ALMA Correlator Online Processing Pipeline

Rodrigo Amestica

June 18, 2020

Contents

1 ALMA correlator online processing pipeline 2
1.1 Reference Documents 2
1.2 Introduction 2
1.3 FXF Correlator Output 4
1.4 Spectral Specification 5
1.5 Back-end Computers Structure 6
1.6 Spectral Data Processor Storage 7
1.7 Time Domain Operations 8
1.7.1 Lag Normalization 8
1.7.2 Quantization Correction 9
1.7.3 Window Function 10
1.7.4 Side-band Separation 11
1.8 Frequency Domain Operations 12
1.8.1 Correlation Function to Power Spectrum 12
1.8.2 Bandshape Correction and Stitching of Poly-phase Filter Bank Slices 13
1.8.3 Power Spectrum Normalization 14

1 ALMA correlator online processing pipeline

1.1 Reference Documents

Doc. #	Title
RD01	Algorithms and Formulas for Hybrid Correlator Data Correction
RD02	ICD Between 64-Antenna Correlator And Correlator Computing System
RD03	Binary Data Format
RD04	The ALMA correlator (Escoffier et al.)
RD05	Specifications and Clarifications of ALMA Correlator Details
RD06	Programming Manual for Tunable Filter Bank
RD07	Van Vleck Correction for the GBT Correlator
RD08	ALMA Cycle 7 Technical Handbook
RD09	Interferometry and Synthesis in Radio Astronomy
RD10	Walsh Function Choices for 64 Antennas

1.2 Introduction

The ALMA Correlator Data Processor (CDP) subsystem ingests time-domain correlations and processes them into spectral-domain autocorrelations and visibilities. It packetize and delivers the results in a binary format at a cadence close to real-time (~10 seconds or less after an integration has completed).

The actual algorithms applied to the correlator data by the CDP can be categorized in two major groups:

1. time-domain operations,
2. spectral domain operations, and

They are specific to the nature of the underlying correlator hardware. The sequential set of software operations applied to the data, represents the processing pipeline described in the present paper.

The software pipeline implementation is based on technical aspects developed in ALMA memo #583 [RD01]. The memo includes a graphical description of what the correlator hardware does and, therefore, what the software pipeline is expected to apply to the data. See Fig. #1 below.

Once a correlation function has been ingested by a CDP node, it then waits for ancillary information to arrive through the network before further processing can proceed. Basically, a node must wait for:

- autocorrelation power levels for spectral normalization, and
- geometric delay values for fine delay corrections (phase rotation.) Constant instrumental delays for each antenna, are accessed from a central database each time a subarray is created.

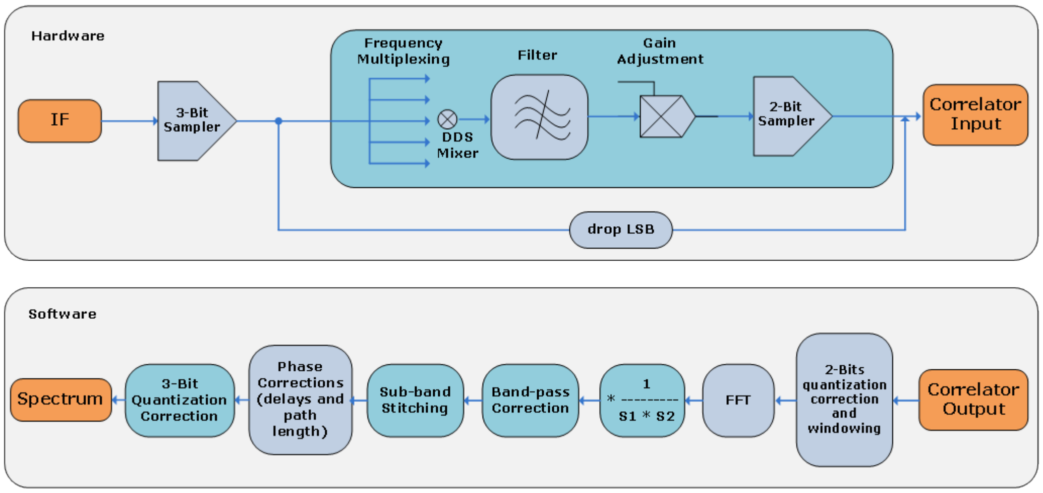

Figure 1: CDP pipeline stages in relation to operations that happen at hardware level, throughout the correlator signal path.

