Software Design Document

Jim Pisano

*National Radio Astronomy Observatory*

---

**Owner**  
Jim Pisano (jpisano@nrao.edu)

**Keywords:** correlator software design real-time

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author Signature:</strong> Jim Pisano</td>
<td><strong>Date:</strong> 2003-08-25</td>
</tr>
<tr>
<td><strong>Approved by:</strong></td>
<td><strong>Signature:</strong></td>
</tr>
<tr>
<td><strong>Institute:</strong></td>
<td><strong>Date:</strong></td>
</tr>
<tr>
<td><strong>Released by:</strong></td>
<td><strong>Signature:</strong></td>
</tr>
<tr>
<td><strong>Institute:</strong></td>
<td><strong>Date:</strong></td>
</tr>
</tbody>
</table>
Change Record

<table>
<thead>
<tr>
<th>REVISION</th>
<th>DATE</th>
<th>AUTHOR</th>
<th>SECTIONS/PAGES AFFECTED</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td></td>
<td>Pisano</td>
<td></td>
<td>Initial Draft</td>
</tr>
<tr>
<td>0.2</td>
<td>2003-05-16</td>
<td>Pisano</td>
<td>All</td>
<td>Incorporated IDR comments</td>
</tr>
<tr>
<td>0.3</td>
<td>2003-07-29</td>
<td>Pisano</td>
<td>All</td>
<td>Incorporated PDR comments</td>
</tr>
<tr>
<td>1.0</td>
<td></td>
<td>Pisano</td>
<td>All</td>
<td>Incorporated CDR1 comments</td>
</tr>
</tbody>
</table>

Table of Contents

1 Introduction .......................................................................................................3

1.1 Purpose ..................................................................................................3
1.2 Scope .....................................................................................................4
1.3 Glossary .................................................................................................4

2 Requirements ....................................................................................................5

2.1 SSR Requirements ......................................................................................5

3 Architecture .......................................................................................................5

3.1 Overview......................................................................................................5
3.2 CCC Packages and Functional Overview ..................................................7
3.2.1 ACS CCC Interface ...............................................................................7
3.2.2 CCC Monitor ........................................................................................8
3.2.3 CCC Command Dispatcher .....................................................................8
3.2.4 Geometric Delay .................................................................................8
3.2.5 Subarray Management .........................................................................8
3.2.6 Maintenance........................................................................................8
3.2.7 Correlator Hardware Control ...............................................................9
3.2.8 Array Time Interface ..........................................................................9
3.2.9 Correlator Configuration Validation ...................................................10
3.2.10 Correlator CAN Commands ..............................................................10
3.3 Detailed CCC Package Descriptions ........................................................10
3.3.1 CCC Command Dispatcher ...................................................................10
3.3.2 Geometric Delay ...............................................................................11
3.3.3 Subarray Management ........................................................................12
3.3.4 Correlator Hardware Control ............................................................13
3.3.5 CCC Monitor .......................................................................................13
3.3.6 Correlator CAN Commands ...............................................................14
3.4 Observation Control Sequence ..................................................................15
3.5 CDP Packages and Functional Overview ..................................................16
3.5.1 ACS CDP Interface ...............................................................................17
3.5.2 CDP Monitor .......................................................................................18
3.5.3 CDP Command Router .......................................................................18
3.5.4 Lag Processing ....................................................................................18
3.5.5 CDP Maintenance ...............................................................................18
3.5.6 Spectral Processing ............................................................................18
1 Introduction

1.1 Purpose
This document provides a high-level system design for the correlator computer system software which performs the following tasks:

- Configure the correlator hardware and data processing parameters for an observation.
• Process raw lags from the correlator hardware into raw spectral blocks
• Transmit the correlated spectral data to the Archive.
• Monitor and publish status information for the correlator hardware and correlator subsystem computers.
• Provide an interface for maintenance and diagnostics

The main constraints on this subsystem include the real-time demands placed on it by the high data rates of the correlator hardware output, the synchronization of execution with the array-wide timing events and intensive data processing loads due to correlator hardware output rates.

1.2 Scope
The design described is valid for the computer subsystem which controls, configures, monitors and processes the raw data from the NRAO baseline correlator. This design also covers the 2-antenna prototype correlator to be delivered to the Antenna Test Facility (ATF) at the VLA site.

1.3 Glossary
The standard ALMA glossary of terms can be found at:

http://www.alma.nrao.edu/development/computing/docs/joint/draft/Glossary.htm

Common terms used in this document are:

ACS  ALMA Common Software, a software infrastructure to support a distributed software system utilizing CORBA

APC  Atmospheric Phase Correction, path length correction due to atmospheric fluctuations

CAI  Correlator Antenna Input
Each set of 4 baseband pairs at an antenna are routed to correlator inputs in the correlator hardware. These 0-based inputs are called ‘Correlator Antenna Inputs’. A mapping exists between antenna numbers and correlator antenna inputs.

CCC  Correlator Control Computer, a VME PowerPC computer with Ethernet, CAN, RS-232 and digital I/O

CDP  Correlator Data Processor, a PC-cluster which obtains raw lags from the correlator, does digitization corrections, windowing, FFTs, and atmospheric phase corrections to provide spectral data blocks.

DPI  Data Port Interface, the high-speed, 32-bit binary interface between the correlator Long Term Accumulator and the CDP.

FDS  Fast Data Switch, a component of the ALMA archive which receives the correlator spectral data blocks.

LTA  Long Term Accumulator, the main computer interface for control and data processing of the correlator hardware.
QCC Quadrant Control Card, a type of interface card in each correlator quadrant which monitors quadrant voltages and temperatures.

SCC Station Controller Card, a type of interface card which monitors and controls the correlator station and FIR filter cards.

TE Timing Event – an array-wide, highly-accurate 48 ms timing signal which is used to synchronize distributed hardware and computers.

WVR Water Vapor Radiometry

Dump At the end of a correlator switching cycle, raw lags are transferred to the CDP. This transfer is called a dump.

Lag The integrated product of two digitized signals, one delayed in time with respect to the other. If the two signals are from the same antenna source, then it is an ‘auto-correlation product’, else if the signals are from different antennas, then it is a ‘cross-correlation product.’

2 Requirements

2.1 SSR Requirements

General scientific software requirements are found in [1]. Refinements of the SSR as they apply to the specific details of the correlator hardware and computer systems are found in [2].

3 Architecture

3.1 Overview

Figure 1 shows the architectural overview of the correlator subsystem. It shows the five subsystems to which the correlator interfaces with data flows highlighted.
The Control system provides configuration and control information, geometric delay data stream, antenna blanking streams, and WVR value streams for each antenna. The Executive receives events which signal the completion of integrations assisting it in tracking the progress of observations. The telescope calibration subsystem accepts channel average and spectral results in order to provide atmospheric phase correction coefficients to the control system which are passed on to the correlator as configuration parameters. The Quick Look pipeline accepts channel average and spectral results to provide real-time images. Finally the Archive accepts spectral results, channel average data and correlator monitor data.

The correlator subsystem is divided into two computer systems: the CCC and the CDP (see section 3.7 for a description of the physical architecture). The CCC is responsible for controlling and monitoring the correlator hardware and the CDP is responsible for processing raw correlator lag data. ACS [3], a middleware layer which is based on CORBA, provides a variety of services and is used extensively throughout the correlator subsystem. These services include remote function calls, message logging, error reporting, device property monitoring and CORBA notification service. TAO [4] is the ORB used exclusively for the CCC and CDP computers. The use of TAO enables us to utilize some of the real-time CORBA extensions currently not included in ACS.

Both the CCC and CDP are ‘ACS Distributed Objects’ (DOs) whose interfaces are accessible as distributed nodes in the sense of the CORBA paradigm. They implement the porous container design pattern due to their real-time needs. It is important to note that only the CCC is a ‘public’ device, i.e., it is the only device accessible to computers outside of the correlator subsystem. In general, the CDP is not accessed from external computers as to minimize the interfaces, although there may be exceptions to this rule. The CCC and CDP communicate with one another and the CCC acts as a communication gateway to the CDP for configuration and control.
The correlator device follows a ‘configure-and-run’ model where the hardware is configured for a specific observation and operates in that mode until it is reconfigured or commanded to stop.

### 3.2 CCC Packages and Functional Overview

Figure 2 shows package diagrams for the CCC. A brief summary of each package follows.

![CCC Package Diagram](image)

#### 3.2.1 ACS CCC Interface

This package provides an interface layer needed for common ACS services which provide a wrapper for the ACE Orb (TAO), CORBA remote invocation services, logging services, configuration database, notification services and ACS properties. This package has the following responsibilities:

- instantiates the distributed object component and connects to the ACS Activator container utilizing initial values from a configuration database
- obtains references to the MACI manager. See [3] for more information on MACI.
- subscribes to the notification channel that supplies geometric delays events
3.2.2 CCC Monitor

This package is responsible for constructing ACS properties for the correlator hardware and the CCC. As properties are self-contained within ACS, there is actually little to describe. See section 3.5 of [5] for details. We plan to use CAN and memory properties. This package implements ALMA.Correlator.CCC_Monitor interface in [6].

The ALMA.Correlator.ObservationQuery interface in [7] can be implemented as a set of properties. It is envisaged that this interface will most often be used in debugging the correlator software during development and commissioning. Thus it will be a part of correlator diagnostics in this package.

3.2.3 CCC Command Dispatcher

This package is responsible for managing commands which are to be invoked on the correlator hardware. This includes queuing up commands in a time-order sequence for time-critical commands, verifying command parameters and timings and synchronizing commands with array time. This package also routes commands to the CDP.

