ALMA Test Correlator Control Computer Software Design

Jim Pisano

2001-03-27

Contents

List of Figures

List of Tables

1 Introduction

This document is divided into three sections: requirements for the test correlator software, use cases based on those requirements and the test correlator control computer software design. This design is considered a “component design” which describes a specific component in a larger software system, i.e., the ALMA Test Interferometer Control Software (TICS) (see [1]). The purpose of this component design is to describe the details of this software from an object-oriented perspective such that it satisfies the use cases and requirements. Its initial installation is as a stand-alone application to be run in an engineering lab environment at the AOC and later in Tucson. The interface components of this design don’t fully match TICS as the latter was specified much later than this design. Future revisions will modify this design for a seamless interface with TICS.

A standard glossary of terms and acronyms used throughout this document can be found at [2]. Some of the common acronyms used:

- **ACC** - “Array Control Computer” The computer located at the central control area and responsible for controlling all array functions. It will be a Linux workstation.
- **CCC** - “Correlator Control Computer” A real-time computer located at the central control area and responsible for the control and monitor of the test correlator.
- **CDP** – “Correlator Data Processor” The functional part of the CCC that is responsible for extracting and processing lags and transferring the spectral results to the Data Collector. The Data Collector is a computer that accepts well-defined spectra, applies further processing (flagging bad data, etc.) and writes these processed results to a distribution format.
- **CHW** – “Test Correlator Hardware” The test correlator.

The terms observation, integration, dumps and short-term integration are used throughout this document and warrant clarification. The first three terms are defined in [2].

- **Observation** – A set of *integrations* while the antennas complete an elemental pattern across the source, and possibly while frequency switching, nutator switching, etc
- **Integration** – A set of *dumps*, all identical in configuration (except for the antenna motion and some others), that is accumulated and forms the basic recorded unit
- **Dump** – The acquisition of data from the correlator corresponding to the smallest interval of time for which a set of correlated data can be accumulated and output from the correlator.
- **Short-term integration** – The integration that occurs at the correlator chip level during one correlator tick of 1.31072 ms. Multiple short-term integrations comprise a *dump*.

The test correlator will act as an interim device to assist in the evaluation of the prototype ALMA antennas and the test interferometer while the prototype ALMA correlator is being developed. The scope of this software design reflects that purpose.

It is important to realize that the ALMA control software will not be fully available when this software is first needed. In order to utilize this software, a simple “stand-alone” software application will be developed which can be used by engineers for lab tests involving the correlator.

1.1 Test Correlator Hardware Description

The test correlator is 1 quadrant of the GBT spectrometer. Its capabilities and interface requirements are listed in [3]. Some of its characteristics include:

- Supports 2 antennas
- Supports 2 basebands per antenna
- Maximum sampling rate per baseband is 1.6 GHz
- Correlators are 2-bit, 3 level
- Maximum delay of 10 μsecs. at 800 MHz
- Two bandwidths – 800 MHz and 100 MHz
- 4 polarization products available - RR, LL, RL and LR

A photograph of the test correlator appears in Figure 20.

2 ALMA Test Correlator Software Requirements

In order to define the software requirements for the test correlator control software, requirements for the test correlator are also listed here.

The ALMA test correlator contains the following cards:

- **1 System Monitor**

- **1 Correlator Control**
- **2 Long Term Accumulators**
- **1 Data Tap (optional)**
- **1 Sampler Distributor**
- **4 Correlator Cards**
- **4 Memory Cards**
- **1 VME Interface**

The functions required by the ALMA test correlator which define the requirements for the test correlator control software are:

- **Initialization** – when the correlator cold boots, the control computer needs to perform some rudimentary tests and initializations
- **Configuration** – before observations can begin, the correlator must define correlator integration parameters
- **Observing** – this includes commands to start observing, process lags, deliver spectra to the Data Collector and stop observing
- **Monitoring** – the correlator hardware monitors its power supplies and temperatures for out-of-range values.
- **Error Handling** – if the correlator hardware has problems, then warnings need to be sent to operators, information status needs to be obtained, results of diagnostic tests, etc.
- **Communication** – the correlator computer communicates with the ACC. Communication links need to be established to the ACC and the Data Collector.
- **Timing** – minimum dump times and synchronization with other array devices must be specified.
- **Engineering Testing** – the test correlator will be used in a laboratory environment that allows for the testing of the correlator hardware in a stand-alone fashion.
- **Sampler threshold check** – sampler thresholds can be determined using the correlator.

2.1 Initialization

When the correlator boots up, there is a sequence of commands that the correlator computer must send to initialize the correlator hardware. These initialization sequences shall configure the 3 cards:

- Correlator control card
- Long Term Accumulator card
- System monitor card

This initialization process also verifies that the cards' microprocessors are communicating with the correlator computer.

2.2 Configuration

In order to start an observation, a few of items of information must be received by the ACC.

The information required for correlator configuration shall include:

2.1.1 Correlator system mode which defines the following (see Table 2 for details):

2.1.1.1 Bandwidth information either 800 MHz or 100 MHz

2.1.1.2 The number of spectral channels - 512, 1024, 8192 or 16,384.

2.1.1.3 Auto-correlation or cross-correlation mode.

2.1.1.4 Polarization products are fixed based on the correlator system mode. See Table 2 for details.

2.1.2 Dump times

It takes ~46 ms to read data out of the correlator regardless of the dump time due to the transfer rate of the LTA to the VME computer of ~ 5.6 MB/sec and a quantity of data of ~256 KB per dump. Note that this does not allow for any processing of raw lags, e.g., summing of raw lags, FFTs, etc. When these data processing functions are added to the read-out duration, the minimum dump time shall be ~70 ms.

2.1.3 Synchronized short-term integrations
The correlator shall be capable of starting a set of short-term integrations on the leading edge of a 48 ms timing event. There are 36.621… 1.31072 ms correlator ticks in a 48 ms interval, so the correlator performs 36 short-term integrations and blanks correlation for the remaining fraction.

2.1.4 The maximum dump time shall not exceed ~85 seconds in single dish mode which is fixed by the test correlator hardware accumulators. For interferometry observations, the maximum dump time shall not exceed ~1 second due to the coarse delay resolution of the test correlator hardware (D'Addario, private comm.,2000).

2.1.5 The integration duration shall be a multiple of correlator dumps. The minimum integration duration shall be the minimum dump duration. The maximum integration duration shall extend to many hours.

2.1.6 The number of integrations for an observation shall be specified.

2.1.7 One, two or four separate memory bins can be utilized for an integration with bin switching times synchronized to 48 ms array-wide timing events.

2.1.8 Data Processing Parameters

The correlator computer shall allow the following data processing options to occur on the raw lags from the test correlator:

- FFT
- van Vleck quantization correction
- Hanning windowing function
- Spectral averaging where a variable number of adjacent spectral channels are averaged together.
- Spectral decimation: only every n-th channels is retained.
- Define a subset of spectral channels for delivery from the correlator computer to the ACC.

2.3 Observing

In order for an observation to occur, the correlator computer shall:

2.3.1 For interferometry observations, apply the coarse delay to the correlator hardware before the start of an integration based on the delay model whose coefficients are received from the ACC.

2.3.2 Issue a start observing command to the correlator hardware.

2.3.3 Collect raw lag results from the correlator hardware with minimal latency after each dump via an external memory FIFO on the VME bus.

2.3.4 Process raw lags results for each correlator dump using specified data processing options.

2.3.5 Apply a fine delay correction after the FFT (but before averaging or decimation) for interferometry observations for each spectral data set.

2.3.6 Allow for multiple correlator dumps per integration. This is defined as adding together the spectral results for the current correlator dump with other those results from previous dumps in the correlator computer data processing unit before transmitting them to the Data Collector.

2.3.7 Time stamp each integration using array time that specifies the integration start.

2.3.8 Transmit each spectral data set to the Data Collector.

2.3.9 Issue a stop observing command to the correlator after the preset number of integrations has been reached or in response to an "stop observation" command from the ACC.

2.3.10 The correlator data processing computer system should support the maximum data rate of ~160,000 complex spectral visibilities/second (~640 KB/sec) and a processing rate of ~4 MFLOPS. See [4] for details.

2.4 Observing Modes

The test correlator shall support the following observational modes:

2.4.1 Interferometry mode using 2 antennas

2.4.2 One or two antennas operating in single dish mode. Note that due to limitations of the test correlator, both antennas must have identical configurations and start integrations simultaneously.

2.4.3 180° phase switching shall be done externally before input to the correlator (see [5]).

2.4.4 90° phase switching support is a desired feature (D'Addario, 2000, private comm.)

2.4.5 Testing of side band suppression using the LO's and separate baseband channels for each side band so that both side bands can be observed simultaneously (D'Addario, 2000, private comm.)

2.4.6 Beam switching modes shall be supported at a rate of 48 ms.

2.4.7 Frequency switching is a desired feature at a rate of 48 ms.

2.4.8 Correlator bin switching synchronized to the 48 ms array-wide timing events shall be used for synchronized switching with 2 or 4 bins shall be available.

2.4.9 On-the-fly mode shall be supported as it is for the MAC spectrometer on the 12m [6]. Minimum dump times shall be limited to ~70 ms.