Autocorrelation information is actually needed due to the split introduced by the correlator cross-bar on each quadrant. That is, there are two nodes in a quadrant that receive only crosscorrelations and, therefore, need to wait for autocorrelations from the other two nodes before the former could normalize visibilities products. See Fig. #17.1 in [RD02].

The pipeline repeats a sequence of operations for each dump received from the correlator hardware. Until the integration time has elapsed and then the spectral normalization takes place on the accumulated data. Note also that each integration duration might contain a number of lower resolution channel average results. Full resolution and channel average items are time averaged and normalized at the time after the whole integration has completed. That is, shorter duration channel average results are not delivered faster than every integration interval.

After a node completes the processing of an integration, then, the results are streamed out through the network to the CDP master node. Each queued memory buffer contains the integration itself plus all channel averages, and some metadata that makes possible to fully qualify the contained data. The CDP master makes use of the metadata to collect and assemble results into a larger buffer that follows a given Binary Data Format (BDF) [RD03]. Whenever a BDF sub-header (integration or channel average) is ready, then it is streamed out through the network to subsystems downstream (archive and telescope-calibration).

The content of this note is divided as it follows. Chapter one contains the introduction presented above. Chapter 2 describes what's the actual data produced by the ALMA correlator and consumed by the CDP. Chapter 3 presents the XML container used to encapsulate the correlator configuration mode and involved spectral windows and other parameters needed by the CDP pipeline. Chapter 4 presents the physical deployment of CDP computers and their connection to the correlator. Chapter 5 deals with those operations at correlator output directly, that is, time-domain correlation functions (vectors of leads and lags.) Chapter 6 concentrates on the FFT itself and other spectral-domain operations.

Through the note's content there are references (URL) to source code files that implement the pertinent functionally. Those URLs point to files in ALMA's Bitbucket GIT repository.

1.3 FXF Correlator Output

The ALMA correlator corresponds to an FXF correlator design [RD04]. That is, an FPGA coarse frequency channelizer (F) is followed by an ASIC X-engine and FPGA crossbar (X), which finally discharges results to a CPU-based computer cluster running a custom processing pipeline (FFT and others) under the Linux operating system (F).

Time domain auto-correlations and cross-correlations values (colloquially called lags) are ingested by the CDP pipeline in real-time. The pipeline's task is to transform those time-domain lags into their spectral-domain representation and to normalize the resulting spectral vectors to a project's specific convention [RD05]. In other words, the last 'F' in FXF represents the computer-based subsystem described in the present note.

At the same time, it is relevant to keep in mind that the first 'F', is implemented by a custom built tunable-filter-bank (TFB) located in the station electronics section of the correlator. The specific TFB design and implementation plays an important role to the tasks executed by the CDP pipeline.

Each filter bank in a TFB board divides the 2 GHz baseband input signal into 32 narrower frequency slices of 62.5 MHz bandwidth each (2-bit samples at 125 Mega-samples-per-second). These narrower sub-band signals are then processed by the correlator, applying different combinations of available hardware resources to create different correlator modes. For details make the connection between modes in [RD04] and TFB operational details in [RD06]. The frequency filtering/multiplexing mechanism applied to produced these correlator modes make them be called Frequency Division Modes (FDM).

On the other hand, at its lowest spectral resolution, the correlator skips the filtering part in the TFB and it directly drops the least-significant-bit of each voltage sample (correlator input is always based on 2-bit samples). It then multiplexes consecutive blocks of samples through 32 independent correlator planes, and finally it adds the 32 outputs together creating a low resolution but high sensitivity correlation function. Due to the time-multiplexing mechanism involved in this case, these type of correlator modes are called Time Division Modes (TDM).