Commands have a time stamp and a 3-level priority: high, medium, and low. Time-stamped commands with high priority are placed at the head of the queue in time stamp order. Normal priority commands are ordered by their time stamp. Low priority commands may or may not have a time stamp and essentially are tacked on to the end of the command queue. Any commands which have expired their time stamp, i.e., arrived to late to execute, are discarded and an error is logged.

3.2.4 Geometric Delay

This package distributes antenna-based geometric delay parameters to correlator hardware registers. The control system’s geometric delay model server supplies these delay parameters before a scan and as needed throughout the scan. The digitizers at the antennas can accept 4 bits of fractional delay of one sample (at 4 x 10^9 samples/sec), i.e., 0 – 0.25 ns with a resolution of 15.625 psecs (0.25ns / 16). The correlator hardware accepts a 16-bit coarse delay value which represents the remaining delay up to a value of ~16.384 µs (0.25ns x 2^16-1). Any residual delay is then removed in the CDP as discussed in section 3.6.2.6.

3.2.5 Subarray Management

This package manages the current subarray definitions used within the correlator hardware. A subarray in this context means a list of antenna numbers (CAIs really) which are configured and controlled as a unit. This package:

- Assists with correlator configuration validation,
- provides antenna information for a given subarray identifier,
- associates subarray identifiers to a group of antenna numbers for correlator hardware configuration, etc.

The 2-antenna prototype will be able to support two subarrays each having two simultaneous single dishes.

3.2.6 Maintenance

This package is responsible for managing maintenance functions with the correlator hardware and CCC computers. Maintenance implements the ALMA.Correlator.Maintenance interface in [7].
Examples of maintenance for the correlator hardware would be the upgrade of FPGA images or microprocessor code both of which are downloadable to on-board flash memory. Maintenance also performs diagnostics and reports their outcome. It is important to note that no downloads are required for standard correlator observing modes.

Thirdly, this package is responsible for configuration information related to the moving of an antenna to a new pad. The physical antenna number is mapped to an appropriate correlator antenna input. This mapping also involves a optical patch panel at the central electronics building which connects optical fibers from a given pad to a correlator antenna input. Also the fixed delay offset for the new fiber cable length associated with this antenna is set.

Most maintenance functions occur when the correlator hardware and computers are off-line in a non-operational mode. The exception to this is when an antenna is moved to a new pad and the correlator may remain on-line but the specific correlator inputs are not being used. Some diagnostic functions may operate in on-line modes, although the exact details are TBD.

3.2.7 Correlator Hardware Control
This package queues and delivers control and configuration commands to the correlator hardware for an observation. This package also tracks the current state of the correlator hardware and validates correlator configurations as needed in real time.

3.2.8 Array Time Interface

Array time is the fundamental time system of the interferometer utilizing TAI and is maintained centrally by a master clock computer (ARTM). Distributed clocks in the CDP and CCC are slaved to the master clock and kept synchronous by utilizing an array-wide 48 ms timing event (TE).

This package is responsible for interfacing between the TE hardware signal and internal software packages that require command synchronization and time stamps. As this package must interact with almost all packages in Figure 2, lines are not drawn to show these relationships as it would clutter the diagram too much.

Also this package implements the interface specified ALMA.Correlator.ArrayTime in [7]. Array time is set by first having the ACC provide the distributed clock with a TimeSource object which it can use to request the array time of the next TE from the ARTM. The ACC checks that the distributed clock’s array time is synchronized and, if not, commands the distributed clock to retry obtaining the array time from the ARTM. For details on the control subsystem side, see section 3.7 of [8].

Once set, the distributed clock updates its internal version of array time by counting TEs and updating array time by 48ms. The distributed clock must ensure that no TE is missed as this obviously corrupts the distributed clock’s version of array time. The use of watchdog timers can provide the needed reliability. If a TE is missed, the distributed clock is in an unsynchronized state which the ACC can query and, if necessary, cause the distributed clock to reset itself.

The correlator hardware can provide both a 48ms and a 16ms tick to the CCC and CDP computers. It is most likely the CCC will use the 48 ms tick. Although the CAN nodes will not explicitly track array time, the CCC will keep track of the CAN nodes' current TE count in order to specify a specific TE on which to execute a command.
3.2.9 Correlator Configuration Validation

There are two types correlator configuration validation covered by this package: *static* and *dynamic*. Static validation performs a simple check that the XML configuration string matches the XML schema and that all of configuration parameters are compatible. This static checking will be written in Java and used by the Observation Preparation Tool to ensure that all configurations are within the bounds of current correlator hardware capabilities, e.g., number of antennas, number of basebands, BBCs, etc. It is envisaged that current correlator hardware capabilities will reside in a database accessible to any subsystem which needs this type of information.

Dynamic validation determines if the configuration fits into the current running state of the correlator hardware or if two configurations scheduled for the same time conflict. This is a complex process which can only be evaluated at run time and is handled by the Correlator Hardware Control package.

3.2.10 Correlator CAN Commands

This package is a general one which actually represents three specific packages: one for LTA CAN commands, SCC CAN commands and QCC commands. This package provides a simplified interface to the correlator hardware control, monitoring, maintenance and geometric delay packages to the lower level hardware.

3.3 Detailed CCC Package Descriptions

This section provides detailed class diagrams and descriptions of the more complex CCC packages.

3.3.1 CCC Command Dispatcher

The command dispatcher provides distribution of configuration and control commands for all internal devices of the correlator subsystem – the correlator hardware, CCC and CDP. This division provides a single point of control for external subsystems.

![Command Dispatcher Package Class Diagram](image)

The Command class is base class for all commands that are received via the ACS CCC Interface package. CCC_CmdDispatcher validates commands, flags and rejects erroneous commands. A time tag as to when the command should be executed and a priority are assigned. Note that commands are not scheduled to execute, this is done within the Corr. HW Control package.
3.3.2 Geometric Delay

This package is responsible for subscribing to the geometric delay event channel and distributing the quantized delay values to the correlator hardware.

It is envisaged that on a periodic basis, say every 10 seconds, the model server transmits a set of geometric delay parameters with timestamps which define a valid temporal range, antenna numbers for which they apply, the two quantized delay values and a set of coefficients of the delay model equation. For example, each entry of the set of delay parameters would be as such:

```c
struct DelayValues
{
    // array time when delay values are valid
    Time startTime;
    // array time when delay values are no longer valid
    Time stopTime;
    // antenna number (0-based) to apply quantized delay values
    unsigned byte antennaNumber;
    // 4-bit value of quantized delay for digitizer
    unsigned byte fractionalDelay;
    // 16-bit value of quantized delay for correlator hardware
    unsigned short courseDelay;
    // model coefficients used to remove residual delay in CDP
    double coefficients[5];
}
```

If delay events arrive late, i.e., their start time has passed, then an error is logged. If the stop time has not passed, then these delay values are used. If both the start and stop times have passed, an error is logged and the delay values are discarded. The minimum delay update rate will be 48 ms which appears to be in line with Thompson’s calculations [9] of 32 ms.

![Figure 4 CCC Geometric Delay Class Diagram](image)

**Figure 4 CCC Geometric Delay Class Diagram**

*ProcessDelayParameters* subscribes to the geometric delay event channel (via *DelayEventChannel*) receiving the delay events. It queues *HardwareDelay* objects which define the coarse delay value and when to apply it to the correlator hardware. It sends them to the Corr. HW Control package to be applied using station control card CAN commands.

The sequence diagram for applying the geometric delay is show in Figure 5. The geometric delay model server supplies initial delay values before the scan starts and as needed during the scan.
3.3.3 Subarray Management

Subarray management essentially maintains a list of subarrays each defined by an identifier, a list of 0-based antenna numbers with corresponding 0-based correlator antenna inputs. **IMPORTANT: The antenna numbers from the control system must be mapped to correlator antenna inputs (CAIs) which are inputs the correlator chips.** The CCC and the control system must coordinate the mapping of fixed antenna numbers to pad identifiers to correlator antenna inputs. This mapping also involves a optical patch panel at the central electronics building which connects optical fibers from a given pad to a correlator antenna input. This mapping is part of the `ALMA.Correlator.Maintenance` interface.

The `AntennaList` class maintains a sequence of the following structure:

```c
struct {
    unsigned char antennaNumber;
    unsigned char corrAntInput;
};
```

which maps antenna numbers to CAIs. Correlator antenna inputs are used to configure the correlator hardware while corresponding the antenna number is used in the CDP’s data output block identifying the original antenna numbers in the subarray as defined by the control system.

`SubarrayManger` creates and destroys `AntennaList` objects as subarrays are formed or deleted. It also assists in dynamic correlator configuration validation by determining the intersection of two `AntennaList` objects.
3.3.4 Correlator Hardware Control

All correlator hardware control commands originate at the control system via the ACS CCC interface package. These commands are routed to the CCC command dispatcher package which is responsible for queuing and routing commands.

Figure 7 shows the CorrHWController class which converts a command from the control system to the appropriate CorrCANCmd object (see section 3.3.6). The CorrHWController queues the commands, creates the appropriate CAN command byte stream and, utilizing a scheduler, sends them to the correlator hardware CAN node(s) at the correct TE.

3.3.5 CCC Monitor

The CCC Monitor package provides status information about the CCC software. It also contains two sub-packages: correlator hardware monitoring and CCC hardware monitor. This package contains two sub-packages: Correlator hardware monitor and CCC hardware monitor.
3.3.5.1 Correlator Hardware Monitor

These are the “standard” properties for the correlator hardware like voltages, temperatures, etc. They are managed through the QCC for each quadrant consequently providing 4 sets of voltages and temperatures.