2.4.10 The start of short-term integrations at the correlator chip level can optionally be synchronized to the 48 ms array-wide timing events in order to test system synchronization with other hardware components in the test interferometer. (D'Addario, 2000, private comm.)

2.5 Monitoring

2.5.1 The correlator computer shall monitor the following voltages and temperatures:

Index	Quantity	Units	Resolution
1	+ 5 VDC	Volts	5 mV
2	- 5 VDC	Volts	5 mV
3	-2 VDC	Volts	5 mV
4	+24 VDC	Volts	15 mV
5	+15 VDC	Volts	10 mV
6	-15 VDC	Volts	10 mV
7	Temperature in degrees C	Degrees Celsius	0.1 degree

Table 1 – Test Correlator Monitor Points

2.5.2 The polling rates for these monitor values shall range from 5 – 30 seconds and shall not interfere with other control and data collection and processing functions.

2.5.3 These monitor data shall be made available to the ACC.

2.5.4 The correlator computer shall know the actual operational state of the correlator hardware at all times and make this information available to the ACC.

2.6 Error Handling

2.6.1 The correlator computer shall know the error condition(s) of the correlator hardware at all times.

2.6.2 The correlator computer shall log any run-time errors and report them to the ACC when requested.

2.7 Communication

2.7.1 Communication to the ACC and the Data Collector will be via TCP/IP.

2.7.2 Transmitted spectral or lag data shall be single precision IEEE floating point values.

2.7.3 The correlator computer shall track the status of the connection with the ACC and the Data Collector and set error conditions accordingly.

2.7.4 All transmitted data shall be sent in network byte order with no padding.

2.7.5 Except for reading lag results, the correlator control software shall communicate to the CHW via serial communication using the following parameters: 57,600 baud, 8 data bits & 2 stop bits.

2.7.6 The data transfer rate for lag results from the LTA to the correlator computer shall be about 6 MB/sec.

2.8 Timing

2.8.1 The correlator system tick is an exact value of 1.31072000… ms so all dump times shall be a multiple of this value.

2.8.2 The correlator computer shall have a physical interface to the 20.833… Hz array time (48 ms Timing Event) as defined in [7] for purposes of tracking array time and synchronization with the array.

2.8.3 The test correlator hardware shall utilize the 48 ms timing events as defined in [7] for synchronizing integration starts.

2.8.4 The processing of lag results shall occur in a double-buffered fashion, i.e., while a CHW is integrating on the current dump, the lag results from the previous dump are being processed.

2.8.5 Dump times shall be controlled such that there is sufficient time to process the lags without overrunning one set of lags with a new set. This duration limit shall depend on the correlator system mode.

2.8.6 Time critical commands shall be received from the ACC at least 72 ms before the specific Timing Event to which the command applies.

2.9 Engineering / Test Interface

2.9.1 A simple computer interface on a separate host computer shall be provided for testing the correlator in “stand-alone” mode. This interface shall have the following functionality:

2.9.2 Act as a simple ACC to issue control commands to and view responses from the correlator computer.

2.9.3 Act as a simple Data Collector as to accept spectral results from the correlator computer for archiving and viewing in a near real time fashion.

2.9.4 Allow the user to execute simple sequences of commands

2.9.5 Allow the user to view raw lag results and other information useful in debugging the test correlator hardware.

2.9.6 A graphical interface for configuration, execution of tests and display of test results.

2.10 Calibration

2.10.1 A procedure shall exist to determine the sampling threshold value for each test correlator sampler.

2.10.2 Water vapor radiometry will not be available for the baseline plan for the test interferometer so no atmospheric correction is required.

3 Correlator Use Cases

These use cases apply to the test correlator when used to evaluate the prototype antennas in single dish and interferometry modes. For engineering tests, there will be a host computer acting as a simple ACC and Data Collector.

The layout of use cases follows that used by ESO. See [8] for more information regarding this format. States identified in these use cases appear in the state diagram in Figure 15.

3.1 Use Case: Test Correlator Initialization

This use case describes the test correlator system initialization process. The correlator subsystem is comprised of two hardware components that require initialization:

- Correlator Hardware (CHW) which must be powered on before initialization of the correlator computer.
- Correlator Control Computer (CCC)
  Initialization of the correlator computer can be initiated in three ways:
  1) by turning on the power,
  2) by pressing the correlator computer's reset button on the correlator computer front panel,
  3) by a reset command sent from the ACC.

States

- **DISABLED** – Correlator subsystems have been turned on or reset via a reset command or a physical reset switch and are in a state which requires an initialization sequence to be performed.
- **IDLE** – Correlator subsystems are functioning correctly and the CCC is ready to accept commands from ACC while the CHW is ready to accept commands from the CCC.
- **FAULTED** – An error has occurred initializing a correlator subsystem thus causing that subsystem to be not usable. Might require operator intervention.

Actors

Primary:

- ACC
- CCC
- Data Collector
- Human operator who can power up the CHW and/or the CCC.

Secondary:

- Correlator Hardware (CHW)

Priority: Critical

Performance: N/A

Frequency: At startup and irregular intervals otherwise, such as after a reset

Preconditions: State(CCC) = DISABLED | DON'T_CARE. Power to the CHW & CCC must be on.

Basic Course:

- Subflow: Initialize Correlator Computer
- Subflow: Initialize Correlator Hardware

Post conditions: State(CCC), State(CHW) = IDLE

Subflow: Initialize Correlator Computer

- Instantiate singleton objects

- Spawn tasks
- Wait for connection from ACC client
- Verify connection to ACC client
  Exception Course: CCC initialization error
- Wait for connection from Data Collector client
- Verify connection to Data Collector client
  Exception Course: CCC initialization error

Postcondition: State(CCC) = IDLE

Subflow: Initialize Correlator Hardware

- CCC sends initialization commands to CHW
- Verify that CHW operational via tests
  Exception Course: CHW initialization error

Postcondition: State(CHW) = IDLE

Exception Course: CCC initialization error

- Raise error condition and log error locally which can be polled by ACC
- If ACC Connection error
  - Display error message on CCC console
- End if

Postcondition: State(CCC) = FAULTED

Exception Course: CHW initialization error

- Raise error condition and log error locally which can be polled by ACC
- Execute Reset Correlator use case

Postcondition: Postcondition of Reset Correlator use case.

3.2 Use Case: Correlator Data Observation

This use case describes the execution of an observation using the correlator when the antennas are in single dish or interferometric mode. This is a multi-step process comprised of the following: observation configuration, start observing, obtain and process raw correlator lags, transmission of spectral results to the Data Collector and stop observing. Recall that an observation contains one or more integrations and that an integration contains one or more dumps.

States

- **IDLE** – Correlator subsystems initialized, functioning correctly and ready to receive commands
- **CONFIGURED** – Correlator subsystems are configured for an observation
- **CORRELATING** – Correlator subsystems are performing an observation
- **FAULTED** – An error occurred in a correlator subsystem.

Actors

Primary:

- Array Control Computer (ACC)
- Data Collector
- Correlator Control Computer (CCC)
- Correlator Data Processor (CDP)

Secondary:

- CHW

Priority: Critical

Performance: The CCC must keep up with the transfer of lags from the CHW within the specified dump time.
Exception Course: CCC misses one lag set

Frequency: As needed

Preconditions: CCC and CHW have been initialized. State (CCC), State (CHW) = IDLE

Basic Course:

- Subflow: Configure observation
- If State(CCC) & State(CHW) = CONFIGURED
  - Subflow: Start observing
  - While State(CCC) = CORRELATING
    - Subflow: Process raw correlator lags from a dump
    - Subflow: Send spectral results per integration to Data Collector
    - If number of integrations reached OR Stop Observing command received from ACC
      - Subflow: End observation
    - End if
  - End while
- End if

Post conditions: Dependent on subflow post condition

Subflow: Configure Observation

- CCC receives observation configuration description from the ACC which specifies the following:
  - dump time
  - integration time
  - number of integrations to perform
  - number of bins 1, 2, or 4 and bin switching time in 48 ms increments
  - correlator system mode (see Table 2)
  - If observing in interferometric mode
    - Geometric delay model coefficients for each antenna
  - End if
  - data processing options. Data processing options include:
    - Booleans to perform FFT, Hanning windowing, van Vleck correction
    - Spectral channel decimation
    - Spectral channel range
    - Spectral averaging

  Exception Course: Correlator configuration invalid parameters
- CCC sends observation configuration description information to CDP which the CDP utilizes in preparation of processing the correlator lags
- CCC transmits configuration relevant information to CHW
- If using 100 MHz bandwidth mode
  - Download 100 MHz FPGA images to CHW
    Exception Course: Error configuring 100 MHz mode
- End if
  Exception Course: CHW not configured.
- CCC sends CONFIGURED to ACC

Post conditions: State(CCC), State(CHW) = CONFIGURED

Subflow: Start Observing

Precondition: State(CCC),State(CHW) = CONFIGURED
Exception Course: Not configured for observation

- CCC receives a “start observing” command from the ACC specifying on which timing event to start integrations (TE_{start}) either in synchronized mode or not.