The diagram in Fig. #2 presents the major hardware components taking part during the generation of a correlation function. Prompt blocks both produce 256 individual multiply-and-accumulate outputs. Only one of them generates a value that has no delay between samples from both antennas. A value that for the same antenna ($i = j$) and only in TDM mode, represents the total digital power at that correlator antenna input. To make easier to refer to this specific lag across subsequent stages in the pipeline (quantization correction and spectral normalization in particular), this particular lag value is named lag-zero.

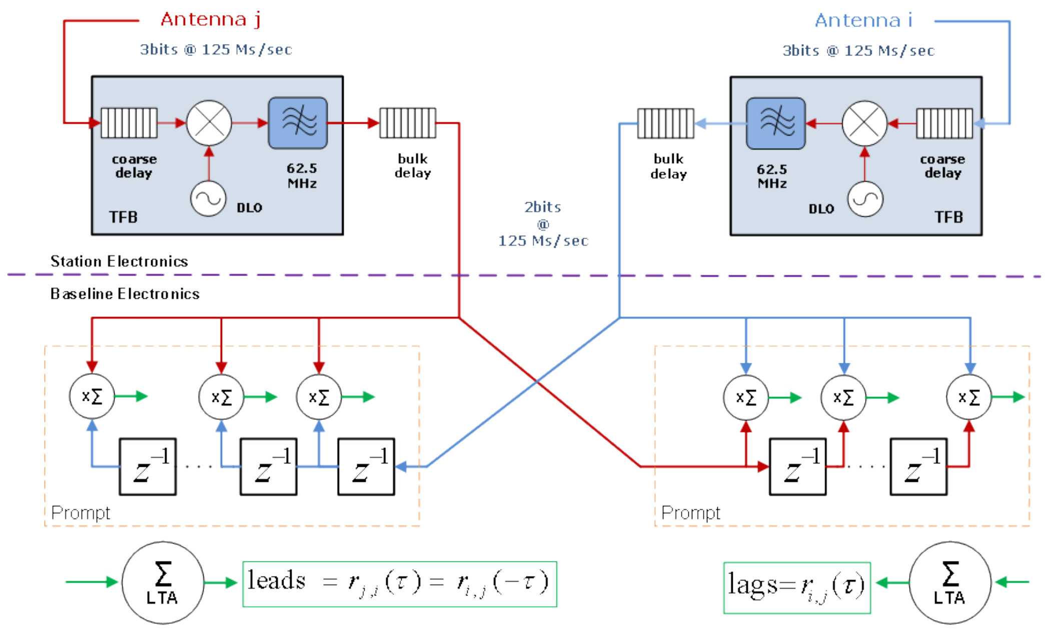

Figure 2: Major hardware stages through the station and correlator electronics.

As explained above, in Frequency Division Mode (FDM) each TFB sub-band observes just one narrower slice (62.5 MHz) of the 2 GHz baseband input digital signal and, therefore, the CDP misses access to a measurement of the total input digital power. In this case, the CDP defaults to using the lag-zero value of the first sub-band for normalization purposes. A proper description of the consequences of this issue can be found in [RD01] Chapter 7.

1.4 Spectral Specification

At different times during an observation and across multiple concurrent sub-arrays, correlator modes are commanded by a higher level software that dispatches sets of parameters to the correlator subsystem. The parameters are encapsulated in an XML document (a string on the wire) that conforms to the SpectralSpec type in the SchedBlock XML schema. A diagram showing attributes and elements in the spectral specification can be inspected in this UK ATC web page. Note that the link is specific to ALMA Cycle-7. Given that the APDM is an abstraction that changes as soon as new capabilities are added or other optimized, then the diagram might change and the pointer updated to the latest version.

From a SchedBlock string the correlator software extracts the SpectralSpec element, validates it for parameters correctness and then it converts it to a C++ structure that the CDP software can actually consume in real-time.