It is important to note that the correlator hardware is self-monitoring in the sense that if an out-of-voltage, or over-temperature condition is encountered which may damage the hardware, it shuts itself down. How much of the hardware is powered down is still TBD. For now, the CCC monitoring software detects these error conditions and delivers monitor data to the monitor store which can then be utilized to flag bad data. This leads to ‘correlator quadrant blanking’ where data are blanked on a quadrant basis. The CCC then notifies the CDP which quadrants are malfunctioning with the CDP ignoring any data from the affected quadrant.

3.3.5.2 CCC Hardware Monitor

Hardware dependent code retrieves monitor information from the CCC computer hardware. The current CCC hardware uses a SIP VME chassis which has voltage, temperature and fan speed monitor data available as properties.

3.3.6 Correlator CAN Commands

The correlator utilizes a CAN interface to communicate with the CCC. It is important to note that the protocol used by the correlator hardware as described in [10] is related to the ALMA M&C protocol defined in [11] in that it supports some of the basic ALMA M&C functions like node identification and its master-slave architecture, but not the timing specifications. More importantly, the correlator CAN protocol differs from the ALMA M&C protocol by providing multicasting of CAN messages to a range of CAN nodes and to transmit multiple packet data structures. Multicasting is a powerful tool for the correlator as there are many nodes which require identical information allowing efficient configuration and control.

The class diagram (Figure 9) shows the layout of basic CAN protocols as a base class and *mix-in* classes from which specific commands inherit. The base class, CorrCanCmd contains generic functionality used by the CorrHWControl class. The leaves of the class hierarchy then define the specific command either as a multicast or singlecast command with CANMulticast or CANSinglecast providing the necessary addressing capability.

The currently available CAN commands are listed in the Appendix [5.2].
3.3.6.1 LTA CAN Commands
Software associated with CAN commands to configure, control and monitor the LTA. Included in the LTA are three types of nodes – the LTA card, the Final Adder card and the 32-bit data port interface card (see [16] for details).

3.3.6.2 SCC CAN Commands
Software associated with CAN commands to configure, control and monitor Station Control Card (for Station & FIR Filter cards).

3.3.6.3 QCC CAN Commands
Software associated with CAN commands to monitor the quadrant control card which provide voltage, current and temperature monitors for each quadrant.

![Figure 9 CAN Commands Package Details](image)

3.4 Observation Control Sequence
Figure 10 is a sequence diagram which shows the time-ordered sequence for a configuration and start of an observation with the correlator subsystem from the viewpoint of the CCC. Geometric delay configurations were covered earlier and are omitted here. Note that there is some interaction with the CDP here. This will be discussed in further detail.

First, a subarray is defined. All subsequent configuration and control commands for the observation refer to this subarray identifier.

A complete correlator configuration contains a time stamp as to when the correlator starts using this configuration – this can be viewed as 'start observing with a specified configuration'. Once the configuration is received at the CCC, the CCC transmits the relevant configuration information to the CDP along with the time stamp. The CDP then configures itself at the correct TE in synchronization with the correlator hardware being configured at the same TE so that lags are correctly processed according to the new configuration. At the appropriate TE, the CCC commands the correlator hardware to use the new configuration. The observation continues until the requested number of integrations is reached or the observation is commanded to stop.
3.5 CDP Packages and Functional Overview

Figure 11 shows packages for the CDP. A brief summary of each package follows. Table 1 shows deployment details as to which packages exist on the (single) master or (multiple) compute nodes. Most are shared between the two.
Figure 11 CDP Package Diagram

<table>
<thead>
<tr>
<th>Package</th>
<th>Master Node</th>
<th>Compute Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS CDP Interface</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>CDP Command Router</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>CDP Monitor</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Lag Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDP Maintenance</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Spectral Processing</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>CDP Configuration</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Data Publisher</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Subarray Configuration</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>RTAI</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Data Port Interface</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Array Time Interface</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Cluster Admin</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Atmospheric Phase Correction</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Double side band receivers</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

Table 1 CDP Node Package Deployment

3.5.1 ACS CDP Interface
This package provides an interface layer needed for common ACS services providing a wrapper for the ACE Orb (TAO), CORBA remote invocation services, logging services, configuration database, notification services and ACS properties. This CORBA interface is limited in its access, i.e., most command invocations are from the CCC while ACS properties, notification channel subscriptions and publications are “public”.

Notification services for published events and subscribing to data streams are managed by this package. Subscribed data are then transmitted to the appropriate compute nodes via the CDP command dispatcher package. ACS will be used in the master and compute nodes.

This package is responsible for interfacing functional code with the CORBA interface. This package has the following responsibilities:

- constructs and manages the distributed object component and connects to the ACS Activator container
- obtains references to the MACI manager
- creates (or attaches to existing) notification channels to publish integration events, logging information, correlator spectral and channel average bulk data
- subscribes to notification channels that supply antenna blanking, geometric delays and WVR receiver values.
- instantiates CORBA reference objects for the ALMA.Correlator.CDP_Control and ALMA.Correlator.CDP_Monitor private interfaces.
- provides inter-node communication for commands and data channels.
3.5.2 CDP Monitor

This package is responsible for constructing ACS properties for the CDP. As properties are self-contained within ACS, there is actually little to describe. The same comments as in section 3.2.2 apply here.

The master node monitors each compute node to check its health, i.e., ‘pinging’. If the master detects that a compute node has rebooted, it logs this error and reconfigures the affected node.

3.5.3 CDP Command Router

This command router routes commands which are to be invoked on the CDP. This package runs in the master node determines what target compute nodes need the current command and sends it to them. Although a history of commands is kept here, no queuing is performed, i.e., commands are immediately sent out. Also CDP configurations are saved here in case a compute node needs its configuration parameters reloaded.

3.5.4 Lag Processing

This package is responsible for buffering raw lags from correlator hardware. It receives the raw lags from the Data Port package after each correlator dump and queues the lags in the order received and by bin number. These lags remain in this fashion for a certain period (nominally one second) allowing for antenna blanking which can remove lags as needed (see section 3.6.1 for details). The output of this package are groups of lags called ‘lag sets’ which are processed into spectra with the Spectral Processing package.

3.5.5 CDP Maintenance

This package is responsible for managing maintenance functions CDP computers. An example of maintenance for the CDP nodes would be the regular cleaning of computer filters. The maintenance package is also responsible for performing diagnostics and reporting their outcome.

3.5.6 Spectral Processing

This package manages processing of lag sets into raw spectra. Lag sets are received from the Lag Processing package and FFT’d with resulting spectra published to the FDS and, in turn, to the archive, telescope calibration subsystem and quick look pipeline. Spectral processing details are discussed in section 3.6.2.

3.5.7 CDP Configuration

This package is responsible for receiving, tracking and managing the configuration of the CDP. Also configurations are queued at each CDP node and applied at the correct array time. Configurations arrive to the master node as an XML configuration file defined in the Correlator ICD [7]. This XML file is parsed by the master node and then sent to the appropriate compute nodes in binary form via a function call.

Each compute node handles ¼ of the baselines for a given correlator quadrant. The master node tracks which compute nodes have which configurations to facilitate the ‘assembly’ of spectral data packets from individual compute nodes into spectral data blocks for a subarray to be sent to the FDS. This assembly process is part of the Data Publisher package.
3.5.8 Data Publisher
This package publishes spectral and channel average data to the FDS. As each compute node completes the processing of spectral data, it transmits its results to the master node. The master node then assembles all of the baseline results for a given subarray and integration and publishes it to the FDS. The bulk data format is in VOTables [12] which utilizes XML headers and binary tables. See the Correlator ICD [7] for content and format. The exact software technology to publish data is TBD. A standard CORBA notification channel to the FDS will not be used as it too slow. CORBA Audio/Visual Streaming Service (which is supported by TAO [13]) will be investigated. Straight TCP/IP socket communication is also an option although the A/V service can attain throughput rates comparable to raw socket connections.

3.5.9 Subarray Configuration
Observation configurations are specific to a subarray and a correlator quadrant. Much of the discussion of subarrays in section 3.3.3 applies here. It is important to note here that the correlator antenna inputs must be mapped to the antenna numbers in the output data in order to recover the original subarray information.

3.5.10 RTAI
This package contains the RTAI real-time Linux kernel extensions. RTAI is a modular micro-kernel for which needed services are installed as Linux kernel modules. Drivers for the 32-bit data port and Array Time interfaces are implemented as kernel modules. This package contains these kernel modules in source form and procedures to modify the kernel, debugging tools, etc.

An important aspect of this package is the determination of which parts of the software run as real-time task in kernel mode vs. non-real time in user space on the compute nodes. All of the lag and spectral processing could be performed as kernel modules, but debugging is more difficult. Prototyping will determine this division. For the master node, RTAI will be used for the Array Time interface.

3.5.11 Data Port Interface
Raw lags are sent from the correlator LTA via 32-bit data port interfaces to each compute node. Low-level drivers (running as real-time tasks within RTAI) transfer the raw lag data to a shared memory area via DMA transfers and signal other tasks to begin the lag processing once the DMA transfer is complete.

3.5.12 Array Time Interface
This package is responsible for interfacing between the TE hardware signal and internal software packages that require synchronization and time stamps. The software functionality duplicates the CCC array time interface discussed previously in section 3.2.7. except for the hardware interface. The compute nodes receive the TE signal via the standard PC parallel port (‘LPT1’) while the master node receives it via an RS-485 connection.

If a compute node misses a TE, it must signal an error and blank its data while it resynchronizes with array time from the ARTM.

