LWA Correlator

Release 1.0.0

Jack Hickish

Dec 15, 2020

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]).

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] [-f] [-c CONFIGFILE] [-l LOGFILE] [-v]
                          [--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]
                          [--etcdhost ETCDHOST] [--ip IP]
                          [--bufgbytes BUFGBYTES]
                          [--target_throughput TARGET_THROUGHPUT]

LWA-SV ADP DRX Service

optional arguments:
  -h, --help            show this help message and exit
  -f, --fork            Fork and run in the background (default: False)
  -c CONFIGFILE, --configfile CONFIGFILE
                        Specify config file (default: adp_config.json)
  -l LOGFILE, --logfile LOGFILE
                        Specify log file (default: None)
  -v, --verbose         Increase verbosity (default: 0)
  --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)
  --etcdhost ETCDHOST   Host serving etcd functionality (default: etcdhost)
  --ip IP               IP address to which to bind (default: 100.100.100.101)
  --bufgbytes BUFGBYTES
                        Number of GBytes to buffer for transient buffering
                        (default: 4)
  --target_throughput TARGET_THROUGHPUT
                        Target throughput when using --fakesource (default:
                        1000.0)
```

CHAPTER

FIVE

SOFTWARE

The LWA-352 pipeline comprises ?? 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. `copy`: Transfer blocks of data from CPU to GPU, for high-performance computation.
3. `triggered_dump`: Buffer large quantities of time-domain data for triggered dump to disk.
4. `corr`: Correlate data using the `xGPU` library.
5. `corr_output_full`: Output full, accumulated visibility matrices.
6. `corrsubsel`: Down-select a sub-set of the complete visibility matrices.
7. `corr_output_part`: Output subselected visibilities as UDP/IP streams.
8. `beamform`: Form multiple voltage beams.
9. `beamform_vlbi_output`: Package and transmit multiple voltage beams for VLBI purposes.
10. `beamform\_sum_beams`: Form integrated power-spectra for multiple beams.
11. `beamform\_output`: Output accumulated power beams.

5.1 High-Level Parameters

- `GSIZE`: "Gulp size" – the number of samples processed per batch by a processing block.

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 `lwa352_pipeline.blocks.capture_block.`**`Capture`**(*log*, *fs_hz=196000000*, *chan_bw_hz=23925.78125*, *input_to_ant=None*, *nstand=352*, *npol=2*, **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
`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.
- **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 `lwa352_pipeline.blocks.copy_block.`**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 `lwa352_pipeline.blocks.triggered_dump_block.`**`TriggeredDump`**(*log, iring, ntime_gulp=2500, ntime_per_file=1000000, guarantee=True, core=-1, nbyte_per_time=135168, etcd_client=None, dump_path='/fastdata'*)

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 `lwa352_pipeline.blocks.corr_block.`**`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
`seq 0`	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 `lwa352_pipeline.blocks.corr_acc_block.`**`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 `lwa352_pipeline.blocks.corr_output_full_block.`**`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*)

Perform GPU side accumulation and then transfer to CPU

5.2.7 `corrsubsel`

class `lwa352_pipeline.blocks.corr_subsel_block.`**`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*)

Grab arbitrary entries from a GPU buffer and copy them to the CPU

5.2.8 corr_output_part

5.2.9 beamform

5.2.10 beamform_vlbi_output

5.2.11 beamform_sum_beams

5.2.12 beamform_output

CHAPTER

SIX

OUTPUT DATA FORMATS

This section defines the output packet formats for each of the pipeline output data products. Unless otherwise specified, all data products are transmitted in network- (i.e. big-) endian format.

Packet sizing is partially determined by the pipeline configuration. Specifically:

- `NCHAN` – The number of channels processed per pipeline.
- `NSTAND` – The number of antenna stands in the array.
- `NPOL` – The number of polarizations per antenna stand.

For the LWA-352 system:

- `NCHAN` = 184
- `NSTAND` = 352
- `NPOL` = 2

6.1 Full Correlation Packets

Data from the full, slow correlator are transmitted as a series of UDP packets, with each packet carrying data for one dual-polarization baseline, for multiple channels. Each packet has a 56 byte header followed by a payload of signed 32-bit integers.

```
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.

6.2 Partial Correlation Packets

Data from the fast dump correlator are transmitted as a series of UDP packets, with each packet carrying data for multiple frequency channels of multiple, single-polarization visibilities.

