	uint8		Number of beams in this packet. Currently always 1.
nserver	uint8		The total number of pipelines in the system times the number of beams per pipeline
chan0	uint32		Zero-indexed ID of the first frequency channel in this packet.
seq	uint64	ADC sample period	Central sampling time since 1970-01-01 00:00:00 UTC.
data	float		Data payload. Beam powers, in order (slowest to fastest) `Channel x Beam x Beam Element`. Beam elements are `[XX, YY, real(XY), imag(XY)]`. Data are sent in native host endianness

5.2.12 `beamform_vlbi_output`

class BeamformVlbiOutput(*log*, *iring*, *guarantee=True*, *core=- 1*, *etcd_client=None*, *dest_port=10000*, *ntime_gulp=480*)

Functionality

Output a stream of UDP packets containing multiple beam voltages

New Sequence Condition

This block is a bifrost sink, and generates no downstream sequences.

Input Header Requirements

This block requires that the following header fields be provided by the upstream data source:

Field	Format	Units	Description
`seq0`	int		Spectra number for the first sample in the input sequence
`nchan`	int		The number of frequency channels in the input data buffer
`chan0`	int		The index of the first frequency channel in the input data buffer
`nbeam`	int		The number of beams in the input data buffer
`nbit`	int		The number of bits (per real/imag part) of the input data samples. Must be 32
`complex`	Bool		True indicates that the input samples are complex, with real and imaginary parts both having `nbit` bits. Must be True.
`system_ncha n`	int		The total number of frequency channels in the multi-pipeline system. Must be a multiple of `nchan`.
`npol`	int		The number of polarizations in the input data buffer. Must be 1

Output Headers

This is a bifrost sink block, and provides no data to an output ring.

Data Buffers

Input Data Buffer: A CPU- or GPU-side bifrost ring buffer of 32+32 bit complex floating-point data containing beamformed voltages. Data have dimensions (slowest to fastest): `time x channel x beams x complexity`. This buffer is read in blocks of `ntime_gulp` samples.

Output Data Buffer: This block has no output data buffer.

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **iring** (*bifrost.ring.Ring*) – bifrost input data ring. This should be on the GPU.
- **guarantee** (*Bool*) – If True, read data from the input ring in blocking "guaranteed" mode, applying backpressure if necessary to ensure no data are missed by this block.
- **core** (*int*) – CPU core to which this block should be bound. A value of -1 indicates no binding.
- **etcd_client** (*etcd3.client.Etcd3Client*) – Etcd client object used to facilitate control of this block. If `None`, do not use runtime control.
- **dest_port** (*int*) – Default destination port for UDP data. Can be overriden with the runtime control interface.
- **ntime_gulp** (*int*) – Number of time samples to read on each loop iteration.

Runtime Control and Monitoring

Field	Format	Units	Description
`dest_ip`	string		Destination IP for transmitted packets, in dotted-quad format. Eg. `"10.0.0.1"`. Use `"0.0.0.0"` to skip sending packets
`dest_port`	int		UDP port to which packets should be transmitted.

Output Data Format

Each packet output contains a single time sample of data from multiple channels and multiple voltage beams. The output data format complies with the LWA-SV "IBEAM" spec This format comprises a stream of UDP packets, each with a 32 byte header defined as follows:

```
struct ibeam {
    uint8_t  server;
    uint8_t  gbe;
    uint8_t  nchan;
    uint8_t  nbeam;
    uint8_t  nserver;
    uint16_t chan0;
    uint64_t seq;
    float    data[nchan, nbeam, 2]; // Channel x Beam x Complexity x 32-bit float
};
```

Packet fields are as follows:

Field	Format	Units	Description
server	uint8		One-based "pipeline number". Pipeline 1 processes the first `nchan` channels, pipeline `p` processes the `p`-th `nchan` channels.
gbe	uint8		AKA "tuning". Set to 0.
nchan	uint8		Number of frequency channels in this packet
nbeam	uint8		Number of beams in this packet
nserver	uint8		The total number of pipelines in the system.
chan0	uint32		Zero-indexed ID of the first frequency channel in this packet.
seq	uint64		Zero-indexed spectra number for the spectra in this packet. Specified relative to the system synchronization time.
data	float		Data payload. Beam voltages, in order (slowest to fastest) `Channel x Beam x Complexity`