- Wait until one timing event before TE_{start} and then do the following:
- If observing in interferometric mode
  - Evaluate geometric delay model and apply the coarse delay to CHW before the first dump of an integration starts. Due to limitations in the test correlator, coarse delays can only be applied before an integration starts and cannot change during an integration.
- End if
- Send start observing command to CHW so that it starts integrations at TE_{start}.
- CCC records array time of start of observation corresponding to TE_{start}
- CCC notifies CDP that observation start command received initializing integration results accumulator.
- Send OBSERVING STARTED response to ACC after $TE_{start.}$

Post conditions: State(CCC), State(CHW) = CORRELATING

Subflow: Process Correlator Lags from a Dump

Precondition: State(CCC), State(CHW), State(CDP) = CORRELATING

- If first dump
  - Initialize integration accumulator.
- End if
- Wait for a set of raw lags from the CHW resulting from a dump
  Exception Course: Cannot obtain raw lag results from CHW
- Optionally calculate van Vleck, Hanning, and FFT
- If observing in interferometric mode
  - Apply fine delay correction
- End if
- Apply spectral averaging and/or spectral decimation if applicable.
- Add dump spectral results to integration results accumulator

Post conditions: State(CCC), State(CHW) , State(CDP) = CORRELATING

Subflow: Send Spectral Results per Integration to Data Collector

Precondition: State(CCC), State(CHW) = CORRELATING and integration duration complete flag set

- Create header for spectral data block identifying correlator configuration, data processing that occurred, integration time stamp, integration duration, bin number and dump duration.
- If last integration
  - Flag last integration in header
- End if
- Append spectral data set to header
- Send spectral results to Data Collector
  Exception Course: Connection to Data Collector lost.

Post conditions: State(CCC), State(CHW) = CORRELATING

Subflow: End Observation

Exception Course: Previous subflow step post conditions not met

- CCC either receives “stop observing” command from ACC or the number of integrations has been reached.
- Send STOP OBSERVING command to CHW
- Record stop observation time
- If “stop observing” command received
  - Process a possible partial set of raw lags via subflows: Process Correlator Lags from a Dump and Send Spectral Results per Integration to Data Collector flagging last integration.
- End if
- Send STOP OBSERVING response to ACC which includes the stop observing time.

Post conditions: State(CCC), State(CHW) = IDLE

Exception Course: CCC misses one lag set

This can occur if a new set of lags arrives before the current set being processed does not complete. This essentially means that the dump time is too short.

- Flag error: New lag set arrived prematurely
- Complete current data processing
- Discard newly arrived lag set
- Although the integration duration remains the same, the recorded duration is decreased by one dump duration.

Post conditions: State(CCC) = CORRELATING

Exception Course: Correlator configuration invalid parameters

- Flag error reflecting invalid parameter(s)
- Return NOT_CONFIGURED to ACC

Post conditions: State(CCC) = IDLE (Note not CONFIGURED)

Exception Course*:* Error configuring 100 MHz mode

This exception occurs if the CCC can't read the FPGA image from disk.

- Flag error: Unable to configure 100 MHz mode
- Abort remainder of configuration – unable to proceed with observations in this state.
- Return NOT_CONFIGURED to ACC
- Note that 800 MHz parameters are resident in the test correlator and there is no fault detection at the CCC level.

Post conditions: State(CCC) = FAULTED

Exception Course: Not configured for observation

This exception occurs if the ACC commands an observation to start without being configured first.

- Flag error that an observation was attempted to start without being in the configured state.
- Return NOT_STARTED_OBSERVATION to ACC

Post conditions: State(CCC) = IDLE

Exception Course: CHW not configured

- Flag error: CHW Not Configured
- Abort remainder of configuration – unable to proceed with observations in this state.

Post conditions: State(CCC) = FAULTED

Exception Course: Cannot obtain raw lag results from CHW

- Flag error: Unable to obtain raw lags from CHW
- If successful in retrying retrieval of raw lags within a time limit that allows for sufficient time to process results.

  Post conditions: State(CCC), State(CHW) = CORRELATING
- Else

  Post conditions: State(CHW) = FAULTED
- End if

Exception Course: Lose connection to Data Collector

- Flag error: Data Collector connection lost
- Notify ACC that data connection lost
- Stop observing

Post conditions: State(CCC) = FAULTED

3.3 Use Case: Correlator Monitor Information

This use case describes the polling of correlator hardware monitor data and the publishing of this data.

States

- **IDLE** – This is a valid state in which monitor information is available
- **CORRELATING** – This is a valid state in which monitor information is available
- **FAULTED** – A correlator subsystem has encountered a fatal error

Actors

Primary:

- Array Control Computer (ACC)
- Correlator Control Computer (CCC)

Secondary:

- Correlator Hardware (CHW)

Priority: Critical

Performance: ~0.5 secs.

Frequency: once every 10 – 30 seconds

Preconditions: CCC, and CHW have been initialized. State (CCC) , State (CHW) != FAULTED

Basic Course:

- CCC periodically polls CHW for all monitor data points
- Deliver monitor data points with time stamps to ACC
- If monitor data point(s) unavailable
  Exception Course: Monitor values unavailable
- End if

Postcondition: Monitor values sent to ACC

Exception Course: Monitor values unavailable

- Log an error that the CHW is not responding
- Deliver this monitor value flagging it as unavailable.

PostCondition: State(CHW) = FAULTED. This is because the CHW is not responding to commands from the CCC and cannot be controlled by it.

3.4 Use Case: Correlator Reset

This use case describes a reset of the CHW and/or the CCC. The CHW can be "warm reset", i.e., it microprocessors can be reset. The CCC can be rebooted allowing the CPU to reload the OS, and reload and restart the correlator control application.

States

- **IDLE** – Correlator subsystems initialized, functioning correctly and ready to receive commands.
- **FAULTED** – CCC or CHW does not recover from reset and transition to the IDLE state.

Actors

Primary:

- Array Control Computer (ACC)
- Correlator Control Computer (CCC)
- Operator who can manually reset the CCC via power-on switch or the CCC's reset button.

Secondary:

- Correlator Hardware (CHW)

Priority: Critical

Performance: N/A

Frequency: As needed

Preconditions: State(CHW) = Don't care, state(CCC) = Don't care unless resetting the CHW, then state(CCC) = IDLE

Basic Course:

- If CCC receives RESET command from ACC
  - If RESET is warm reset for CHW
    - CCC sends warm reset command to CHW
  - Else If RESET is cold reset for CCC
    - Cause CCC to reset
  - End if
  - Execute Test Correlator Initialization use case
- Else hard reset is performed by operator physical switch
  - Execute Test Correlator Initialization use case
- End if

Post conditions: State(CCC), State(CHW) = post condition of Test Correlator Initialization use case.

3.5 Use Case: Execute Diagnostic Check

This use case occurs when the ACC commands the CCC to perform a diagnostic check on the CHW components. In addition, this use case covers checking the sampler threshold values. These tests include communication paths between the CHW and the CCC and internally between correlator cards in the CHW. Note that during a diagnostic check, "normal" correlator functionality is unavailable.

States

- **IDLE** – Correlator subsystems initialized, functioning correctly and ready to receive commands
- **DIAGNOSTIC** – Reflects that the CCC is performing diagnostics and should not be interrupted.
- **FAULTED** – CHW does not communicate with CCC

Actors

Primary:

- Array Control Computer (ACC)
- Correlator Control Computer (CCC

Secondary:

- Correlator Hardware (CHW)

Priority: Critical

Performance: About a second

Frequency: As needed

Preconditions: State(CCC), State(CHW) = IDLE

Basic Course:

- CCC receives run diagnostic check command from ACC
- Set State(CCC) = DIAGNOSTIC
- If performing sampler threshold check

        - For each sampler S
            - For each bit B
                - Configure CHW configuration information for sampler S bit B
                - CCC starts 1 dump
                - CCC returns raw lags to ACC
            - End for
        - End for
    - Else
        - CCC executes requested diagnostic tests on CHW
        - *Exception Course*: CHW does not respond to CCC
        - CCC returns results of diagnostic tests to ACC
        - CCC executes Correlator Reset use case with CHW warm reset
    - End if

Post conditions: State(CCC), State(CHW) = IDLE

Exception Course: CHW does not respond to CCC

- Flag error and wait for resolution of problem by ACC

Postcondition: State(CCC) = IDLE, State(CHW) = FAULTED

3.6 Use Case: Request Characteristic Information

This use case occurs when the ACC requests the CCC's characteristic information.

States

- **IDLE** – Correlator subsystems initialized, functioning correctly and ready to receive commands

Actors

Primary:

- Array Control Computer (ACC)
- Correlator Control Computer (CCC)

Secondary:

- N/A

Priority: Critical

Performance: About a second

Frequency: As needed

Preconditions: State(CCC) = IDLE

Basic Course:

- CCC receives request for characteristic information from ACC
- CCC returns requested characteristic information to ACC (see Table 4 for details).

Post conditions: State(CCC) = IDLE

4 Correlator Control Software Overview

Figure 1 shows the main components of this design and their interfaces to external devices. The two components are the Correlator Controller, which interfaces to the ACC and the CHW, and the Correlator Data Processor, which interfaces to the Data Collector and the CHW. The term "Correlator Controller" is introduced which refers to the correlator control aspect of the correlator control computer software and provides the distinction from the physical hardware encompassed by the term "CCC" .