Each packet has a variable length header followed by a payload of signed 32-bit integers.

```
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.

6.3 VLBI Beam

6.4 Integrated Beams

CHAPTER

SEVEN

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, and the pipeline id - `pid` - of this pipeline.

In general, keys to which status information is published have the prefix:

`/mon/corr/xeng/<hostname>/pipeline/<pid>/<blockname>`.

Keys to which users should write commands have the prefix

`/cmd/corr/xeng/<hostname>/pipeline/<pid>/<blockname>`.

The format of these status and command messages, and their allowed values are given in the remainder of this section on a per-block basis.

7.1 Capture Thread (blockname: `capture`)

7.1.1 Commands

The `capture` block accepts no runtime commands. When a pipeline is executed, the capture module will automatically begin filling processing buffers. Buffer boundaries occur every `GSIZE` samples.

7.1.2 Monitoring

The `capture` block writes monitoring data to the key `/mon/corr/xeng/<hostname>/pipeline/<pid>/capture`. Data are written as a JSON-encoded dictionary with the following entries:

Field	Format	Units	Description
thoughput	float	Gbits/s	Block throughput
n_dropped	int	packets	Number of packets dropped since pipeline start
n_received	int	packets	Number of packets received since pipeline start
frac_dropped	float		Fraction of packets dropped since pipeline start
n_late	int	packets	Number of late packets since pipeline start
n_f_missing	int	boards	TODO
n_part_dropped	int	packets	TODO
time	float	UNIX time	The time this key was updated.

[1] See etcd.io

7.2 Copy Thread (blockname: gpucopy)

The gpucopy block accepts no runtime commands and outputs no run-time statistics.

7.3 Correlation Thread (blockname: corr)

The corr block takes blocks of GSIZE 4-bit time samples from the gpucopy thread and generates visibility matrices using an xGPU computation kernel. Integration takes place over the GSIZE input samples.

7.3.1 Commands

The corrblock has a run-time configurable accumulation length and start time. These can be set by writing a JSON-encoded dictionary to the key /cmd/corr/xeng/<hostname>/pipeline/<pid>/corr, which should have the following fields:

c c c X Field & Format & Units & Description

acc_len & int & samples & Number of samples to integrate. Must be a multiple of GSIZE. acc_len = 0 can be used to force the corrmodule to stop processing.

start_time & int & samples & Sample index on which to begin integrating. Must be a multiple of GSIZE.

Sample indices are relative to the F-Engine sync time – i.e., sample index 0 is the first sample after an F-Engine sync event. Sample indices can only be converted to real time with the knowledge of the F-Engine sync time and F-Engine ADC clock rate.

It should be noted that modifying the run-time configuration of the corrmodule will impact both the fast- and slow-visibility processing streams. Both streams will re-synchronize onto new correlator integration boundaries.

7.3.2 Monitoring

The corrblock writes status data as a JSON-encoded dictionary to the key:

/mon/corr/xeng/<hostname>/pipeline/<pid>/corr

The status dictionary has the following fields:

c c c X Field & Format & Units & Description

thoughput & float & Gbits/s & Block throughput

acc_len & int & samples & Number of samples currently set to integrate

start_sample & int & samples & Current start time.

curr_sample & int & samples & The last sample to be processed.

update_pending & bool & - & True if new integration parameters are waiting to be loaded.

last_update_time & float & seconds & The time since UNIX epoch that the imtegration parameters were last updated.

new_acc_len & int & samples & The commanded integration length

new_start_sample & int & samples & The commanded start sample

last_cmd_time & float & seconds & The time since UNIX epoch that the last command was received

7.4 Visibility Sub-Select Thread (blockname: corrsubsel)

7.4.1 Commands

The `corrsubsel`block outputs a run-time configurable set of baselines. These can be set by writing a JSON-encoded dictionary to the key `/cmd/corr/xeng/<hostname>/pipeline/<pid>/corrsubsel`, which should have the following fields:

c c c X Field & Format & Units & Description

subsel & list(int) & - & A list of baselines for subselection. This field should be provided as a multidimensional list with dimensions [`N_VIS`, 2, 2]. The first axis runs over the 4656 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 4656 entries.

7.4.2 Monitoring

The `corr`block writes status data as a JSON-encoded dictionary to the key: `/mon/corr/xeng/<hostname>/pipeline/<pid>/corrsubsel`.

The status dictionary has the following fields:

Field	Format	Units	Description
thoughput	float	Gbits/s	Block throughput
subsel	list(int)	samples	Current set of visibility indices being selected
update_pending	bool		True if new selection parameters are waiting to be loaded.