3.5.13 Cluster Administration
This package contains tools and procedures to monitor and administer the CDP master and slave nodes. There are several Beowulf cluster administration tools, OpenSCE [14] looks particularly promising for diskless clusters while NPACI Rocks [15] supports ‘disk full’ clusters. As there will
be other Beowulf clusters in ALMA, there is a possibility of using a common TBD administrative tool.

3.5.14 Atmospheric Phase Correction

This package performs atmospheric phase correction. The control subsystem provides path length correction coefficients as part of an observation configuration. The CDP subscribes to an event channel from each antenna which provides the WVR receiver values. The CDP applies the path length correction to each channel of the spectral results with the receipt of each new set of WVR receiver values.

3.5.15 Double sideband receivers

The use of double sideband receivers requires that separate phase states must be binned and summed separately. Four bins are required for the 0°, +90°, and -90° combinations. As the correlator hardware is unable to calculate the -90° combinations, then the CDP is responsible for calculating and summing them.

An interface is required to determine the phase of each antenna for each 16 msec sample, (e.g. the start time and 64 patterns, each 64 states long). The baseline phase is computed from the antenna phase (90°, 0°, -90°) with the results summed into two bins (90°, 0°) and normalizing correctly. The -90° is a subtraction in the 90° bin. Once the integration is complete, the upper and lower sideband visibilities are formed with:

\[ V_{\text{lower}} = \frac{V_{0°} + iV_{90°}}{2} \quad \text{and} \quad V_{\text{upper}} = \frac{V_{0°} - iV_{90°}}{2} \]

The signs may depend on the conventions of the LO system.

3.6 Detailed CDP Package Descriptions

This section provides detailed class diagrams and descriptions of the more complex CDP packages.

3.6.1 Lag Processing

At each dump, a set of raw lags are transferred via DMA to one of 2 memory buffers. Once the DMA transfer is complete, processing of the memory buffer begins.

Each raw lag set from the correlator hardware has a header which identifies the baseline using *correlator antenna inputs*, the polarization product, bin number, number of lags (in resolution of 16 lags) and other information (see section 18 in [16]). This information is carried through to the spectral processing phase to determine how spectra are summed and becomes part of the header information when the spectral data are published.
Each antenna publishes status information for each 48 ms period to an antenna blanking event channel to which the CDP subscribes. If a negative status or no status information is received for a given time period, the CDP discards the affected lag sets. Due to the potential latency in the notification channel, a TAO real-time event channel may be used [17]. Also the AntennaBlanking class will wait up to one second in order to not miss any blanking data. This introduces a one second latency in the data throughput.

Blanking introduces some subtleties regarding the integration times and durations which must be recorded for each integration:

- Requested integration duration – the observer sets this integration duration
- Actual integration duration – the requested duration minus any blanked dumps
- Integration start time – the time stamp that the integration starts
- Integration centroid time – the average time that non-blanked data was integrated. The timestamp for that data record needs to indicate the exact centroid of the data.

The BinMgmt class is responsible for placing raw lags into the correct bins. There are two types of binning envisaged. Antenna-based switching where, e.g., the nutator switches positions and lags are placed into separate bins in the correlator hardware and consequently the CDP for each nutator position. Phase switching is another form of switching which uses this 2-bin approach. Baseline-based switching is used with double sideband receiver data where different correlation products of 90° phases for a baseline are kept in separate bins.
3.6.2 Spectral Processing

Once a lag set is processed, it is ready to be converted to a spectral data block. Figure 14 shows the classes which are involved with the steps of this processing. The order of the steps are: lag normalization, quantization correction, windowing, FFT, atmospheric phase correction, fine (or residual) delay adjustment, spectral averaging and channel averaging. Figure 15 is a sequence diagram for the spectral processing which shows the order of processing steps.
Figure 14 Spectral Processing Package Details
3.6.2.1 Lag Normalization

Lag normalization is a two-step process:

The cross spectra are normalized to unity by dividing each lag value by the square root of the product of the zero\(^{th}\) lag values for the corresponding antennas, while the auto spectra are normalized by just dividing by the zero lag value. This is discussed in "Spectral normalization" of [18], and would produce lags where 100% correlation has a value of unity, except for the presence of multiplication table bias which is dealt with in the next section.

In this second step, a multiplication bias introduced at the correlator chip level in its multiplication table when calculating the correlation function is removed. This bias removal is sometimes called ‘Vs subtraction’. See section 5.1.1 for this equation.

3.6.2.2 Quantization Correction

Performs quantization correction on the lags. The algorithm for quantization correction (F. Schwab priv. comm.) is to first perform a 2-bit, four-level correction followed by a 3-bit, eight-level correction. The first correction compensates for the FIR filter decimation and the second correction compensates for the 3-bit, eight-level digitizers. See F. Schwab’s memo paper [19] which discusses his correction methods.

3.6.2.3 Windowing

Applies window function on the lags. Normally a Hann (or Hanning) window is used, but the following windows are also available: Uniform (no window), Hamming, Welch, Bartlett, Blackman and Blackman-Harris. See section 5.1.6 for the equations for each of these windowing functions.

3.6.2.4 FFT

The correlation functions are Fourier transformed to give spectra. The FFTW algorithm from MIT [20] is used due to its optimizations and fast calculations. It has an important efficiency which creates a ‘plan’ for a given size of FFT. This plan is relatively slow to calculate, but once it has
been created the actual FFT calculations use this plan repeatedly for each data set which is very fast.

3.6.2.5 Spectral Normalization
In the spectral domain, the cross correlation functions are normalized so that each spectral channel represents a fractional correlation of the power in that channel. See ‘Spectral Normalization’ of [18] for details.

3.6.2.6 Residual Delay Adjustment
As discussed in section 3.2.4, the majority of the geometric delay adjustment is performed in hardware at the digitizers and correlator. Nevertheless, small errors in the geometric delay resulting from the discrete nature of the digital delay are accumulated over the course of the integration. These residual delay errors are removed in the CDP. Residual delay adjustment may be also known as fine phase adjustment.

The spectrum is corrected for the average delay error over the integration by removing the residual delay which is defined as the difference between the unquantized delay and the quantized delay integrated over the fine delay interval, i.e., the integration duration.

The quantized delay includes the quantized delay provided by the model server which has a resolution of 15.625 psecs (1/16 of 1 sample period of 0.25 ns) plus the delay offset for a given antenna due to its cable distance from the array centerpoint to the correlator. Note that the finest resolution may be 1/32 of 1 sample which is 7.8125 psecs.

The residual delay is simply:

\[ D(n)_{\text{res}} = D(n) - D(n)_{\text{quant}} \]  

where \( D(n) \) is the total, unquantized delay using the geometric delay model coefficients provided by the geometric delay model server. The delay model provides delay information for each antenna based upon its distance from the array’s geometric center.

As delays are baseline-based, the actual residual delays are the difference of two antenna values for a baseline, leading to:

\[ D_{\text{baseline}} = D(n)_{\text{res}} - D(m)_{\text{res}} \]  

\( D_{\text{baseline}} \) can be used to apply the residual delay correction.

3.6.2.7 Atmospheric Phase Correction
The path length correction due to the atmosphere as a function of the intermediate frequency band is done in the CDP nodes. For each antenna and each frequency band, correction coefficients are obtained from telescope calibration subsystem via the control subsystem. WVR channel values are published from each antenna’s WVR receiver. The CDP utilizes these two streams and applies the atmospheric phase correction to the spectra.

The path length correction coefficients are published before an observation starts, when a scan starts and at periodic intervals during which they are valid. The WVR channel values are published about every ½ second which is the frequency of these corrections.

The correction is of the form (note that these are preliminary forms which will be revised in future versions of this document):
\[ \Phi_n = \phi_i \{ C_{n0} V_{n0} + C_{n1} V_{n1} + C_{n2} V_{n2} + C_{n3} V_{n3} + C_{n4} V_{n4} \} \]  

\[ \Phi_m = \phi_i \{ C_{m0} V_{m0} + C_{m1} V_{m1} + C_{m2} V_{m2} + C_{m3} V_{m3} + C_{m4} V_{m4} \} \]  

\[ \Phi_i = \Phi_n \Phi_m^* \]  

Where:

- \( \Phi_i \) is the atmosphere-corrected phase for a specified antenna in a baseline n-m for the \( i^{th} \) channel
- \( \phi_i \) is the phase at the \( i^{th} \) channel for baseline n-m
- \( C_{0-4} \) are the atmospheric path length correction coefficients (converted to phase) for a specified antenna (n or m) supplied by the telescope calibration system
- \( V_{0-4} \) are the WVR values from a specified antenna’s (n or m) WVR receiver

Equations (3) and (4) provide phase corrections for antenna \( n \) and \( m \). Equation (5) shows the correction for antenna \( n \) multiplied by the complex conjugate of the correction for antenna \( m \) producing the final phase correction for each channel in the spectral window.

It is important to note that APC and non-APC data are integrated separately.

### 3.6.2.8 Integration Spectral Averaging

This step simply adds spectra for each dump into an integration accumulator with APC and non-APC data summed separately. The integration duration is an integer multiple of correlator dumps and once the integration duration is complete, the resulting spectral data are passed on to the DataWriter component (see section 3.6.3).

### 3.6.2.9 Channel Average

This class performs periodic channel averaging on the spectral data sets. The term ‘channel average’ is also known as ‘channel 0’. A complete discussion of channel averaging is provided in section “Channel Average” of [18].

The channel average for the cross correlation spectra is the vector average of the complex visibilities across the spectrum. For a continuum source the channel average has the same value as the channels, but with reduced noise. To remove any phase slope across the band, residual delays are first removed before the channel average is calculated.