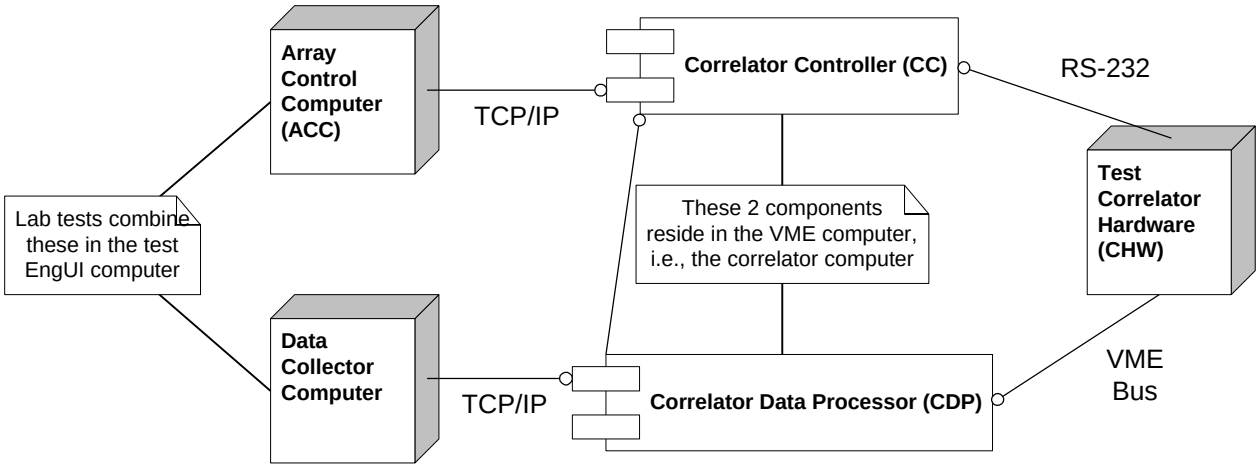

Figure 1 – Correlator Computer System Component Diagram

The correlator controller's purpose is to provide an interface between the CHW and the ACC and to monitor and control the CHW. The correlator controller also acts as a conduit for configuration and control information from the ACC to the CDP. Note that the correlator control computer is a "slave" device, that is, it only operates on the correlator hardware when told and performs only the functions that the ACC instructs it to do.

The purpose of the CDP is to obtain the lag results from the CHW's LTA (Long Term Accumulator), optionally convert the lag results to spectral points and transmit blocks of spectral data to the Data Collector for further processing and storage. The correlator controller and CDP components reside in the correlator control computer.

The correlator computer is a Motorola MVME-2700 266 MHz VME single board computer (SBC). This SBC is based on an MPC750 PowerPC microprocessor. The VME computer runs VxWorks RTOS ver. 5.4 with Tornado II. The engineering operational interface for the correlator computer is the Tornado II hosted on either a Windows NT or Sun computer. In addition, a test engineering user interface (TC_EngUI) application exists whose details are described in § 13.2 and assists with operation and debugging. There will be a period when the correlator will run in a "stand-alone" mode where the ACC and Data Collector computers will be simulated the TC_EngUI application. This application can run on the same computer as the Tornado II host or a separate computer.

There exists a set of C software routines for the MAC spectrometer authored by Jeff Hagen. Some parts of his code have been utilized in the following design. There were two parts of the Hagen code, a set of test routines which were used to perform a complete system test on the spectrometer hardware and a set of routines which performed observations for the 12-meter telescope in Tucson. This software has been in operation for about 2 years at the 12-m. The test software is used pretty much "as is" to validate the test correlator hardware with modifications necessary due to a different real-time computer platform while only a fraction of the operational part is used.

This document describes an object-oriented design that utilizes UML diagrams. If the reader is unfamiliar with UML, then [9] provides an adequate overview to familiarize oneself with UML.

4.1 Correlator Software System Packages and Functional Overview

The correlator software has to perform following high-level functions:

- Initialization of the correlator subsystems
- Observation control
- Processing of raw lags
- System monitoring and error logging
- Array time tracking and synchronization utilities
- Maintaining characteristic information
- Communication with external control and data storage systems
- CHW diagnostics including sampler threshold checking

Figure 2 shows the package diagram that defines the highest level objects of the correlator computer software . These packages implement the above operations. As the correlator software is a real-time software application, the description of these packages will include both the functional and time-critical requirements.

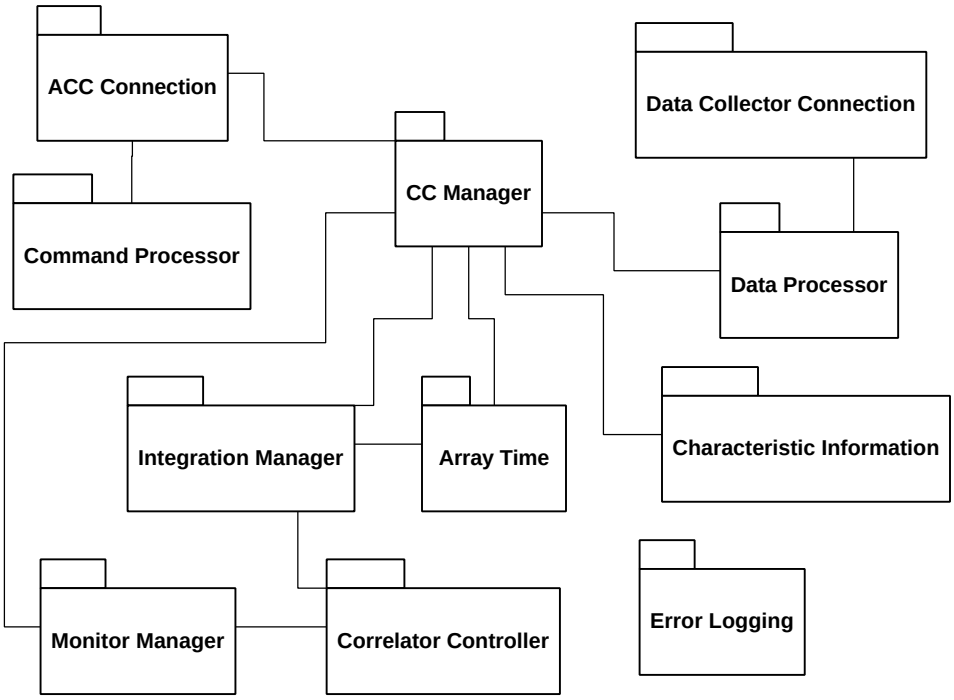

Figure 2 – Test Correlator Package Diagram

4.1.1 ACC Connection Package

This package provides an interface layer between the ACC and the correlator controller . The command architecture is considered master-slave in which the ACC initiates commands and the correlator controller responds. All commands respond with either an acknowledgment in the case of a command or a response of requested information thus enabling the ACC to quickly determine the command execution status. The connection protocol will be CORBA.

4.1.2 Array Time Package

This package has two responsibilities: 1) to maintain array time within the correlator computer by counting the external 48 ms Timing Events and to make array time available to all other packages needing array time services, e.g., time stamps of monitor and spectral data and 2) to provide accurate internal timing events for synchronization with the 48 ms Timing Events. The CC Manager and Integration Manager packages utilize the Array Time package for these synchronization items.

4.1.3 CC Manager Package

This is the highest level package whose responsibilities include:

- Providing a single point of control, thus enforcing a master-slave architecture within the correlator computer software
- Interfacing to the ACC
- Processing and routing of command objects from the ACC to other packages and returning responses from them to the ACC (assuming a command-driven interface between the ACC and the correlator computer)
- Maintaining the state of the correlator computer
- Executing diagnostic tests

- Assists in coordinating events that must be synchronized to the 48 ms Timing Events.

4.1.4 Characteristic Information Package

This package provides correlator system characteristic information to the ACC. This characteristic information is currently hard-coded into the software, but could be transparently transferred to a configuration database in the future if any benefit could be derived from this.

4.1.5 Command Processor Package

The ACC Connection package uses this package as a lexical analyzer of simple ASCII strings commands sent from the ACC to create command objects. The ACC Connection package then transfers the command objects to the CC Manager which in turn delivers them to the appropriate package for execution. The intention is to replace the ASCII string implementation with a CORBA interface. These details are well encapsulated and will not result in major disruptions in the software.

4.1.6 Correlator Controller Package

The Correlator Controller package provides an interface layer between the Integration Manager and the CHW via a serial port connection. It encapsulates the details of this communication link between the correlator computer and the CHW.

4.1.7 Data Processor Package

This package obtains raw correlator lag results from the CHW's LTA, places the lags into the correct order, processes them into spectral results and delivers spectral data blocks to the Data Collector through the Data Collector Connection Package. This package has a direct connection to the CHW via the VME bus as show in Figure 1.

4.1.8 Data Collector Connection Package

This package provides an interface between the Data Processor package and the Data Collector computer. Note that this connection is uni-directional, i.e., data flows only from the Correlator Control Computer to the Data Collector. The connection protocol will be CORBA.

4.1.9 Error Logging Package

This package provides a globally accessible repository for run-time errors that occur in all of the correlator computer's packages. A listing of run-time errors is accessible to the ACC.