last_update_time	float	seconds	The time since UNIX epoch that the selection parameters were last updated.
new_subsel	list(int)	samples	The commanded visibility selection indices.
last_cmd_time	float	seconds	The time since UNIX epoch that the last command was received

7.5 Visibility Integrator (blockname: `corracc`)

7.5.1 Commands

The `corracc` block further integrates the output of the `corr` block. Integration parameters can be set by writing a JSON-encoded dictionary to the key:

`/cmd/corr/xeng/<hostname>/pipeline/<pid>/corracc`

This should have the following fields:

Field	Format	Units	Description
acc_len	int		Number of samples to integrate. acc_len = 0 can be used to force the `corracc` module to stop processing.
start_time	int	samples	Sample index on which to begin integrating.

Note that the acc_len configuration must be compabible with – i.e., must be a multiple of – the accumulation length set in the `corr` block. Furtherm the start_time must be compatible with the integration boundaries associated with the `corr` block's integration settings.

Run-time checks will flag bad configurations as errors, but no check is made on issuing a command to ensure it is valid. After booting the pipeline, a safe order of configuration is:

1. Boot pipeline.
2. Configure `corracc` block
3. Configure `corr` block

For changes of configuration, the safe order of updates is:

7.5.2 Monitoring

The `corracc` block writes status data as a JSON-encoded dictionary to the key:

`/mon/corr/xeng/<hostname>/pipeline/<pid>/corracc`.

The status dictionary has the following fields:

Field	Format	Units	Description
acc_len	int	samples	Number of samples currently set to integrate
start_sample	int	samples	Current start time.
curr_sample	int	samples	The last sample to be processed.
update_pending	bool		True if new integration parameters are waiting to be loaded.
last_update_time	float	seconds	The time since UNIX epoch that the imtegration parameters were last updated.
new_acc_len	int	samples	The commanded integration length
new_start_sample	int	samples	The commanded start sample
last_cmd_time	float	seconds	The time since UNIX epoch that the last command was received

7.6 Beamformer (blockname: `beamform`)

The `beamform` block forms `2 x NBEAM` independent, single polarization voltage beams. Beam pointings are specified by relative antenna delays and a set of universal, frequency-dependent calibration coefficients, which are shared among all beams. Note that this interface precludes direction-dependent calibrations.

7.6.1 Commands

Commands are sent to be `beamform` module by writing a JSON-encoded command to the key:

`/cmd/corr/xeng/<hostname>/pipeline/<pid>/beamform`

This command should have the following fields

Field	Format	Units	Description
delays[x]	list(float)	ns	An `NINPUT` element list of geometric delays, in nanoseconds. [x] is a beam index, and should be between 0 and `NBEAM - 1`
gains	list(complex32)		A two dimensional list of calibration gains with shape `[NCHAN, NINPUT]`
load_sample	int	sample	Sample number on which the supplied delays should be loaded. If this field is absent, new delays will be loaded as soon as possible

The `beamform` block calculates voltage beams only and has no concept of polarization. Instead, the `beamform` block generates `2 x NBEAM` beams and computes the auto- and cross-power spectra between beams in order to generate auto- and cross-pol products. Beams are paired such that the cross-power of beams `2n` and `2n+1` are computed – it is the user's responsibility to ensure that these beams have the same pointing and are formed from complementary antenna polarizations.

7.6.2 Monitoring

The `beamform` block writes status data as a JSON-encoded dictionary to the key:

`/mon/corr/xeng/<hostname>/pipeline/<pid>/beamform`.

The status dictionary has the following fields:

Field	Format	Units	Description
thoughput	float	Gbits/s	Block throughput
delays[x]	list(float)	ns	An `NINPUT` element list containing the delays currently loaded for beam `x`
gains	list(complex32)		A two dimensional list of currently loaded calibration gains. The dimensions of this list should be `NCHAN x NINPUT`
new_delays[x]	list(float)	ns	An `NINPUT` element list containing the next set of delays to be loaded for beam `x`
new_gains	list(complex32)		A two-dimensional list of calibration gains with shape `[NCHAN, NINPUT]`
curr_sample	int	samples	The last sample to be processed.
update_pending	bool	•	True if new integration parameters are waiting to be loaded.
last_update_time	float	seconds	The time since UNIX epoch that the imtegration parameters were last updated.
new_acc_len	int	samples	The commanded integration length
new_start_sample	int	samples	The commanded start sample
last_cmd_time	float	seconds	The time since UNIX epoch that the last command was received

CHAPTER EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

C

T