The channel averaging configuration defines a rate at which the channel averaging is performed and a set of spectral channels over which to calculate the average (up to 10 separate ranges can be defined). The spectral channels are chosen to avoid the edges of the bandpass and sometimes selected to match a spectral line (maser). The integration duration ranges between 0.5 – 1.0 seconds such that there are an integral number of channel average integrations within a spectral integration duration. These data are accumulated separately for APC-corrected and –uncorrected results and are also accumulated separately from spectral data.

### 3.6.3 Data Publishing

Finally, the spectral and channel average data are published separately. The DataWriter assembles the baselines of an integration for a specific subarray. The XML header and binary data are joined
together in the appropriate VOTable. The spectral data blocks and channel average data are then published using the `DataPublisher` class.

A multicasting mechanism is proposed where the CDP publishes to the FDS which in turn multicasts the data to three subscribers: (1) the Archive, (2) the telescope calibration subsystem and (3) the quick look pipeline.

```
Data Publisher::DataWriter
- subArray
- startTime
- duration
- polnProduct
- binNumber
+ assembleIntegrations()
+ createHeader()
+ getAtmoCorrSpectra()

Data Publisher::Publisher
- connectionStatus : bool
+ connectToArchive()
+ sendSpectraToArchive()
+ monitorConnection()
```

Figure 16 Data Publisher Package Details

Some buffering of spectral results occur here. If, for example, the network connection between the CDP and the FDS malfunctions, spectral results can be buffered here. As there are no hard disks in any of the CDP computers, only a limited amount of buffering at the average data rates can occur. If one takes the average data rate of 4 MB/sec, then approximately 4 minutes of data can be buffered in 1 GB of RAM. Thus the master node could be loaded up with RAM to provide data buffering of 15 GB or one hour. This is a technical issue which needs to be specified.

The autocorrelation spectra are always taken and archived. These passbands may be used in the calibration process and are certainly important for debugging the front ends, filters, digitizers and data transmission system. They are a free spectrum analyzer.

As discussed earlier, a VOTable format for spectral and channel average data will be used. The VOTable contains XML headers with binary data. See section 7 of [7] for data formats. The binary portion of the data should be stored in a fashion that avoids rotation of the data by pipeline processing computers to preserve compatibility with AIPS++ MeasurementSetV2 format for an efficient quick-look operation. The current plan to use AIPS++ measurement sets expects the data to be ordered by polarization then channel. Also the performance implications of this rotation on the CDP compute nodes are not yet known and require more investigation.

3.6.3.1 Scaling Factors

The final spectral results must be scaled for efficient storage in the archive as specified in SSR 2.3-R5. A scaling factor is computed based on bandwidth and integration duration which is used to determine if the spectral results can fit into a 16- or 32-bit signed integer. This scale factor accompanies the data results and is specific to each integration, although it may be the same for many integrations. Scaling factors for auto and cross correlations will differ as the dynamic range for autocorrelations is generally larger. A detailed procedure for determining scale factors can be found in ‘Data Scaling’ of [18]. Note that channel average data is not scaled as it is a single complex value.
3.7 Physical Architecture

*Figure 17 shows a block diagram of the correlator computer hardware and interconnections for the full 64-antenna correlator.*
ALMA Baseline Correlator Computer Systems

- VME Correlator Control Computer (CCC) PowerPC
- CAN Busses (8)
- ~32 CAN nodes per Quadrant
- 48 ms TE

Master Node
- TCP/IP to Archive
- Real-time Linux
- Dual CPU
- 1+ GB RAM

Internal CCC-CDP CORBA

Data Processing Computer (CDP)
Each compute node:
- Dual CPU Athlon 2.4+ GHz
- 66 MHz PCI
- 25 MHz 32-bit data port
- Gigabit Ethernet
- Real-time Linux
- Diskless 512 MB RAM

Note: Network routers TBD

- 48 ms TE bus
- Private CDP network
- CCC/Corr. CAN bus
- ALMA computer network

August 25, 2003
3.7.1 Correlator Control Computer
The correlator control computer (CCC) is a PowerPC MVME2700 which utilizes a VME-64 backplane and runs the VxWorks 2.2 O/S. At this time (2003 July), there is discussion of changing this system to real-time Linux using RTAI [21] and an Intel Pentium-based CPU.

The CCC connects to the correlator hardware via 8 CAN busses using Tews TPMC816 CAN and Tews TIP916 CAN interface cards (1 CAN bus for the 2-antenna prototype with 1 TPM816). The CCC accesses the array-wide timing events via a Dynamic Engineering Parallel I/O card which has an Industrial I/O Pack form factor. The array-wide timing event connects directly to the correlator hardware which distributes it to the CCC and CDP. The CCC interfaces to the external computer systems via Ethernet.

3.7.2 Correlator Data Processor
The correlator data processor computer (CDP) is a cluster of dual Athlon CPU rack mounted PCs. The computers connected to the correlator hardware via 32-bit data ports are ‘compute nodes’ and connect to a ‘master node’ which acts as a bridge between the internal cluster network and the external telescope network.

The correlator hardware distributes the array-wide TEs to the CDP compute nodes via the PC’s standard parallel port. The TE goes to the CDP master node via an RS-485 interface.

Each compute node essentially performs the same function of extracting raw lags from the correlator hardware and converting them to spectral data blocks. There is no sharing of lags among compute nodes and all the lags for a given baseline end up in one compute node. An exception to this is when the quantization correction is made – the channel 0 lags of the auto-correlation are necessary to compute the correction. This is a single value should not impose a huge burden on inter-node communication.

3.7.3 Network Infrastructure
There are 3 networks associated with the correlator computers:

- **CAN** – a deterministic serial protocol used to control and monitor the correlator hardware
- **Gigabit Ethernet** – used within the CDP cluster for communication between the master and slave nodes.
- **Gigabit Ethernet** – used to interface the CCC and CDP to external computer systems including the ACC, Archive, telescope calibration, quick-look pipeline and antenna computers. It is important to note that Ethernet is a non-deterministic interconnect and that transmissions to real-time systems must be sent with far enough in advance to overcome any latency introduced by Ethernet. Currently this is assumed to be about 1 second.

The present plan has a 10Gbit fiber link from the AOS to the OSF to support correlator spectral data and a separate 10Gbit fiber link for monitor data and remote disk access.
3.7.4 Correlator Hardware

While complete descriptions of the correlator hardware can be found at [22], [16], and [23], a brief overview is provided here.

3.7.4.1 Basebands

Each quadrant is connected to a “baseband pair”. Each cable of the pair carries digitized samples at 4 giga-samples/sec of 3-bits for a given intermediate frequency in ‘V’ and ‘H’ polarizations and 2 GHz bandwidth. Depending on the correlator configuration, 1, 2 or 4 of the polarization products are used in calculating the cross-correlation function. The baseband index discussed in the correlator ICD identifies which baseband (and quadrant) is configured. Also the baseband configuration determines which polarization products are calculated.

A digital finite impulse response (FIR) filter provides a bandpass in powers of 2 from 2 GHz (essentially no filter) to 62.5 MHz (selectable by the BBC) and decimates one bit of data feeding the correlator chips 2-bit 4-level samples. Each correlator chip handles 4 x 4 matrix of antenna inputs (CAIs) with a “correlator card” holding 64 chips (for 32 x 32 CAIs) and a “correlator plane” holding 4 correlator cards (for 64 x 64 CAIs). There are 32 planes to a quadrant and the entire correlator consists of 4 quadrants.

Auto- and cross-correlation products are accumulated in the LTA which then dumps the raw lags via the DPI to the CDP for processing. The correlator hardware is synchronized with other array devices via the timing event bus.

3.7.4.2 Correlator Chip Accumulation

Correlator chip accumulation duration is measured in 1 ms for auto-correlations or 16 ms units for cross-correlations. This is specified as the correlator accumulation mode (CAM). Although 1 ms accumulations can be programmed, they are dumped at most every 16 ms, with 16 1-ms accumulations.

3.7.4.3 Bin switching

The LTA has up to 4 separate ‘bins’ or memory accumulation registers into which separate accumulations can be added. The sequence of switching among bins can be programmed on 16 ms boundaries. Bin switching can be utilized for phase or other types of ‘antenna’ switching or it can be used for double sideband 90º phase switching. This is in addition to separate binning provided by the CDP.

3.8 Dynamic Model

The operational flow of commands and data to and from CCC and CDP has been previously discussed. Here we attempt to evaluate the real-time constraints on these systems.

3.8.1 CCC Timing Discussion

The main timing constraint, i.e., hard deadline, for the CCC is the 48 ms TE. Most CCC time critical processes are synchronized to the TE:

- Start observing command. This command instructs the correlator hardware to begin correlating with a given configuration on a specific TE.
- Distributing the geometric delay to the station cards at a specific TE. Assuming all 64 antennas require delay updates, then ~10% of the CAN bandwidth is required – the
CAN bus can transmit 2.4KB/TE and 256 bytes for delay values. While the CAN bandwidth is not a problem, overhead in the CCC may be significant.

- Besides distributing the geometric delays to the correlator hardware, the CCC must not miss geometric delay events published on the delay notification channel. This can be avoided with the model server transmitting delay values in blocks in advance. Although it is the intention to provide shorter setup times, lead times of tens of seconds would ensure that network latency problems would be avoided. Prototype testing will allow better definitions of these lead values.

- Array time package cannot miss a tick. A short, period task is triggered each 48ms with a watchdog timer to expire if the tick is missed. The watchdog timer can execute recovery routines to re-establish array time.