4.1.10 Integration Manager Package

The primary responsibility of this package is to control observations using the Correlator Controller package. The Integration Manager Package configures the CHW including downloading of Xilinx personalities used for 100 MHz modes (the 800 MHz modes are resident in the CHW and don't require downloading), starts and stops integrations, and helps coordinate the starting and stopping of integrations with the starting and stopping of the data collection process performed by the Data Processor package.

4.1.11 Monitor Manager Package

This package manages the correlator hardware monitoring and relays monitor information to the ACC. It obtains information through Correlator Controller package and uses the ACC connection package to deliver monitor data to the ACC.

5 Time

There are two basic time constants important to the test correlator system: the basic tick of the correlator hardware (1.31072) and array time which is obtained by reading a master clock that counts 48 ms (20.833… Hz) timing events (TEs). Timing event pulses will be available to all devices for highly accurate timing including the correlator computer and the CHW. TEs are counted internally in the correlator controller to increment array time which is initialized via the array time master clock. The correlator controller will utilize an on-board watchdog timer to detect if any timing event is missed. An discrepancies between array time in the correlator controller and the master clock will constitute a serious fault which will require resynchronization with the master clock. There is no need for test correlator to keep track time at a finer resolution than 1 ms. See [7] and [1] for details regarding array time.

Correlator integration times are divided into three levels, bin switch time, dump time and integration time.

- The bin switch time is an integral number of TEs which the correlator uses to switch short-term integrations into separate memory bins. Each bin switch occurs on the leading edge of a TE allowing for synchronized frequency or nutator switching.
- The dump time is the time interval at which lag results are available to the correlator data processor. An integration is a set of dumps that are converted to spectral data and summed together internally in the correlator computer before being delivered to the Data Collector computer.
- The integration time is a multiple of correlator dump times.

Time stamps also use array time. This applies to time stamps that mark integration times and for monitor point values.

5.1 Timing Synchronization Issues

Synchronization of the correlator computer and CHW with the rest of the array is provided by the 48 ms timing events. Time-critical commands from the ACC to the correlator controller are time stamped with the array time at which they are to be executed at the hardware device level. Synchronization is required only when starting an observation. It will be incumbent on the ACC to deliver the start observation command far enough in advance to allow for latency in the network and in correlator controller to process the command and then issue a start observing command to the CHW. Figure 3 gives an example of this where the ACC sends a “start observing” command with a time tag of t_m so that the CHW starts observing on the leading edge of timing event **m**. In practice, the correlator controller must receive the ACC’s time-tagged start observing command by timing event $\mathbf{t_{m-2}}$, that is, two timing events before the actual CHW starts correlating to ensure that the correlator controller has enough time to process the start observing command.

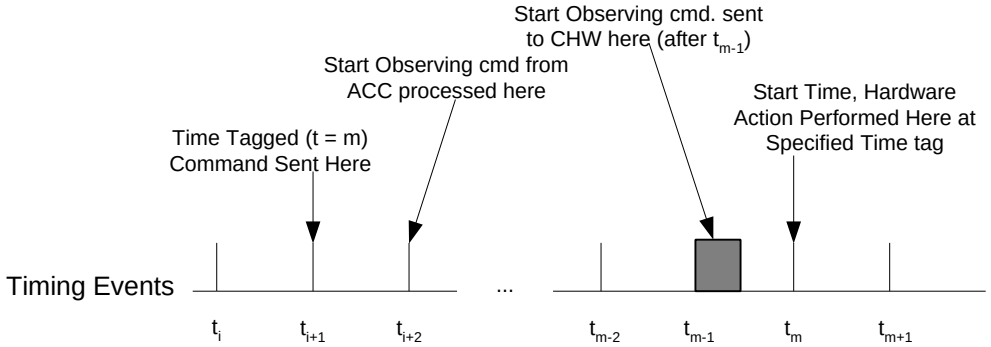

Figure 3 – Start Observation Timing Synchronization

Short term integrations at the correlator chip level can be synchronized to array TEs. In this case, 36 short term correlator integrations occur and then blanked until the leading edge of the TE. The starting of integrations is identical to the non-synchronized case.

Figure 4 shows the timing of synchronized short-term integrations using two bins with a bin switching time of 48 ms and a dump time of 192 ms. It also shows the short term integrations being blanked for a fraction of the 48 ms leading to a correlation minor inefficiency. Also the dump duration is 96 ms as each dump transfers 2 bins' worth of raw lags. All signals are shown as positive logic.

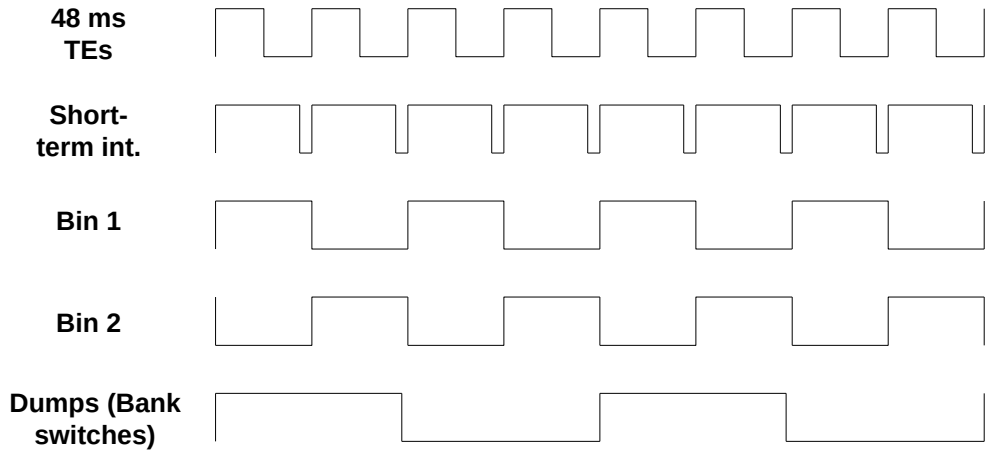

Figure 4– Synchronized Short Term Integrations

6 Initialization

Sections 6 through 11 detail the classes that contain the various functions of correlator initialization, operation and data processing and follow the functional layout described in § 4.1. These functions are then controlled by higher level task classes described in section 12. A class glossary in section 14 shows the relationship between these two sets of classes.

Initialization is a set of procedures the CC Manager performs at correlator computer power-up or reset. There are three areas that are addressed at initialization: correlator hardware initializations, internal systems and connection(s) to the ACC and Data Collector. After initialization is performed, any possible initialization errors are recorded locally and are available to the ACC except a failure to connect to ACC.

6.1 Initialize Correlator Computer

Correlator computer initialization performs internal checks to ensure that all packages are functioning correctly, e.g., tasks are correctly started, objects are instantiated, communications among the packages are okay, correlator hardware initializations were successful, etc.

Initialization of the ACC Connection and Data Collector Connection packages start processes which allow for connections from the ACC and Data Collector clients. If errors occur with the ACC Connection package, then they are displayed on the VxWorks console as there is no means of transmitting these errors to the ACC.

6.2 Initialize Correlator Hardware

The CC Manager performs CHW initialization using the Correlator Controller package. The following are required to initialize the test correlator hardware:

- Load initialization sequences into the correlator control, LTA and system monitor control cards.
- Perform simple system checks on each control card to ensure that the serial connections between the correlator computer and the embedded microcontrollers are valid. Do a similar check for VME bus-connected devices.
- If any errors occur, record them and set the state to FAULTED. This will require rebooting or further diagnosis.

7 Observation Control

This section discusses the steps to configure, start, process lags, and stop observations. All the packages in Figure 2 are utilized except for the Monitor Manager and Characteristic Information packages. Control for an observation addresses the following areas:

- Preparation for acquiring data involving the reception of configuration information from the ACC, setting up internal data structures describing the observation and relaying relevant configuration information to the CDP and configuring the geometric delay model evaluator as needed.
- Configure the CHW in preparation for an observation.
- Applying a coarse delay as needed when using interferometry mode.
- Starting integrations.
- Retrieving raw lag results from the LTA which is a function performed by the CDP.
- Processing correlator results done by the CDP, i.e., FFTs, fine delay adjustments, etc.
- Transmitting the spectral results to the Data Collector.
- Repeating the above 3 steps until integrations are stopped..

Figure 5 is a sequence diagram for an observation which shows the main flow of control between the two external system actors, the ACC and the Data Collector, and the internal correlator subsystems actors, the correlator controller , CHW and CDP. Messages flow in from the ACC as commands and are received and handled by the CC Manager which distributes them to the appropriate handlers for further processing. Spectral results flow out of the CDP to the Data Collector. A complete discussion of all these processes follows.