A given correlator configuration is meant to execute during a given lapse of time, produce integration results at a given rate and consume a given slice of baseband bandwidth. These major timing and spectral configuration parameters are present in the SpectralSpec XML element like shown below:

```
SpectralSpecT -->
```

```
BLCorrelatorConfiguration -->
   integrationDuration    [time units]
   channelAverageDuration [time units]
   dumpDuration           [time units]
   BLBaseBandConfig       [1..4]
```

Where the four basebands vector captures the specific correlator configuration for each quadrant and, therefore, the specific operational configuration handled by each CDP node attached to that quadrant. Implicitly stated here, the system is able to observe using different correlator modes across all four basebands (quadrants).

1.5 Back-end Computers Structure

The correlator ingests 8 GHz of bandwidth per polarization per antenna. To maintain a digital processing workload matching the hardware technology on which the correlator is built on, the ALMA back-end subsystem samples the antenna bandwidth into four independent frequency slices. Reducing the processing requirement per quadrant, to only 2 GHz of instantaneous bandwidth per polarization.

For similar reasons, each correlator quandrant further segregates its output into four independent stream products. Each one associated to different baseline sets in the array (all them processing the same frequency slice.) Such that computers in the correlator back-end, each one connected to only one stream, are able to handle the associated processing workload. This means that there are four sets of four CDP computers, each set independently working on the output associated to each quadrant.

In the Spectral Specification, individual correlator quadrants are abstracted as four spectral windows. Either one single spectral window per quadrant or a set of spectral regions in the frequency range processed by that quadrant (FDM modes).

Mapping a correlator quadrant output onto four computers has a peculiar implication. Leads and lags as shown in Fig. #2 are produced by Prompt blocks that, when correlating two different antennas, are not physically close in hardware. Sending leads and lags of a baseline to two different CDP nodes in inconvenient. Because it would require to move sizable blocks of data through the network outside the correlator before being able to start processing those baselines.

For that reason, and to always collect together leads and lags associated to one individual base-line in one individual node, a Cross-Bar mechanism has been implemented in hardware. This mechanism transport leads and lags from different correlator planes to one particular output stream.

However, some network communication between nodes is still necessary. As seen in [RD02] Fig. 17.1, there are two stream outputs that include no auto-correlation results, and auto-correlations are always necessary to normalize the spectral product of any baselines in the array.

The mechanism to communicate auto-correlation products from auto-nodes to cross-nodes is implemented in AutoDataProvider. Originally, the idea was to communicate the information over the notification-channel service. However, the performance of this method was not good enough, translating into large latency delays for a cross-node to actually receive the data. A better performance has been achieved by transmitting the auto-correlations over CORBA calls (NodeImpl::setAutoDataEvent).

1.6 Spectral Data Processor Storage

In order to minimize overheads while processing new correlator dumps ingested by a node, each node instantiates and keeps in memory all possible processing objects that are needed to process ant spectral-specification. This approach is possible due to the limited number of modes under which the correlator is commanded to operate. Modes that had been commissioned and enabled for PI science are only 42 (including 4x4-bit modes.) See the *CompleteModeTable* workspace of the pertinent spreadsheet in EDM.

The SpectralDataProcessor class implements all the time-domain and frequency-domain operations to process a dump, given the spectral-specification parameters that individualize a dump. Some methods are static, meaning that their implementation do not extensively depend on specific dump parameters. But only on the number of spectral channels, provided as an input parameter to those methods.

Inspecting the correlator modes in the spreadsheet, one can see that there are only 8 possible number-of-channels values ($[2^6, 2^{13}]$). And provided that there are 7 taper window-functions (see Magic Draw model), then, the total number of SpectralDataProcessor instances to instantiate is 56. To collect all these instances in a convenient container, the class SpectralDataProcessorStorage makes part of the CDP pipeline.

The storage class exposes methods to access different processing objects (SpectralDataProcessor) based on the spectral-specification parameters associated to a correlator dump. The pipeline implementation just needs to attach the strictly minimum set of parameters to each dump when ingested, to then be able to process them accordingly through the different software scopes that a correlator dump goes through.