- Lead time of 1.5 seconds for configuration of the correlator hardware. This restriction is linked to a fast switching interval requirement of 1.5 seconds (see sections 3.1.6.3.1.1 – 3.1.6.3.1.1 of [24]). Prototyping will allow us to determine if 1.5 seconds is sufficient overhead for the CCC. This is shown schematically in Figure 18.

```
Figure 18 Observation Configuration Timing Diagram
```

- Configuration for the CDP must be transmitted from the CCC with enough lead time to configure the CDP by a given TE. This is discussed in further detail in section 3.8.3.

In light of this timing information, Table 2 describes the beginnings of a rate monotonic analysis (RMA) done for the CCC. A simple procedure of RMA as described in [25] is followed where

\[
\text{Utilization} = \frac{\text{execution time}}{\text{period}} \times 100\%, \quad \text{i.e., the percentage of CPU time taken by a task.}
\]

As long as the cumulative utilization does not exceed 100%, all tasks are schedulable. Note that RTOS overhead is not accounted which is usually a few percent of total CPU time. Each task is listed in decreasing priority order.

```
<table>
<thead>
<tr>
<th>Task</th>
<th>Purpose</th>
<th>Execution Time ($C_i$) ms</th>
<th>Period ($T_i$) ms</th>
<th>Utilization %</th>
<th>Cumulative Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Array Time</td>
<td>Process 48ms TE</td>
<td>0.5</td>
<td>48</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2. Start Observing</td>
<td>Send start observing cmd. to LTA &amp; CDP</td>
<td>10</td>
<td>1500</td>
<td>0.7</td>
<td>1.7</td>
</tr>
<tr>
<td>3. Send delay</td>
<td>Distribute delay to SCC</td>
<td>10</td>
<td>48</td>
<td>20.8</td>
<td>22.5</td>
</tr>
<tr>
<td>4. Receive delays</td>
<td>Receive delay values from event channel</td>
<td>500</td>
<td>10000</td>
<td>5.0</td>
<td>27.5</td>
</tr>
<tr>
<td>5. Configure Observation</td>
<td>Receive &amp; schedule obs. configuration</td>
<td>100</td>
<td>1500</td>
<td>6.7</td>
<td>34.2</td>
</tr>
<tr>
<td>6. Monitor</td>
<td>Monitor corr. hw. &amp;</td>
<td>10</td>
<td>10000</td>
<td>0.1</td>
<td>34.3</td>
</tr>
</tbody>
</table>
```
### Table 2 CCC Rate Monotonic Analysis

<table>
<thead>
<tr>
<th>Task</th>
<th>Purpose</th>
<th>Execution Time ($C_i$) ms</th>
<th>Period ($T_i$) ms</th>
<th>Utilization %</th>
<th>Cumulative Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All execution times are currently estimates. As development progress continues, actual execution times based on prototypes can be developed to better define this analysis.

### 3.8.2 CAN Bandwidth

CAN overhead for delay updates: 1 CAN msg per 2 antennas each TE, and assuming ~6000 CAN messages/second (transmit only)

\[
64 \text{ antennas } \times 1/48 \text{ms} \times 1000 \text{ms/sec} \times 1 \text{ Msg/2 antennas} = 667 \text{ msg/sec} \approx 11\% \text{ of CAN bandwidth}
\]

### 3.8.3 CDP Compute Node Timing

For the compute nodes in the CDP, the important deadlines are the receiving of raw lags from the 32-bit data port and processing the previous set of raw lags before the subsequent set arrives. Specific time-critical events include:

- Transfer raw lags from 32-bit data port to a memory buffer via DMA. Every 16ms the LTA can transmit 1024 sets of 256 4-byte lags (1 MB). 1 MB/16ms = 64MB/sec.
- Process 1024 sets of 256 raw lags into spectra each 16 ms. Note that this data set size is an example. There are many size lag sets which affect processing times linearly except for the FFT – $O(n\log_2(n))$. For now, I am ignoring larger data sets as I believe this impact will be minor. This processing includes these steps:
  - Non-DMA transfers and copies of lag sets involved with lag processing
  - Antenna blanking including overhead of real-time event channel
  - Double sideband binning and calculations
  - Lag normalization, windowing, quantization correction and FFT. Tests show that on a 1.2 GHz Athlon this takes approximately 14 msecs for 1024 512-point data sets. Assuming 2.4 GHz and a linear increase in processing, this value becomes ~7 ms.
  - Fine geometric delay adjustment
  - Channel Averaging on 500 ms period
  - Transmit spectra to master node for an integration on 500 ms period. Note that this is depends strongly on the integration duration which can often extend to 10 seconds
  - Array time cannot miss a tick. A short, period task is triggered each 48ms with a watchdog timer to expire if the tick is missed. The watchdog timer can execute recovery routines to re-establish array time. If a timing tick is missed, the CDP must log an error and blank its data until it can resynchronize itself to array time.
  - Start observing command on a specific TE
  - Configure data processing parameters for an observation with a lead time of 1.5 seconds
The 32-bit data port interface on the correlator actually transfers the 1 MB at an effective burst rate of 125MB/sec. The PC’s 32-bit data port card can buffer 128K of 4-byte words which is 1/4 of the total data for the 16ms period. The PC’s 32-bit data port card then DMA transfers from the 32-bit data port FIFO buffer to the PC’s main memory.

Figure 19 shows the timing details of the 32-bit data port transfer from the correlator hardware to the PC’s RAM and lag processing. Note that the DMA transfer is not sequential, but interspersed between the 32-bit data port transfers to the PC’s 32-bit data port FIFO. The steps are:

1. Each 16ms the correlator hardware dumps 1024 x 256 x 4 bytes (1 MB) to the 32-bit data port card in the compute node PC. This takes 8 ms since the transfer is effectively 125 MB/sec.

2. The PC’s 32-bit data port card has a 128K x 4-byte FIFO buffer to store intermediate results. When this buffer fills up, a DMA transfer occurs between the 32-bit data port FIFO buffer and the PC’s RAM. It takes 4ms for a 66 MHz PCI bus for the 1 MB of data.

3. 12 ms remain to process the lags of the previous dump. Note that this is all of the time that the CPU has to do all of its processing out of each 16ms block of time. 25% of CPU availability is occupied by DMA mastering of the memory bus transferring raw lags from the 32-bit data port to the PC’s RAM.

4. This is a pipeline process where processing occurs on a previous dump.

5. In Table 3, the DMA transfer (task #2) is shown as a CPU utilization. Although this is not completely correct, the CPU is effectively idle as no memory accesses can occur while the DMA controller is mastering the bus.

<table>
<thead>
<tr>
<th>Task</th>
<th>Purpose</th>
<th>Execution Time (Cᵢ) ms</th>
<th>Period (Tᵢ) ms</th>
<th>Utilization %</th>
<th>Cumulative Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Array Time</td>
<td>Process 48ms TE</td>
<td>0.5</td>
<td>48</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2. Transfer Lags</td>
<td>DMA transfer of raw lags from 32-bit data port to RAM</td>
<td>4</td>
<td>16</td>
<td>25.0</td>
<td>26.0</td>
</tr>
<tr>
<td>3. Process Lags</td>
<td>Lag processing</td>
<td>1</td>
<td>16</td>
<td>6.3</td>
<td>32.3</td>
</tr>
<tr>
<td>4. Antenna Blanking</td>
<td>Discard invalid lag sets</td>
<td>1</td>
<td>16</td>
<td>6.3</td>
<td>38.5</td>
</tr>
<tr>
<td>5. Spectral Processing</td>
<td>Convert lag sets to spectra</td>
<td>7</td>
<td>16</td>
<td>43.8</td>
<td>82.3</td>
</tr>
</tbody>
</table>
According to these rough estimates, all tasks are schedulable, albeit very tight. There are some issues that need addressing:

- What improvement can be gained by using dual CPUs? Dual CPUs come into play with the task # 3 – 6. Assuming a 2X gain in processing throughput, the total utilization of these 4 task drops to 28% resulting in a total cumulative utilization of 66%. It is TBD if the added complexity of dual CPUs justifies this increase in processing capability, most likely it is justified. Also a full 2 times improvement is optimistic as coherence issues between the two CPUs is not understood.

- The Transmit Spectra task assumes that 1 Gigabit Ethernet is used, i.e., ~50 MB/sec see [26] with ~50% CPU utilization. The percent utilization is determined as such:
  - Amount of data to send every second: 10 MB which represents 10 100 ms integrations
  - Time to send out 1 second of data: 10 MB/(50 MB/sec) = 200 ms.
  - From [26], using a CPU utilization of 50%, the total CPU time used is: 200 ms * 0.5 = 100 ms. Thus every 1000 ms the CPU is utilized for 100 ms – Utilization = 100 ms/1000 ms = 10%

- A backup plan could be to create a ‘network of compute nodes’ to handle the loads. In this scenario, one primary compute node would connect to the 32-bit data port and have 2 or more secondary compute nodes connected to it. The primary compute node distributes lags to the secondary compute nodes for processing. This solution lowers the computation load, but may increase bandwidth loads and definitely increases system complexity.

- General overhead questions which need to be evaluated
  - How much overhead exists in master-slave communication?
  - How much of this work runs in Linux kernel modules vs. user space?

### 3.8.4 CDP Master Node Timing

The main function of the master compute node is a router. It sends commands from external computers to the compute nodes and routes data to the archive.