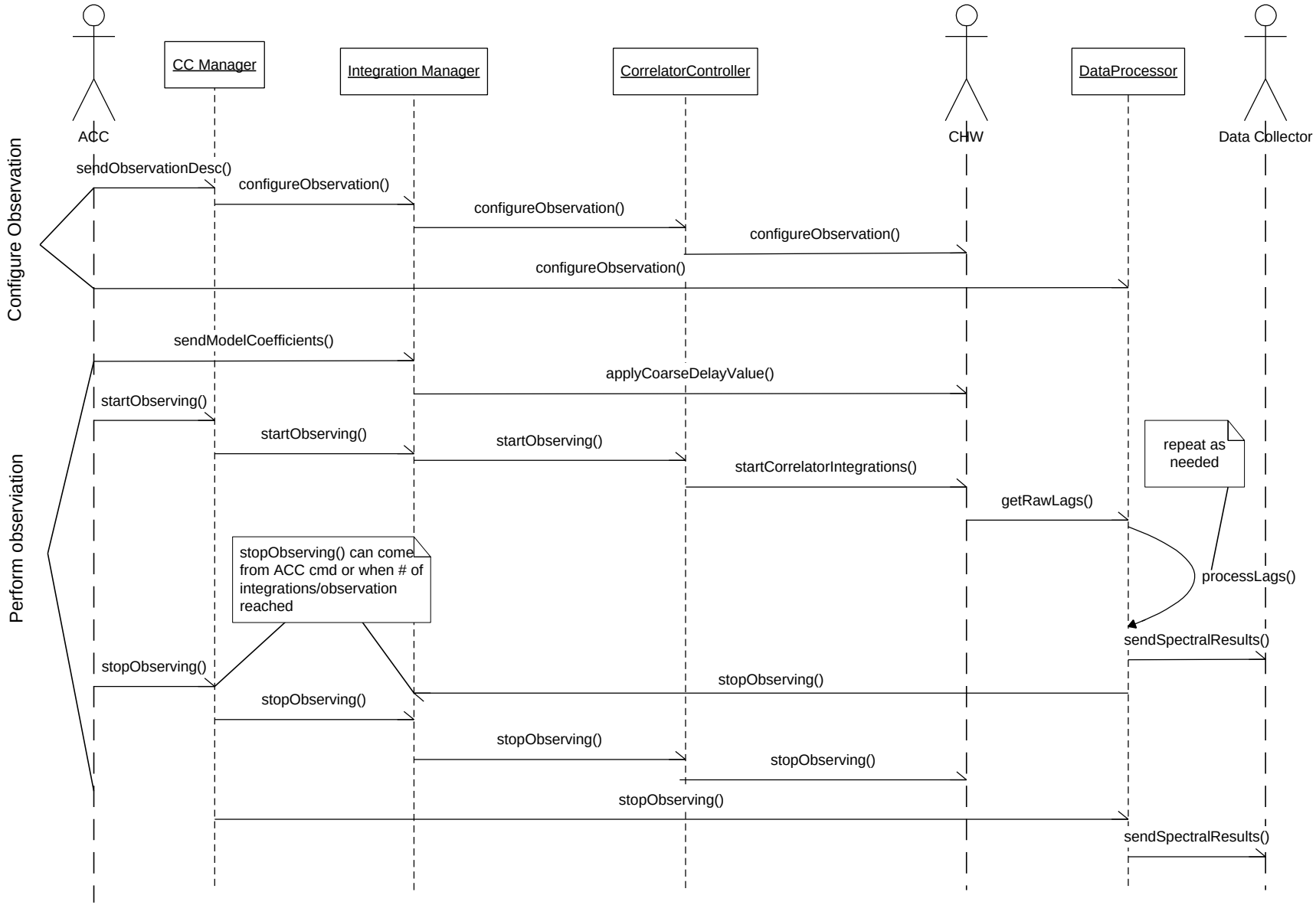

Figure 5 – Observation Sequence Diagram

7.1 Preparation for acquiring data

The CorrelatorConfiguration class represents the current configuration of the correlator. This configuration is provided in the observation configuration description sent by the ACC. It is important to note that the test correlator contains one active configuration parameter set for the entire correlator. Therefore multiple frequency bands, resolutions, dump times etc., **cannot** be configured for different antennas.

7.1.1 Correlator Configuration Class

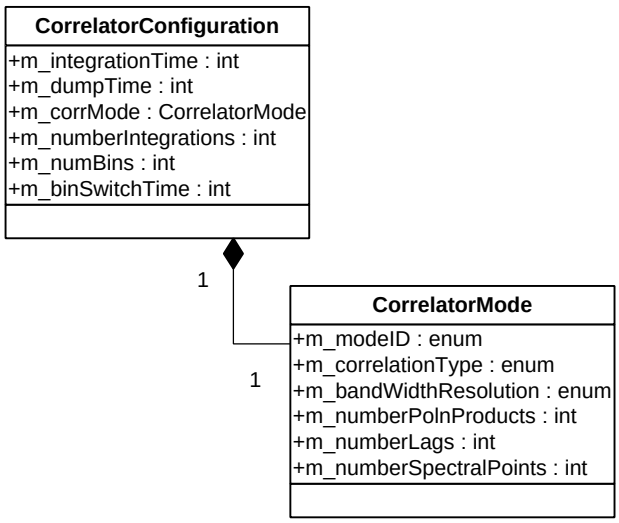

Figure 6 – CorrelatorConfiguration Class

The CorrelatorConfiguration class contains the following information:

- **Dump Time**

  This is the correlator dump duration represented as the number of correlator ticks (1.31072 ms)

- **Integration time**

  An integer representing the integration duration as a multiple of the dump duration.

- **Number of integrations**

  This integer specifies the number of integrations to perform. When this quantity of integrations is reached, the correlator controller commands CHW to stop observing and the correlator becomes idle.

- **Number of correlator bins**

  This defines the number of correlator memory bins which can be 1, 2 or 4. Multiple bins allow for synchronizing the correlator short-term integrations with the 48 ms timing events. Multiple bins would be used for 90° phase switching, frequency switching, nutator switching, and testing of side band suppression by integrating each side band into a separate bin. Spectral results for each bin are separately sent to the Data Collector, as each bin has an entire set of raw lags.

- **Bin switch time**

  When multiple bins are use, the bin switch time defines an integral number of 48 ms time intervals to be spent in each bin. For example, if 2 bins are specified with a bin switch time of 2, then every 96 ms a bin switch occurs.

- **Correlator Mode**

  The CorrelatorMode object encapsulates the correlator system mode information in Table 2. A CorrelatorMode object can be constructed with a Mode ID which avoids having to specify each member variable.

  - **Mode ID**

    The mode id identifies the configuration setting allowing one to set an aggregate of mode parameters that include auto or cross correlation, bandwidth resolution, number of lags and polarization products.

  - **Correlation Type**

This enumerated value specifies if the mode is an auto- or cross-correlation mode.

- **Bandwidth Resolution**

  This enumerated value defines the observing bandwidth as either 800 MHz and 100 MHz.

- **Number of Polarization Products**

  This specifies the number of polarization products for a given correlator system mode which can be either 2 or 4.

- **Number of Lags**

  Based on the bandwidth resolution, this integer defines the number of lag results which is either 1024, 2048, 4096, 8192 or 16384. See Table 2 for details.

- **Number of Spectral Channels**

  This is the number of final spectral channels that is based on the number of lags and correlation type, which is either 512, 1024, 4096 or 8192.

CORRELATOR MODE ID	BANDWIDTH	POLARIZATION PRODUCTS	LAGS	DELAY RESOLUTION	DELAY RANGE
1 (cross-products)	800 MHz	0R X 1R (1) 0L X 1L (2) 0R X 1L (3) 0L X 1R (4)	512 Leads & 512 Lags	5 ns	10 μs
2 (cross-products)	800 MHz	0R X 1R (5) 0L X 1L (6)	1024 Leads & 1024 Lags	5 ns	10 μs
3 (self-products)	800 MHz	0R X 0R (7) 0L X 0L (8) 1R X 1R (9) 1L X 1L (10)	1024 Lags	N/A	N/A
4 (cross-products)	100 MHz	0R X 1R (11) 0L X 1L (12) 0R X 1L (13) 0L X 1R (14)	4096 Leads & 4096 Lags	20 ns	80 μs
5 (cross-products)	100 MHz	0R X 1R (15) 0L X 1L (16)	8192 Leads & 8192 Lags	20 ns	80 μs
6 (self-products)	100 MHz	0R X 0R (17) 0L X 0L (18)	8192 Lags	N/A	N/A
7 (self-products)	100 MHz	1R X 1R (19) 1L X 1L (20)	8192 Lags	N/A	N/A

Table 2 – ALMA Test Correlator Configuration Modes

One important validity check for correlator configuration parameters is to ensure that the correlator dump time is sufficiently long to allow for extracting the raw lags and processing them to spectral results. This are discussed in § 12.2.

7.2 Configure the correlator hardware

The IntegrationManager class is the primary class contained in the Integration Manager package. It is responsible for configuring the CHW based upon the active CorrelatorConfiguration object and for controlling observations at the CHW. The IntegrationManager utilizes the CHW_Controller (see § 8) to convert the data in the CorrelatorConfiguration object to serial commands understood by the CHW.