5.2.13 `dummy_source`

The Dummy Source block is not used in the default LWA pipeline, but can replace the `Capture` block for testing purposes.

class DummySource(*log*, *oring*, *ntime_gulp=2500*, *core=- 1*, *nchan=192*, *nstand=352*, *npol=2*, *skip_write=False*, *target_throughput=22.0*, *testfile=None*)

Functionality

A dummy source block for throughput testing. Optionally writes test data to an output buffer.

New Sequence Condition

This block starts a single new sequence when main() is called.

Input Header Requirements

This block is a bifrost source, and thus has no input header requirements.

Output Headers

Field	Format	Units	Value	Description
sync_time	int	UNIX seconds	int(time.tim e())	Synchronization time (corresponding to spectrum sequence number 0).
seq0	int		0	Spectra number for the first sample in this sequence
chan0	int		0	Channel index of the first channel in this sequence
nchan	int		nchan	Number of channels in the sequence
system_nchan	int		nchan	The total number of channels in the system (i.e., the number of channels across all pipelines)
sfreq	double	Hz	0.0	Center frequency of first channel in the sequence
bw_hz	int	Hz	24000 * nchan	Bandwidth of the sequence
nstand	int		nstand	Number of stands (antennas) in the sequence
npol	int		npol	Number of polarizations per stand in the sequence
fs_hz	int	Hz	196608000	ADC Sample Rate
input_to_ant	list[int]		entry i is [i // npol, i % npol]	List of input to stand/pol mappings with dimensions [nstand x npol, 2]. E.g. if entry N of this list has value [S, P] then the N-th correlator input is stand S, polarization P.
ant_to_input	list[ints]		entry [s,p] is npol*s + p	List of stand/pol to correlator input number mappings with dimensions [nstand, npol]. E.g. if entry [S,P] of this list has value N then stand S, polarization P of the array is the N-th correlator input

Data Buffers

Input data buffer: None

Output data buffer: Complex 4-bit data with dimensions (slowest to fastest) `Time x Freq x Stand x Polarization x Complexity`

Instantiation

Parameters

- **log** (*logging.Logger*) – Logging object to which runtime messages should be emitted.
- **oring** (*bifrost.ring.Ring*) – bifrost output data ring
- **nstand** (*int*) – Number of stands in the array.
- **npol** (*int*) – Number of polarizations per antenna stand.
- **nchan** (*int*) – Number of frequency channels this block will output
- **core** (*int*) – CPU core to which this block should be bound. If `-1`, no binding is used.
- **ntime_gulp** (*int*) – The number of time samples to output on each processing loop iteration.
- **test_file** (*str*) – Path to a file containing test data, as a raw binary file containing 4+4 bit complex data in `time x channel x stand x polarization` order, with polarization changing fastest. This file should contain a multiple of `ntime_gulp` samples. When the end of the file is reached, it is repeated.
- **target_throughput** (*float*) – The target Gbits/s at which this block should output data. Throttling will be used to target this rate if necessary.

- **skip_write** (*Bool*) – If set to `True`, no data will be copied to the output buffer, blocks of memory will just be marked full as fast as possible. This can be useful to test the maximum throughput of the dowstream pipeline blocks. If set to `False` and no `testfile` is provided, the output data is a ramp, with each 4+4-bit data sample taking the value `stand % 8`

CHAPTER

SIX

CONTROL INTERFACE

Control and monitoring of the X-Engine pipeline is carried out through the passing of JSON-encoded messages through an `etcd`[1] key-value store. Each processing block in the LWA system has a unique identifier which defines a key to which runtime status is published and a key which should be monitored for command messages.