The time-critical processes include:

![Table 3 CDP Compute Node Rate Monotonic Analysis](Image)
- Array time cannot miss a TE tick. A short, period task is triggered each 48ms with a watchdog timer to expire if the tick is missed. The watchdog timer can execute recovery routines to re-establish array time. This is the only hard, real-time task.
- Subscribe to the antenna blanking event channel and route blanking status data to the appropriate compute nodes.
- Subscribe to the geometric delay event channel and route delays to the appropriate compute nodes.
- Subscribe to the WVR data event channel and route WVR values to the appropriate compute nodes.
- Route configuration and control commands from the CCC to the appropriate compute nodes.
- Receive (from the compute nodes) and assemble all integrations for all baselines lines in a given subarray.
- Transmit the spectral data integrations for a subarray to the archive at 72 MB/sec

<table>
<thead>
<tr>
<th>Task</th>
<th>Purpose</th>
<th>Execution Time (C_i)</th>
<th>Period (T_i)</th>
<th>Utilization</th>
<th>Cumulative Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Array Time</td>
<td>Process 48ms TE</td>
<td>0.5</td>
<td>48</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2. Geometric Delays</td>
<td>Subscribe to event channel &amp; distribute delays</td>
<td>500</td>
<td>10000</td>
<td>5.0</td>
<td>6.0</td>
</tr>
<tr>
<td>3. Antenna Blanking</td>
<td>Subscribe to event channel &amp; distribute blanking info</td>
<td>2</td>
<td>48</td>
<td>4.2</td>
<td>10.2</td>
</tr>
<tr>
<td>4. WVR Values</td>
<td>Subscribe to event channel &amp; distribute WVR values</td>
<td>2</td>
<td>500</td>
<td>0.4</td>
<td>10.6</td>
</tr>
<tr>
<td>5. Route cmds</td>
<td>Route config. &amp; control cmds. from CCC</td>
<td>2</td>
<td>48</td>
<td>4.2</td>
<td>14.8</td>
</tr>
<tr>
<td>6. Spectral data collection</td>
<td>Collect spectral integrations from compute nodes</td>
<td>100</td>
<td>500</td>
<td>20.0</td>
<td>34.8</td>
</tr>
<tr>
<td>7. Transmit Spectra</td>
<td>Transmit spectral data blocks to FDS</td>
<td>250</td>
<td>500</td>
<td>50.0</td>
<td>84.8</td>
</tr>
<tr>
<td>8. Monitor</td>
<td>Monitor CDP master node</td>
<td>10</td>
<td>10000</td>
<td>0.1</td>
<td>84.9</td>
</tr>
<tr>
<td>9. Administer cluster</td>
<td>General cluster administration</td>
<td>1000</td>
<td>6.00E+05</td>
<td>0.2</td>
<td>85.0</td>
</tr>
</tbody>
</table>

Table 4 CDP Master Node Rate Monotonic Analysis

As most of the time is spent transferring data, these values are highly variable as so much depends on PC hardware and network bandwidth.

As a backup plan, there can be multiple master nodes in a hierarchical arrangement. A primary master node can handle configurations from the CCC and there can be multiple node bridges between the internal CDP network and the external network. This plan increases the software complexity, but may be necessary to handle the data loads. These issues will be investigated during development and testing phases.
3.9 System Reliability

Here we attempt to identify reliability issues with the correlator system computers and software. Currently there are no technical requirements specified nevertheless, this is an important issue which needs addressing.

3.9.1 Correlator Control Computer

In general if the CCC fails, then control and operation of the correlator is completely disabled. This single point of failure is critical and there is no plan to have redundant, parallel computers where one computer can automatically switch into operation if the other fails.

- Spares – Spare CPU and I/O boards are available for cold-swapping. Replacement of boards should be straightforward with modular I/O connectors facilitating the cable connections. Down-time due to board replacement should take approximately 1 hour.

- Software Failures – Failures due to software fall into 3 levels of severity:
  - Low – These are warnings for which software errors have been identified in the code and are programmatically handled. These warnings are logged and software execution continues uninterrupted.
  - Medium – These are software errors which allow for execution to continue, but if enough occur, then software failure can ensue. An example of this is dynamic memory consumption increases until there is no available memory for the application to successfully execute. These errors may or may not be logged.
  - Severe – These are software errors which cause one or more major software modules to cease execution. Examples of these are null or invalid pointer references, dynamic memory exhaustion, stack overflows, etc. Often these errors are not logged before execution halts and the fact that the system has failed may be difficult to determine.

Once software execution fails, the CCC must be rebooted which can take several minutes to come up. Subsequent re-configurations of the CCC and correlator hardware will take more time. We plan to have remotely controlled power strips for remote reboots.

3.9.2 Correlator Data Processor Computer

Due to the CDP hardware architecture if one computer fails, there may partial or complete failure of the correlator subsystem. If a CDP compute node fails, then other nodes should still continue to process results. If the CDP master node fails, then no spectral results will flow from the correlator subsystem leading to a single point of failure.

There are several areas of potential hardware failure with the CDP nodes.

- Cooling – The CDP nodes have high-performance AMD CPUs which run very hot. The computer enclosures will have many fans (with filters) and the CPUs also have a cooling fan. The CDP will share the temperature-controlled environment of the correlator hardware which should prevent overheating as long as all of the fans are functioning. These fans are the most susceptible components to failure and can be monitored in real time. CPU and enclosure temperatures can be tracked to assist in scheduling maintenance.

- Spares – There will be complete compute nodes which can be cold-swapped in to minimize replacement time. A replaced node will need to have its interface
connections reattached, its software reloaded and configurations reloaded. Down time due to computer replacement should take about an hour.

- Networking – Cabling and routers used in the CDP are hardware-replaceable items which should involve a down time of about an hour. Also the CDP master node can load up with RAM to buffer network latencies to the FDS.

- Software Failures – The same ideas regarding software failures and severity outlined for the CCC apply to the CDP. The only difference is the single point of failure issue with respect to a software failure in a compute node versus a failure in the master node.

4 System Interfaces

This section discusses all external interfaces provide by and used by the correlator subsystem.

4.1 Package – Interface Relationship

This section shows the relationship of interfaces, data streams and events defined in [7] to packages which have been defined in this document. As the following table shows, many interface functions are divided among many packages.

<table>
<thead>
<tr>
<th>Interface Item</th>
<th>Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMA.Correlator.ObservationControl</td>
<td>ACS CCC Interface</td>
</tr>
<tr>
<td></td>
<td>CCC Command Dispatcher</td>
</tr>
<tr>
<td></td>
<td>Correlator Hardware Control</td>
</tr>
<tr>
<td></td>
<td>Subarray Management</td>
</tr>
<tr>
<td></td>
<td>Correlator CAN Commands</td>
</tr>
<tr>
<td></td>
<td>CDP Interface</td>
</tr>
<tr>
<td></td>
<td>CDP Configuration</td>
</tr>
<tr>
<td>ALMA.Correlator.CCC_Monitor</td>
<td>CCC Monitor</td>
</tr>
<tr>
<td></td>
<td>ACS CCC Interface</td>
</tr>
<tr>
<td></td>
<td>Correlator CAN Commands</td>
</tr>
<tr>
<td>ALMA.Correlator.CDP_Monitor</td>
<td>CDP Monitor</td>
</tr>
<tr>
<td></td>
<td>ACS CDP Interface</td>
</tr>
<tr>
<td>ALMA.Correlator.ObservationQuery</td>
<td>CDP Monitor</td>
</tr>
<tr>
<td></td>
<td>CCC Maintenance</td>
</tr>
<tr>
<td>ALMA.Correlator.Maintenance</td>
<td>CCC Maintenance</td>
</tr>
<tr>
<td>ALMA.Correlator.ConfigurationValidator</td>
<td>Corr. Config Validation</td>
</tr>
<tr>
<td>ALMA.Correlator.ArrayTime</td>
<td>Array Time Interface (CCC &amp; CDP)</td>
</tr>
<tr>
<td>PublishIntegrationEvent</td>
<td>ACS CDP Interface</td>
</tr>
<tr>
<td></td>
<td>Data Publisher</td>
</tr>
<tr>
<td>WVRValueEvents</td>
<td>ACS CDP Interface</td>
</tr>
<tr>
<td></td>
<td>APC Correction</td>
</tr>
<tr>
<td>AntennaBlankingEvents</td>
<td>ACS CDP Interface</td>
</tr>
<tr>
<td></td>
<td>Lag Processing</td>
</tr>
<tr>
<td>GeometricDelayModelEvents</td>
<td>ACS CDP Interface</td>
</tr>
<tr>
<td></td>
<td>Geometric Delay</td>
</tr>
<tr>
<td></td>
<td>Spectral Processing</td>
</tr>
<tr>
<td></td>
<td>ACS CCC Interface</td>
</tr>
<tr>
<td>Interface Item</td>
<td>Packages</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Correlator Data Stream</td>
<td>Data Port Interface</td>
</tr>
<tr>
<td></td>
<td>Lag Processing</td>
</tr>
<tr>
<td></td>
<td>Double sideband receivers</td>
</tr>
<tr>
<td></td>
<td>Data Processing</td>
</tr>
<tr>
<td></td>
<td>Data Publisher</td>
</tr>
<tr>
<td>Channel Average Stream</td>
<td>Lag Processing</td>
</tr>
<tr>
<td></td>
<td>Data Processing</td>
</tr>
<tr>
<td></td>
<td>Data Publisher</td>
</tr>
<tr>
<td>Monitor Stream</td>
<td>ACS CDP Interface</td>
</tr>
<tr>
<td></td>
<td>CDP Monitor</td>
</tr>
<tr>
<td></td>
<td>CCC Monitor</td>
</tr>
</tbody>
</table>

*Table 5 Interface-Package cross reference*

## Appendices

### 5.1 Equations

Equations used in this design are summarized here.