SpectralProcessorTask implements the C++ thread where most of the spectral-domain processing happens (except normalization). The class instantiates as a thread, and six of them are spawned per CDP node. Their number is simple a good assumption of how many are needed to cope with the maximum possible workload, and the number of available CPU cores per node (12).

New lags to process are obtained by each SpectralProcessorTask thread by means of the LagQueue container. Each item removed from the queue corresponds to a LagOutput::Corr instance, which encapsulates all the correlation functions associated to one correlator dump (for that particular node). The same method also returns the spectral specification associated to the correlator dump, represented as a CorrConfig object. This later object is then used by the SpectralProcessorTask thread to individualize which processing object in the processor storage should be applied to each spectral window in the set of lags obtained before.

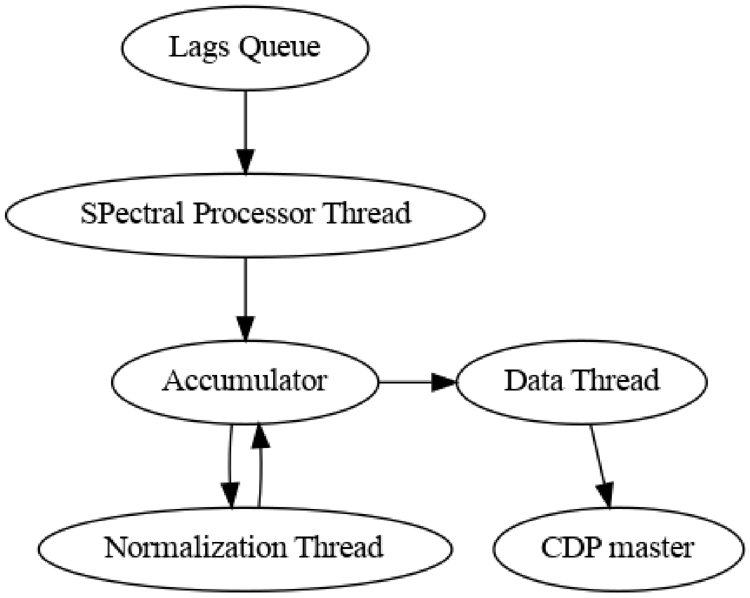

Figure 3: Software modules that implement different stages through the pipeline. Arrows in the diagram represent data flows.

Per spectral window, only two parameters are actually needed to individualize the processor object. That is, the number of spectral channels per TFB sub-band (or whole band in TDM) and the taper window function. This minimal number of parameters to deduce how to process a set of lags is a consequence of the very limited number of possible correlator modes.

1.7 Time Domain Operations

1.7.1 Lag Normalization

The numeric meaning of the lag values ingested by the CDP pipeline, is a function of how the multiply-accumulate operation is implemented in the correlator itself.

The purpose of a lags normalization operation is to remove the correlator specifics, and render values representing a correlation product. The following expression in [RD01]:

$$R_4(\tau) = 9K\frac{L(\tau) - V_s}{V_s} \tag{1}$$

represents the basic relation between the correlator output lags $L(\tau)$ and the correlation product $R_4(\tau)$. K and V_s are both a function of the correlator mode. V_s is a function of the mode and lapse of time that the correlator has integrated the lags, or dump duration.

Eq. $\#\#$ is rearranged in software to minimize the number of arithmetic operations across all possible values of τ (function of the mode's spectral resolution):

$$R_4(\tau) = \frac{9K}{V_s}L(\tau) - 9K = \frac{N_m}{V_s}L(\tau) - N_m \tag{2}$$

The actual code is implemented in the LagOutput class. The 'normalize' method requires as input parameters actual values for N_m and V_s; which are retrieved from the correlator configuration object by means of its methods getNm and getVs.

In terms of value types, the correlator outputs lags as 32-bit unsigned integers and the CDP (V_s makes possible to make all them positive), after the normalization in Equation $\#1$, moves all arithmetic operations to single precision floating-point (32-bits) until after spectral normalization. At which point, and with the aim of saving memory/storage space, cross-correlations are converted to either 16 or 32 bits signed integers plus a single floating-point scaling factor.