The unique key of a processing block is derived from the `blockname` of the module within the pipeline, the `hostname` of the server on which a pipeline is running, the pipeline id - `pipelineid` - of this pipeline, and the index of the block - `blockid` - which can disambiguate multiple blocks of the same type which might be present in a pipeline.

The key to which status information is published is:

`/mon/corr/x/<hostname>/pipeline/<pipelineid>/<blockname>/<blockid>/status`

The key to which users should write commands is:

`/cmd/corr/x/<hostname>/pipeline/<pid>/<blockname>/<blockid>/ctrl`

On receipt of a command, a processing block will respond on the key:

`/resp/corr/x/<hostname>/pipeline/<pid>/<blockname>/<blockid>/ctrl`

The status key contains a JSON-encoded dictionary of status information reported by the pipeline. The command key allows a user to send runtime configuration to a pipeline block, also in JSON dictionary format.

In addition to the pipeline processing blocks, a system control daemon listens for commands on the key:

`/mon/corr/x/<hostname>`

And responds on key:

`/mon/corr/x/<hostname>`

Some fields in these status and command messages are common amongst blocks, while others are block-specific.

Users can either interact directly with the etcd keys, and perform encoding and decoding as appropriate, or can use the Pythonic control interface provided in the `lwa352-pipeline-control` library.

[1] See etcd.io

6.1 Command Format

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

Field	Type	Description
id	string	A unique string associated with this command, used to identify the command's response
cmd	string	Command name. To update control keys of pipeline processing blocks, this should be "update". Otherwise, it should reflect the command supported by the correlator control daemon.
val	dictionary	**A dictionary containing keys:** • kwargs (dictionary): Dictionary of pipeline-specific keys to set. These should match the control keys definied by individual blocks below.

Allowed values for **``block``** are any of the block names in the processing pipeline. I.e.:

- Capture
- Copy
- TriggeredDump
- Corr
- CorrSubsel
- CorrOutputPart
- CorrAcc
- CorrOutputFull
- Beamform
- BeamformVlbiOutput
- BeamformSumBeams
- BeamformOutput

Additionally, the block name `xctrl` may be used to issue high-level system commands as defined in the `XengineController` class.

Allowed values for **``cmd``** are

- For `block="xctrl`, and non-private commands of the `XengineController` class
- For all other blocks, `cmd` should be the string `"update"`.

The **``kwargs``** field of the `val` dictionary should contain:

- For `block="xctrl`, arguments and values for the `XengineController` method being called.
- For all other blocks, `kwargs` should be a dictionary of control keys, and their designed update values, for the relevant processing block.

For example, to set the Correlator short term accumulation length to 4800, a command should be issues with the following values:

Field	Value
cmd	"update"
val	{"acc_len": 4800}:

An example of a valid command JSON string, issued with the above parameters and with `id="1"` is:

`'{"cmd": "update", "val": {"block": "delay", "kwargs": {"acc_len": 4800}, "id": "1"}'`

Consult the Pipeline block descriptions for details of the control keys associated with particular processing blocks.

6.2 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
id	string	A string matching the `id` field of the command string to which this is a response
val	dictionary	**A dictionary containing keys:** • 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 (string): The response of the command method, as determined by the command.

Processing blocks have the following response strings:

- "0": Command OK
- "-1": Command not recognized
- "-2": Command kwargs had wrong data type
- "-3": Command invalid

Field	Value
id	"1"
val	{timestamp: 1618060712.8, status: "normal", response: "0"}

or, in JSON-encoded form:

`'{"id": "1", "val": {"timestamp": 1618060712.8, "status": "normal", "response": "0"}}'`

The correlator controller (i.e., `block="xctrl"`) responds to commands with the following error strings in the `response` field:

"JSON decode error"	Command string could not be JSON-decoded.
"Sequence ID not string"	Sequence ID was not provided in the command string or decoded to a non-string value.
"Bad command format"	Received command did not comply with formatting specifications. E.g. was missing a required field such as `block` or `cmd`.
"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.