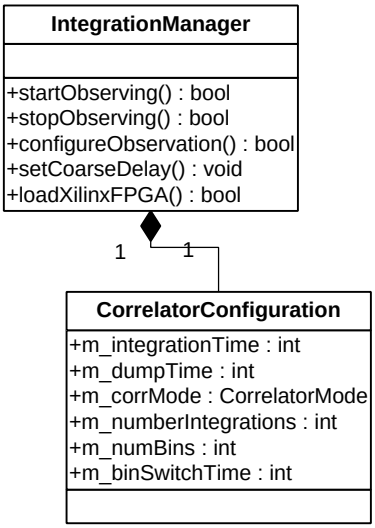

Figure 7 – IntegrationManager class

The data members of this class are:

- **m_correlatorConfiguration** – the correlator configuration for the observation

The member functions associated with configuration

- **configureObservation()** sends the appropriate configuration or integration control commands to the CHW
- **setCourseDelay()** sends the coarse delay value based on the DelayModel in the CorrelatorDataProcessor object if this is a cross-correlation observation.
- **loadXilinxFPGA()** manages the download of the FPGA images for the 100MHz narrow bandwidth modes. Due to their size, the FPGA images take about 40 seconds to download to the CHW during which no other correlator functionality can occur, i.e., the correlator controller is blocked.

7.3 Configuring Data Processing

The next step in configuration for an observation is to configure the CorrelatorDataProcessor object for data processing options (see § 7.5 for details). These configuration values come from observation configuration description and set the appropriate data members of CorrelatorDataProcessor. The CC_Manager transmits this configuration information to the CorrelatorDataProcessor object.

7.4 Starting and stopping correlator integrations via the LTA

Once the correlator is configured for the observation, the correlator controller waits for a “start observing” command from the ACC which it then commands the CHW to start observing via the IntegrationManager::startObserving() method. If the CorrelatorConfiguration specifies cross correlation mode, the correlator controller first evaluates the delay model and applies the coarse delay value to the CHW again utilizing the IntegrationManager class.

Stopping observations happen when 1) the number of integrations specified in the active CorrelatorConfiguration are reached or 2) the ACC sends a “stop observing” command which completes the current integration or 3) the ACC sends an “stop observing” which immediately halts the integration returning any partial integration results.

7.5 Retrieving and processing raw lag results from the LTA

Data processing is a multi-stage process within the Data Processing package. Lags are obtained from the CHW, converted to spectral points and summed into an integration accumulator, a header identifying the spectral results is attached to the results (defined by the CorrelatorScienceDataResults class) and then spectral results are passed onto the Data Collector. All of these functions are managed by a higher-level task as discussed in § 12.1.6.

Completion of these time-critical functions must occur before the next set of lags are dumped from the LTA. If the data processing tasks are not completed within the allowed time, then an error is logged, the newly arrived lags are ignored and the total integration time is decreased by one correlator dump period. The way to prevent this overrun situation is empirical – first calculate the data transfer and data processing times in tests and then use these values to establish the minimum correlator dump times. Note that this situation is not fatal, it means a loss of observing time. Checks of integration and dump times can be made in the configuration stage to avoid overrun problems. Estimations of data processing rates are discussed in [4] which shows that the maximum processing load is approximately 2 MFLOPs. The MVME-2700 VME computer has an estimated rate of approximately 250 MFLOPS which is more than sufficient to handle the output data rates of the test correlator. Preliminary timing analysis has shown that it takes at about 3 ms to process 1K lag results and that the CPU has attained a rate of ~170 MFLOPS.

Once a set of raw lags from an LTA dump has been obtained they then are processed by the CorrelatorDataProcessor class. The two processing options are: 1) convert the lags into spectral points with the following steps (some of which are optional), or 2) pass the raw lag results directly onto the Data Collector (which is usually reserved for debugging). The possible processing steps occur in the following order:

- van Vleck correction.
- Hanning windowing function.
- FFT.
- Fine delay correction.
- Average the spectral channels.
- Spectral channels decimation.
- Accumulate the spectral data set into the integration accumulator (handled by a higher-level task).

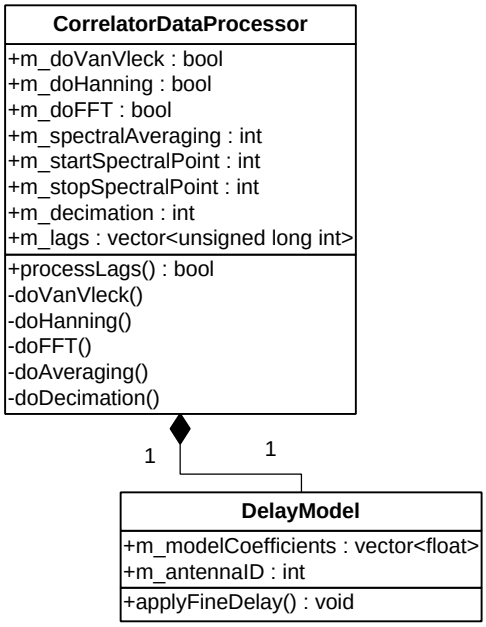

Figure 8 – CorrelatorDataProcessor class

The information to perform these various steps is obtained from the observation configuration description sent by the ACC and held in the CorrelatorDataProcessor class whose members are:

- **Lags**

  This array hold the raw lag results from a dump as 32-bit unsigned integers.

- **A set of Boolean flags to perform data processing functions**
  - **van Vleck Correction**

    Performs the van Vleck quantization correction of the data in the lag domain which corrects for the discreet clip-

ping of the input signal.

- **Hanning Smoothing**

  Performs a Hanning windowing function on the lag results.

- **FFT**

  Performs an FFT using the MIT FFTW library (see http://www.fftw.org for details). If this flag is not set, then raw lag results are sent to the Data Collector (as a diagnostic tool).

- **Delay Model**

  The Delay Model object holds the geometric delay model coefficients for an antenna in the correlator configuration for interferometry observations. It is assumed that the delay model is applied to one antenna while the other antenna is "fixed". The DelayModel class has the following members:

  - **antennaID**

    This is the antenna number to which the model coefficients apply. The delay model, as described in § 7.6, applies the delay correction on an antenna basis.

  - **modelCoefficients**

    This is an array of floating point values which hold the coefficient values for the given antenna.

- **Decimation of Spectral Points**

  This integer number defines which spectral points to discard, or decimate. The values can be 0 - no decimation, 1 every other point discarded, 2 – one point every two points, 3 – one point every three points, etc., up to 10.

- **Spectral Averaging**

  Allows the spectral points to be averaged together. The values of 2, 3, 4, etc. refer to the number of adjacent channels averaged together. The number of spectral points decreases accordingly with the average, e.g., ½ the channels result for an averaging of 2.

- **Starting/Stopping Spectral Points**

  This selects the spectral channels of interest either for spectral or lag points to be delivered to the Data Collector after all data processing steps have been performed.

7.6 Delay Tracking

When the correlator is operating in interferometry mode, the delay in the signal paths must be adjusted. This delay adjustment is time-dependent and is applied in two steps. The coarse delay is applied to the CHW before an integration starts and is valid throughout the integration keeping in mind the one-second integration limit in interferometry mode. A find delay adjustment is made on the spectral results for each dump before they are accumulated in the integration accumulator.

The DelayModel member object of the active CorrelatorDataProcessor object contains delay model coefficients for a specific antenna in the baseline – it is assumed that the other antenna's delay is zero. The delay model coefficients are provided by the ACC using the delay model server from Benson which is based on the Mark III analysis software (CALC/SOLVE) from Goddard [10]. Evaluation of a delay model equation provides the necessary delay correction and is of the form:

$$\mathbf{d}t(\mathbf{t}) = \mathbf{D_0} + \mathbf{D_1 t} + \mathbf{D_2 t^2} + \mathbf{D_3 t^3} + \mathbf{D_4 t^4}$$

where $\mathbf{D_0}$ - $\mathbf{D_4}$ are coefficients that depend on the baseline between the two antennas and their pointing position and **t** is the current time relative to the start of the integration.

The coarse delay sent to the CHW before an integration starts is of the form:

$$\mathbf{D_{coarse}(n) = round(\ D_{off}(n) + d}t\mathbf{(\ n,\ t_0 + t_{delay}/\ 2\ )\ /\ t_{resolution})} \qquad \mathbf{(1)}$$

where:

$\mathbf{D_{coarse}(n)}$	the coarse delay for antenna **n**
$\mathbf{D_{off}(n)}$	a constant delay offset for antenna **n** which reflects cable length differences for antenna **n**
$\mathbf{t_0}$	the start time of the integration interval

$\mathbf{t_{delay}}$	the delay setting interval, i.e., the interval of time for which the coarse delay is valid which is nominally ~ 1 second
$\mathbf{t_{resolution}}$	the delay resolution time (either 5 or 20 ns)
round()	a nearest integer function.

As the test correlator hardware is based on an auto-correlator, a modification was made to allow cross correlations. This modification requires that a delay offset be programmed for the correlator hardware so that the lags from one antenna are shifted such that they line up correctly with the lags for the other antenna. Consequently, each $\mathbf{D_{coarse}}$ sent to the CHW has this delay offset value added to it which is a fixed quantity relating to the number of lags for a given correlator system mode.