#### 5.1.1 $V_a$ Subtraction

For each lag value the following correction is performed to remove the correlator chip multiplication bias:

$$
lag(k) = \left( \frac{lag(k) - \text{dumpIntegrationCounts}}{lag(0)} \right)
$$

#### 5.1.2 Lag Normalization

See ‘Spectral Normalization’ [18] for details.

#### 5.1.3 Geometric Delays

The geometric delays are discussed in section 3.6.2.6.

#### 5.1.4 Atmospheric Phase Correction

See section 3.6.2.7.

#### 5.1.5 Digitization Correction

See [19] for details.

#### 5.1.6 Windowing Functions

Bartlett:
\[ w(k + 1) = \begin{cases} \frac{2(k)}{n - 1}, & 0 \leq k \leq \frac{n}{2} - 1 \\ \frac{2(n - k - 1)}{n - 1}, & \frac{n}{2} \leq k \leq n - 1 \end{cases} \]

Blackman:
\[ w(k + 1) = 0.42 - 0.5 \cos\left(2\pi \frac{k}{n - 1}\right) + 0.08 \cos\left(4\pi \frac{k}{n - 1}\right), k = 0, \ldots, n - 1 \]

Blackman-Harris:
\[ w(k + 1) = a_2 - a_1 \cos\left(2\pi \frac{k}{n - 1}\right) + a_0 \cos\left(4\pi \frac{k}{n - 1}\right) - a_0 \cos\left(6\pi \frac{k}{n - 1}\right), 0 \leq k \leq (n - 1) \]

Hamming:
\[ w(k + 1) = 0.54 - 0.46 \cos\left(2\pi \frac{k}{n - 1}\right), k = 0, \ldots, n - 1 \]

Hann (Hanning):
\[ w(k + 1) = 0.5 \left(1 - \cos\left(2\pi \frac{k}{n - 1}\right)\right), k = 0, \ldots, n - 1 \]

Welch:
\[ w(k) = 1 - \left( \frac{k - \frac{n}{2}}{\frac{n}{2}} \right)^2, k = 0, \ldots, n \]

5.1.7 Discrete Fourier Transforms
See [20] for details.

5.1.8 Channel Averaging
See ‘Channel Average’ in [18].

5.1.9 Double Sideband Separation
See section 3.5.15.
5.2 Correlator Hardware CAN Commands

Here we list the currently available commands which the CCC can use. The source for these commands comes from sections 8 – 11 in [10].

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Identification Request</td>
<td>Request node ID</td>
</tr>
<tr>
<td>Block Transfer to Absolute Memory Address</td>
<td>Send data to absolute target memory address</td>
</tr>
<tr>
<td>Block Transfer of Defined Data Structures</td>
<td>Send data structure</td>
</tr>
<tr>
<td>Download Configuration Parameters</td>
<td>Send correlator configuration data</td>
</tr>
<tr>
<td>Download Phased array parameters</td>
<td>Configure phased array mode</td>
</tr>
<tr>
<td>Generate New State</td>
<td>Command embedded CPU to generate new FPGA control words for a correlator configuration</td>
</tr>
<tr>
<td>Apply New State</td>
<td>Apply new correlator configuration at the next TE</td>
</tr>
<tr>
<td>Set 48ms Tick Count</td>
<td>Set a specific TE to a given value</td>
</tr>
<tr>
<td>Bootstrap Operations</td>
<td>Download new microprocessor code &amp; reboot</td>
</tr>
<tr>
<td>Erase Flash Sectors</td>
<td>Erase microprocessor flash memory</td>
</tr>
<tr>
<td>Program Data Flash</td>
<td>Program microprocessor data flash sector(s)</td>
</tr>
<tr>
<td>Program Code Flash</td>
<td>Program microprocessor code flash sector(s)</td>
</tr>
<tr>
<td>Setup Correlator Card Tests</td>
<td>Run microprocessor tests</td>
</tr>
<tr>
<td>Block Read back from Absolute Memory</td>
<td>Read data from absolute target memory address</td>
</tr>
<tr>
<td>Read back of Defined Data Structures</td>
<td>Read data structure</td>
</tr>
<tr>
<td>Read CAN Debug or History Queue</td>
<td>Read CAN debug data or CAN history data</td>
</tr>
<tr>
<td>Read Status Blocks</td>
<td>Read TBD status blocks</td>
</tr>
<tr>
<td>Read back of Correlator Card Lags</td>
<td>Read raw lags</td>
</tr>
<tr>
<td>Set/Get 48ms Tick Count</td>
<td>M&amp;C set or get 48 ms tick count</td>
</tr>
<tr>
<td>Set/Get My Accumulation Planes</td>
<td>M&amp;C Specifies which 8 correlator planes for an LTA microprocessor</td>
</tr>
<tr>
<td>Set/Get My Control Planes</td>
<td>M&amp;C Specifies a pair of correlator planes for an LTA microprocessor</td>
</tr>
<tr>
<td>Get Data Flash Status</td>
<td>Get status information regarding flash memory erase &amp; program operations</td>
</tr>
<tr>
<td>Get Bootstrap Status</td>
<td>Get status of bootstrap operations</td>
</tr>
<tr>
<td>Get Checksum Status</td>
<td>Get microprocessor code checksum status</td>
</tr>
<tr>
<td>Get Code Checksum</td>
<td>Get microprocessor code checksum</td>
</tr>
<tr>
<td>Get Code Flash Status</td>
<td>Get microprocessor code flash status</td>
</tr>
<tr>
<td>Get Data Checksums</td>
<td>Get microprocessor data checksums</td>
</tr>
<tr>
<td>Set/Get CAN History Queue Number of Entries</td>
<td>Set/Get size of CAN history queue</td>
</tr>
<tr>
<td>Command CAN Debug Capture</td>
<td>Begin capture of CAN commands</td>
</tr>
<tr>
<td>Set/Get Test Accumulation Number of Ticks</td>
<td>Set/Get number of 16 ms ticks used for test accumulations</td>
</tr>
<tr>
<td>Get List of Cards Present</td>
<td>Get list of correlator and interface cards present</td>
</tr>
</tbody>
</table>

Table 6 - LTA CAN Commands

The following table shows CAN commands which are specific to the station control cards.

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Identification Request</td>
<td>Request node ID</td>
</tr>
<tr>
<td>Block Transfer to Absolute Memory Address</td>
<td>Send data to absolute target memory address</td>
</tr>
<tr>
<td>Set 48ms Tick Count</td>
<td>Set a specific TE to a given value</td>
</tr>
<tr>
<td>Bootstrap Operations</td>
<td>Download new microprocessor code &amp; reboot</td>
</tr>
<tr>
<td>Erase Flash Sectors</td>
<td>Erase microprocessor flash memory</td>
</tr>
<tr>
<td>Program Data Flash</td>
<td>Program microprocessor data flash sector(s)</td>
</tr>
<tr>
<td>Command Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Program Code Flash</td>
<td>Program microprocessor code flash sector(s)</td>
</tr>
<tr>
<td>Download BBC</td>
<td>Set BBC for 4 CAIs</td>
</tr>
<tr>
<td>Apply BBC</td>
<td>Apply the downloaded BBC at the next TE</td>
</tr>
<tr>
<td>Download FIR Tap Weights</td>
<td>Set the FIR tap weights</td>
</tr>
<tr>
<td>Download Delay Model</td>
<td>Set quantized delay model parameters</td>
</tr>
<tr>
<td>Block Read back from Absolute Memory</td>
<td>Read data from absolute target memory address</td>
</tr>
<tr>
<td>Read back of Defined Data Structures</td>
<td>Read data structure</td>
</tr>
<tr>
<td>Read CAN Debug or History Queue</td>
<td>Read CAN debug data or CAN history data</td>
</tr>
<tr>
<td>Read Status Blocks</td>
<td>Read TBD status blocks</td>
</tr>
<tr>
<td>Set/Get 48ms Tick Count</td>
<td>M&amp;C set or get 48 ms tick count</td>
</tr>
<tr>
<td>Get Data Flash Status</td>
<td>Get status information regarding flash memory erase &amp; program operations</td>
</tr>
<tr>
<td>Get Bootstrap Status</td>
<td>Get status of bootstrap operations</td>
</tr>
<tr>
<td>Get Checksum Status</td>
<td>Get microprocessor code checksum status</td>
</tr>
<tr>
<td>Get Code Checksum</td>
<td>Get microprocessor code checksum</td>
</tr>
<tr>
<td>Get Code Flash Status</td>
<td>Get microprocessor code flash status</td>
</tr>
<tr>
<td>Get Data Checksums</td>
<td>Get microprocessor data checksums</td>
</tr>
<tr>
<td>Set/Get CAN History Queue Number of Entries</td>
<td>Set/Get size of CAN history queue</td>
</tr>
<tr>
<td>Command CAN Debug Capture</td>
<td>Begin capture of CAN commands</td>
</tr>
<tr>
<td>Get CPLD2 Status</td>
<td>Return status of target cards in bin</td>
</tr>
<tr>
<td>Get FPGA Init Status</td>
<td>Get status of FPGAs and initialization</td>
</tr>
<tr>
<td>Get FPGA Done Status</td>
<td>Get status of FPGAs after download</td>
</tr>
<tr>
<td>Get FPGA Flash Storage Status</td>
<td>Mask indicating which FPGA personalities were found to be stored with a proper SYNC word in flash memory. We assume a valid personality is present if the SYNC word is correct.</td>
</tr>
</tbody>
</table>

Table 7 - SCC CAN Commands

6 References