6.3 Block-Specific Interfaces

6.3.1 Correlator Pipeline Control

The correlator `XengineController` control class has the following methods which may be called over etcd using the block name `xctrl`:

class XengineController(*logger=None*)

set_log_level(*level*)

Set the logging level.

Parameters level (*string*) – Logging level. Should be "debug", "info", or "warning"

6.4 Control Library

A control library is provided which provides a Python interface to the underlying etcd API.

6.4.1 Correlator Pipeline Control

class Lwa352CorrelatorControl(*hosts, npipeline_per_host=4, etcdhost='etcdv3service', log=<Logger lwa352_pipeline_control.lwa352_pipeline_control (DEBUG)>*)

Description

A class to encapsulate control of a multi-pipeline LWA correlator system. Internally, this instantiates multiple `Lwa352PipelineControl` instances.

Instantiation

Parameters

- **hosts** (*list of str*) – A list of hostnames of servers running DSP pipelines. Eg. `['lxdlwagpu0', 'lxdlwagpu1']`
- **npipeline_per_host** (*int*) – The number of pipeline instances running on each server.
- **etcdhost** (*string*) – The hostname of the system running the correlator's etcd server

- **log** (*logging.Logger*) – The logger to which this class should emit log messages. The default behaviour is to log to stdout

Variables

- **hosts** – User-passed `host`
- **npipeline_per_host** – User-passed `npipeline_per_host`
- **npipeline** – Total number of pipelines in the system
- **pipelines** – List of `Lwa352PipelineControl` control instances for each server and pipeline.
- **log** – User-passed `log`
- **etcdhost** – User-passed `etcdhost`

ARM_DELAY = 5
Delay, in seconds, between arm issue and start time

WAIT_DELAY = 5
Wait delay, in seconds, after arming blocks

configure_corr(*dest_ip='10.41.0.19'*, *dest_port=10001*, *max_mbps=20000*)
Configure correlator "slow" output, and arm upstream correlation / accumulation logic to being outputting data.

Parameters

- **dest_ip** (*str*) – Hostname or IP address to which slow correlator packets should be sent. If list of ips is provided, pipeline `n` will use destination ip `dest_ip[n % len(dest_ip)]`
- **dest_port** (*int or list of int*) – UDP port to which slow correlator packets should be sent. If list of ports is provided, pipeline `n` will use destination port `dest_port[n % len(dest_port)]`
- **max_mbps** (*int*) – Total correlator output data rate in Mbits/s. This rate is shared over multiple pipelines in the system.

start_pipelines()
Start all pipelines, using the default configuration.

stop_pipelines()
Stop all pipelines.

6.4.2 Beamformer Control

class BeamformControl(*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

update_calibration_gains(*beam_id*, *input_id*, *gains*)
Update calibration gains for a single beam and input.

Parameters

- **beam_id** (*int*) – Zero-indexed Beam ID for which coefficients are begin updated.
- **input_id** (*int*) – Zero-indexed Input ID for which coefficients are begin updated.
- **gains** (*numpy.array*) – Complex-valued gains to load. Should be a numpy array with a complex data type and `nchan` entries, where entry `i` corresponds to the `i` th channel being processed by this pipeline.

update_delays (*beam_id*, *delays*)
Update geometric delays for a single beam.

Parameters

- **beam_id** (*int*) – Zero-indexed Beam ID for which coefficients are begin updated.
- **delays** (*numpy.array*) – Real-valued delays to load, specified in nanoseconds. Should be a numpy array with `nbeam` entries, where entry `i` corresponds to the delay to apply to the `i` th beamformer input.

6.4.3 Beamformer Output Control

class BeamformOutputControl (*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

6.4.4 Beamformer VLBI Output Control

class BeamformVlbiOutputControl (*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

6.4.5 Correlator Control

class CorrControl (*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

get_next_allowed_start (*delay_s*)
Get the time index of an allowed accumulation start at some point in the future.

Parameters **delay_s** (*float*) – Number of seconds in the future when we wish to start.

Returns Nearest allowed start time index to `delay_s` seconds in the future.