1.7.2 Quantization Correction

As shown in the signal processing diagram Fig. $\#1$, the analog voltage from an antenna is first sampled and digitized into 3-bit samples upstream from the correlator hardware (in the antenna cabin itself).

In FDM mode, 3-bit samples from an antenna go through each of the 32 digital filters in a TFB bank. Internally, a many-bits binary representation is used to implement the corresponding digital filter functionality (two many-taps filters in series), to finally optimally resample the filtered result and produce a 2-bit output towards the correlator electronics. 2-bit is the native binary representation that the correlator electronics can receive from the station electronics stage.

In TDM mode, the 3-bit input samples bypass the filters in the TFB board, and their 2-bit representation is obtained by simply discarding the least-significant-bit. Dropping the least significant bit of a 3-bit sampler, is equivalent to a 2-bit sampler using a sub-optimal voltage-threshold equal to two times the optimal 3-bit voltage threshold. Consequently, to attain an optimal quantization efficiency in TDM modes, it is necessary that the signal level from each antenna is amplified 1.4 dB before the 3-bit digitizer.

The above implies that in TDM and FDM modes, and under correct operational conditions, input samples presented to the correlator are equivalent to samples produced by an optimally tuned 2-bit sampler. Consequently, the same 2-bit quantization correction algorithm applies to both mode types. The technical background of the algorithm can be consulted in [RD01] and and [RD07]. The corresponding C++ header file is QuantizationCorrector.h.

The CDP software implementation applies two different algorithms based on the input correlation regime. As described in [RD01], during a low correlation regime ($\rho < 0.2$), the dependency between the unknown correlation factor ρ and the correlator output R_4, is linear enough for a fixed 5th degree polynomial. For higher correlation regimes, a more accurate polynomial representation is needed. And in that case, the algorithm resorts to the a so called Van-Vleck correction curve; which is based on a spline polynomials.

Adjusting the Van-Vleck spline polynomial is time consuming, and given that fitting the spline is a function of the signal level from both antennas (see QuantizationCorrector constructor) the total computing time scales with the square of the number of antennas in a sub-array. This situation introduces a too long processing delay to the pipeline, that for big sub-arrays can cause a timeout error. For this reason, instead of calculating the spline curve each time, a set of pre-computed polynomials is always kept in memory. Instead of fitting a spline each time, the CDP just need to look for the closest available corrector object based on the signal level from both antennas in a base-line.

A grid of QuantizationCorrector is implemented as a two dimensional matrix container in QuantizationGrid. Each side of the matrix represents the possible signal level for each antenna in a baseline. The density of rows and columns is such that the distance between the measured cross-product signal level, and that hardcoded in each node of the grid, introduces a correction error never bigger than 0.08% across the whole grid. See Twiki for additional details.

Up to this point, the description applies to correlator modes in which 2-bit samples are correlated with 2-bit samples. That is to say, 2x2-bit correlation modes. The correlator also supports 3x3 and 4x4-bit modes. Of which only a sub-set of the 4x4-bit modes had been implemented in software.

The quantization correction applied to 4x4-bit modes is equivalent to that used for 2x2-bit modes. That is to say, two different interpolation algorithms depending on the current correlation regime, and a grid of pre-computed corrector objects to minimize the time taken to correct each baseline.

To select the 2x2 or 4x4-bit version of a corrector, two different QuantizationGrid objects are instantiated by the CDP pipeline. The class' constructor makes that possible by accepting as input parameter the number of digitized levels (*nlevels*). Parameter that can take only two valid values: 4 and 16.

1.7.3 Window Function

Correlation functions are measured by the correlator as a finite sequence of lags. The application of an FFT algorithm to extract spectral information from those lags, assumes that the correlation function is periodic outside the measured interval. All this together implies that the recovered spectral product needs to deal with spectral leakage between spectral channels.

Conditioning the time-domain signal such that the leakage is maintained under control for the purpose of the instrument, a tapering function is applied to the lags before the frequency-domain transformation. This is commonly known as a windowing function.

For that purpose the CDP software defines in Taper.h a C++ class that once instantiated, it can thereafter be used repeatedly to apply a given window function to different lag buffers.

To minimize recalculations of a given function, the CDP pipeline keeps in memory the instantiation of all possible objects to be used. That is to say, of all possible combinations of number of lags in a correlation function and type of window function to use. As represented by parameters *N* and *windowType* in the Taper class constructor.

The actual function to apply can then be selected in a per spectral window basis, by means of *windowFunction* in the spectral specification. Tapering window functions specified for ALMA are those listed below. All them defined as an enumeration from a Magic Draw model (ASDM_Enumerations.mdzip):

Enumeration	Window Function
UNIFORM	rectangular (no tapering)
HANNING	Hann
HAMMING	Hamming
BARTLETT	Bertlett
BLACKMANN	Blackman ($\alpha = 0.16$)
BLACKMANN_HARRIS	Blackman–Harris
WELCH	Welch

The frequency behavior (transfer function) of each windowing function is different and their suitability is application dependent. The effective spectral channel width (spectral resolution) after the FFT, is a function of the window function applied. ALMA has made the HANNING function its default. See ALMA Technical Handbook [RD08] for some additional details.

1.7.4 Side-band Separation

ALMA bands 9 and 10 are Double Side-Band (DSB) receivers. That is, they deliver an analog signal that combines together the lower and upper side-bands around the LO mixer frequency. In Interferometry and Synthesis in Radio Astronomy [RD09] (section 6.1.11) a procedure is described to separate lower and upper signals. The procedure is based on having access to the spectral-domain cross-correlation of a baseline during two different combinations of an LO phase offset applied to one of the antennas (zero phase difference and $+90^\circ$ phase difference.)

In ALMA, the different phase states in a baseline are generated by means of a positive offset of 90° applied to the first LO of each antenna in the array. This means that cross-correlations out of the correlator, contain three different phase states for each baseline: 0°, $+90^\circ$ and -90°. This is a consequence of the orthogonal sequences used to drive the LO phase on each antenna. See [RD10] for details about the actual Walsh functions used here.

Accordingly, per correlator dump the CDP pipeline receives three independent integrations or *bins* per antenna intersection. Each bin is associated to one of the three phase combinations described above. The CDP algorithm proceeds then to combine the bins such that time-domain cross-correlation functions are independently generated for the lower and upper side-band signals. Formalized by the following two expressions:

$$U = (Z + i * (P - M))/2. \tag{3}$$

$$L = conjugate(U) \tag{4}$$

where Z, P and M are cross-correlations bins for zero, plus 90° and minus 90° phase offsets. And U and L are the upper and lower side-band cross-correlation functions.

The algorithm produces U and L lags into two separated memory buffers, which are associated to two independent spectral windows. Thereafter, upper and lower side-band lags are processed as otherwise independent and normal spectral windows. That is to say, what's specific to side-band separation, stops after the two expressions above had been applied to the three bins of lags produced by the correlator.

The actual implementation in software starts in LagOutput::Corr::dpi2corr. This method is used from LagQueue::getLags each time a set of lags is removed from the input queue. From within dpi2corr the corresponding method *sidebandProcessingNormal* or *sideband-Processing90d* are used to process the lags. Depending on the side-band separation intent (none or 90d phase switching).

Both methods normalize the lags as prescribed by Equation #1. For side-band separation the normalization is applied to each one of the three bins, and V_s is adjusted by a factor that takes into account the actual time that each bin is integrated during a dump duration. The factors are the same for all antennas and they do not change due to the actual Walsh sequence assigned to one or the other antenna. Their values are $Z_f = 0.5$ and $P_f = M_f = 0.25$.

After normalization, the method LagOutput::Corr::dpi2corr is invoked two times on the same Z, P and M memory buffers, which are not modified each time the method is invoked. Two additional input parameters to the method control whether to compute U or L lags and the memory pointer where to store the result.