Rtype int

set_acc_length (*acc_len*)
Set the block's accumulation length.

Parameters **acc_len** (*int*) – Commanded accumulation length, in units of spectra. This should be a multiple of the blocks underlying accumulation granularity.

set_start_time (*start_time*)
Set the block's accumulation start time.

Parameters **start_time** (*int*) – Commanded start time, in units of spectra since last synchronization (counter reset) event. Use -1 to command the block to start immediately, on the next valid integration boundary.

triggered_start (*delay_s*)
Trigger a start in the future.

Parameters **delay_s** (*float*) – Number of seconds in the future to start.

6.4.6 Correlator Accumulator Control

class CorrAccControl(*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

get_next_allowed_start(*delay_s*)

Get the time index of an allowed accumulation start at some point in the future.

Parameters **delay_s** (`float`) – Number of seconds in the future when we wish to start.

Returns Nearest allowed start time index to `delay_s` seconds in the future.

Rtype int

6.4.7 Correlator Baseline Subselect Control

class CorrSubselControl(*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

6.4.8 Correlator Partial Visibility Output

class CorrOutputPartControl(*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

6.4.9 Correlator Full Visibility Output

The full correlator visibility output packet stream is controlled by the `CorrOutputFull` block, and has the following interface.

Control

class CorrOutputFullControl(*log*, *corr_interface*, *host*, *pipeline_id=0*, *name=None*, *instance_id=0*)

Control interface for the `CorrOutputFull` processing block. Control keys:

Field	Format	Units	Description
`dest_ip`	string		Destination IP for transmitted packets, in dotted-quad format. Eg. `"10.0.0.1"`. Use `"0.0.0.0"` to skip sending packets. See `set_destination()`.
`dest_file`	string		If not "", overrides `dest_ip` and causes the output data to be written to the supplied file
`dest_port`	int		UDP port to which packets should be transmitted. See `set_destination()`.
`max_mbps`	int	Mbits/s	The maximum output data rate to allow before throttling. Set to `-1` to send as fast as possible. See `set_max_mbps()`.

set_destination(*dest_ip='0.0.0.0'*, *dest_port=10000*, *dest_file=''*)

Set the destination IP and UDP port for correlator packets, using the `dest_ip` and `dest_port` keys.

Parameters

- **dest_ip** (`str`) – Desired destination IP address, in dotted quad notation – eg. "10.10.0.1"
- **dest_port** (`int`) – Desired destination UDP port

- **dest_file** (*str*) – If provided, write data output packets to this file, rather than the destination IP. Useful for testing.

set_max_mbps (*max_mbps*)
Use the `max_mbps` key to throttle the correlator output to at most `max_mbps` megabits per second. Throttle is approximate only.

Parameters **max_mbps** (*int*) – Output data rate cap, in megabits per second

Status

`CorrOutputFullControl.`**get_status**()
Get correlator full visibility output stats:

Field	Type	Unit	Description
`curr_sample`	int		The index of the last sample to be processed
`dest_ip`	string		Current destination IP address, in dotted- quad notation.
`dest_port`	int		Current destination UDP port
`last_cmd_time`	float	UNIX time	The last time a command was received
`last_update_time`	float	UNIX time	The last time settings from a command were loaded
`max_mbps`	int	Mbits/s	The current throttle setpoint for output data
`new_dest_ip`	string		The commanded destination IP address, in dotted- quad notation. This IP will be loaded on the next visibility matrix to be transmitted if `update_pending` is True.
`new_dest_port`	int		The commanded destination UDP port, to be loaded on the next visibility matrix to be transmitted if `update_pending` is True
`new_max_mbps`	int	Mbits/s	The commanded throttle setpoint for output data, to be loaded on the next visibility matrix to be transmitted if `update_pending` is True
`output_gbps`	float	Gbits/s	Measured output data rate for the last visibility matrix
`update_pending`	bool		Flag indicating that the IP/port/throttle settings have changed and should be updated on the next visibility matrix

Returns Block status dictionary

Return type dict

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

T

U

W

X