One interesting detail to note of the expressions above, is that they are both complex-valued. Even though, as said before, after this point in the pipeline both spectral windows follow the standard pipeline stages as otherwise normal spectral windows do, the actual FFT operation is a complex-to-complex one, where real-to-complex is the one applied to other spectral windows. See SpectralDataProcessor::doFFT and its boolean *isComplex-CrossCorrelation* input parameter.

1.8 Frequency Domain Operations

1.8.1 Correlation Function to Power Spectrum

As a primer to this section, is convenient to consider the following in [RD09] (section 8.8.2):

> In spectral line observations, measurements at different frequencies across the signal band are required. These measurements can be obtained by digital techniques using a spectral correlator system, which is commonly implemented

by measuring the correlation of the signals as a function of time offset. The Fourier transform of this quantity is the cross power spectrum, which can be regarded as the complex visibility as a function of frequency.

The CDP pipeline takes the cross-correlation functions generated by the correlator and converts them to their power spectrum representation. The process is based on the FFTW library.

Encapsulated by the C++ class FFT, the pipeline applies the *forward* method to convert from lags to spectra. The actual algorithm is based on [RD01] and, therefore, the output frequency channel convention is to always shift them half a channel to eliminate any special consideration for the first and last channels (otherwise they would be half-width channels that contain just half of the expected power.)

FFT::forward is implemented such that real-to-complex or complex-to-complex operations can be selected by a boolean input parameter. The value of the parameter is deduced from the spectral-specification information associated to each set of lags (CorrConfig).

1.8.2 Bandshape Correction and Stitching of Poly-phase Filter Bank Slices

The following applies only to FDM modes. That is, modes in which the signal samples had been filtered by the TFB. Bandshape correction is always applied to undo the digital filter shape; which is precisely known from its filter taps, see [RD06]), and stitching occurs only when the FDM mode involves more than one filter sub-band (bandwidth larger than 62.5 MHz).

The digital implementation of band-pass filters in the TFB FPGAs is such that that signal samples are converted to a higher order bit-depth. The last filter stage produces 9-bit output samples, which have to be re-quantized down to a 2-bit representation before the correlator could ingest the samples. Re-quantizing from 9 to 2 bits is a process that to be optimal (preserve dynamic range) must take into account the actual 'amplitude' of the signal being processed.

The procedure consists on measuring the signal level once, record the reading, and re-quantize based on that level until the measurement is updated (new calibration). From a software point of view, measuring and recording the signal level is encapsulated as a correlator calibration; which records the so called scaling factor that must be applied to the spectral amplitude after the FFT, before the actual stitching of individual sub-bands. Not doing so will normally cause abrupt steps from sub-band to sub-band ('scaling').

The three different actions are implemented in the following SpectralDataProcessor methods:

- doBandShapeCorrection,
- doApplyScalingFactors and
- doSpectralStitching.

1.8.3 Power Spectrum Normalization

The Accumulator class ingests spectral-domain results from SpectralProcessorTask threads. That is, it ingests spectral-domain correlator dumps which are now ready for:

- further integration time in software (one to many dumps),
- spectral normalization,
- spectral averaging,
- packetization in fixed point representation (visibilities only).

The Accumulator thread is in charge of integrating the dumps and when an integration has completed, then, it delivers the result to a NormalizationThread, which applies the normalization convention in [RD05] and it converts visibilities from floating-point numbers to 16 or 32-bit signed integers (as prescribed in [RD05] as well).

For normalization purposes, the critical methods are implemented in SpectralNormalization. In particular, the public methods *normalizeAutocorrelation* and *normalizeAutocorrelation*, which are applied to spectral results before the corresponding spectral averaging.

If there is no spectral averaging to apply (number of input channels equal to output number of channels), then, converting to integers happens by means of Normalization::float2integer. And when there is some averaging to apply (two or more channels to average together as one channel), then, calculating the average itself and conversion to fixed-point numbers happen at the same time in Normalization::average; which is a template method that recognizes the bit-depth to apply based on the actual type of the output buffer.