Once the lags are converted to complex spectral values, the fine delay adjustment is made. This involves evaluating the delay model equation which provides a high-resolution value for the geometric delay at a given point in time. The fine delay is simply the difference of the delay model equation ($\mathbf{dt(t)}$) and the coarse delay determined above in (1) ($\mathbf{D_{coarse}(n)}$) and then applied to the spectral data as a phase shift before being summed into the integration accumulator:

$$\mathbf{D_{fine}(n) = dt(t) - D_{coarse}(n)}$$

Another way to view the delay is that it's a phase adjustment of 2 complex numbers $\mathbf{C_1}$ & $\mathbf{C_2}$:

$$\mathbf{C_1 \cdot C_2 = R_i R_j e^{\,i(\,?1 + ?2)}}$$

7.7 Transmitting Spectral Results to the Data Collector

Once the results are processed, they are transferred to the Data Collector. For each integration of an observation, the results are provided with a header identifying information that describes the data sets of an integration. Many of these header items are copies of the configuration parameters used by the CorrelatorDataProcessor class. There is one set of spectral data results for each polarization product and each bin of an integration.

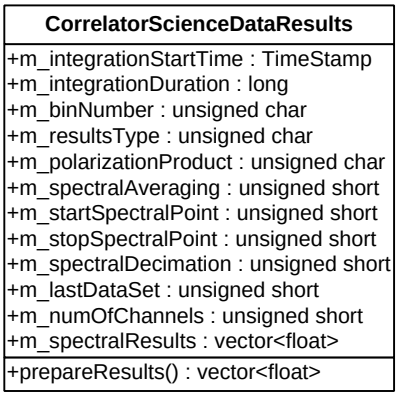

Figure 9 – CorrelatorSciecneDataResults class

The data members of the CorrelatorScienceDataResults class are (all but the last item define the header with the last item being the data):

- **Integration Start Time**

  A time stamp in units of array time that records the start of the first dump of the integration.

- **Integration Duration**

  The actual duration of the integration taking into account any missed dumps in units of 100 nanoseconds.

- **Polarization Product Index**

  From Table 2, column 3 this is the number in parentheses which uniquely identifies the polarization product6 for a given integration.

- **Correlator Bin Number**

Each bin is a separate data set with this 1-based number identifying the bin.

- **Results Type**
  This set of bit-wise flags tells what processing steps have been performed, i.e., FFT, van Vleck correction and/or Hanning windowing.
- **Number of Channels**

  This is the number of data values of raw lags or complex spectral points.
- **Spectral Averaging**
  Reflects any averaging of the spectral points. This is copied from the CorrelatorDataProcessor class.
- **Starting/Stopping Spectral Points**
  Reflects the starting and stopping spectral points. This is copied from the CorrelatorDataProcessor class.
- **Decimation of Spectral Points**
  Reflects any decimation of spectral points. This is copied from the CorrelatorDataProcessor class.
- **Last Data Set**

  A boolean value flagging this as the last set of results for an observation.
- **Spectral Results**
  These are the results as either raw lags or spectral points whose quantity is specified by the appropriate correlator mode.

7.7.1 Correlator Output Data Sizes

When the CorrelatorScienceDataResults are transmitted to the Data Collector as a block of binary data, Table 3 summarizes this data and their sizes. The number of data points, N, represents the 2 times the number of channels for spectral results as each spectral point is a complex value with real and imaginary components, or it equals the number of channels for lag results.

Description	**Data Type**	**Size (bytes)**
Integration Start Time as Array time	long long integer	8
Integration Duration	long long integer	8
Polarization Product Index	unsigned char	1
Correlator Bin Number	unsigned char	1
Results Type	unsigned char	1
Number of channels	unsigned integer	2
Spectral Averaging	unsigned integer	2
Starting Spectral Point	unsigned integer	2
Stopping Spectral Point	unsigned integer	2
Decimation of Spectral Points	unsigned integer	2
Last Data Set	unsigned char	1
Number of Data Points (N)	unsigned integer	2
Spectral/Lag Data Results	uingle-precision float	N x 4

Table 3 – Output Data Sizes

8 Correlator Hardware Interface

The interface between the high-level classes that control the test correlator and the correlator hardware is encapsulated in the CHW_Controller class. It utilizes two other classes, TestCorrelatorCommand and SerialDriver, to transmit via a serial port, the specific bytes required by the test correlator hardware for its control. As this class provides a hardware interface layer, there is only one instance of it whose access is guarded with a semaphore to avoid errors having multiple tasks communicating to the CHW.

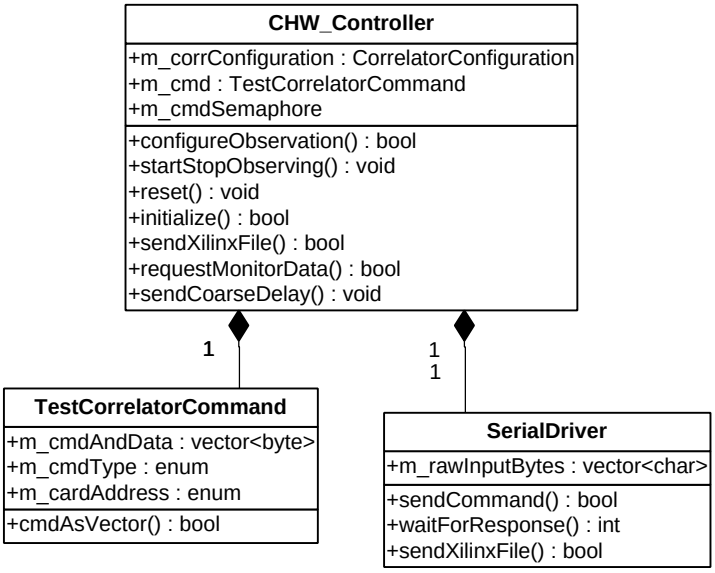

Figure 10 – CHW_Controller Class

The CHW_Controller class contains the following data members:

- **Command Semaphore**

  This is the VxWorks semaphore which controls access to the sending of commands via the serial connection to the CHW.

- **Correlator Configuration**

  A copy of the IntegrationManager's CorrelatorConfiguration object defining the current correlator configuration. It is used to configure the correlator mode and dump duration.

- **Test Correlator Command**

  This class encapsulates all of the specifics of the test correlator commands as described in [14].

- **SerialDriver**

  This object is the low-level interface to the serial port. It contains knowledge of the command protocol utilized by the test correlator and provides functions to send TestCorrelatorCommands, wait for responses from the CHW and to download Xilinx personality images when changing from wide to narrow bandwidth modes.

Besides these data members, the CorrelatorManger has many interface functions which control the correlator hardware. Each of the following functions are translated to one or more instances of the TestCorrelatorCommand class. They are:

- **initialize()**

  Initialize the CHW after a power up or soft reset, loading default configurations into various hardware cards.

- **configureObservation()**

  Allows configuration of the CHW utilizing a CorrelatorConfiguration object

- **startStopObserving()**

  Starts integrations with short-term integration either synchronized or not to the 48 ms Timing Events or stops integrations.

- **requestMonitorData()**
  Requests monitor data from the test correlator.
- **reset()**
  Performs a warm reset on the CHW.
- **applyCoarseDelayValue()**
  Sends a coarse delay value to the correlator hardware.
- **sendXilinxFile()**
  When changing bandwidth modes from 800MHz to 100MHz, this loads the Xilinx FPGA image from a disk file to the CHW.

9 Monitor Information

The test correlator's system monitor card provides the monitor information that can be retrieved by the correlator controller . These monitor values appear in Table 1.

The class that holds monitor data has a simple format. There is one instance of an CorrelatorMonitorPoint for each of the items in Table 1 and they are held in an vector by the CC_Monitor class.

The polling rate of monitor data is done is 30 seconds. The ACC requests specific monitor values from the correlator controller returning the most recently obtained values. The polling process is managed at a higher level. Monitor values are retrieved by **getMonitorData()** and placed into the vector of CorrelatorMonitorPoints.

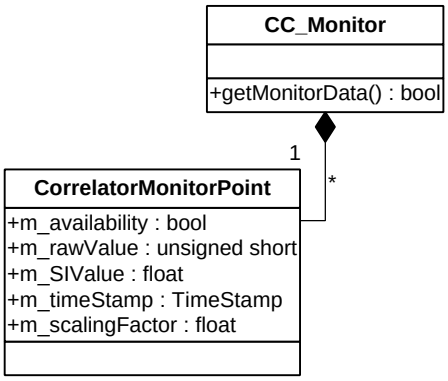

Figure 11 – CC_Monitor class

The data members for CorrelatorMonitorPoint are:

- **availability**

  Availability indicates that this point can be read. In normal operation this will be true.
- **rawValue**

  This is the actual raw value of the monitor point as a 16-bit unsigned integer.
- **SI Value**

  This is the raw value converted to SI units as a floating point value. These units are listed in Table 1.
- **scalingFactor**

  This provides a scaling factor to convert the raw value to SI units which are essentially the resolution values in Table 1.
- **timeStamp**

  This is the array time when the monitor point was polled.

10 Characteristic Information

Characteristic information refers to data that describe the correlator's operating parameters. These do not change for the correlator, but provide a simple mechanism for external devices such as the ACC to query the correlator controller and determine its basic operational information. These informational items are nominally strings unless otherwise noted. This information is held by the Characteristic Information package to which the CC_Manager has access and provides to the ACC on request.

Index	Item	Description
1	Cross-correlation tick time	Number of milliseconds of one correlator tick in cross-correlation mode. This value is 1.30172.
2	Auto-correlation tick time	Number of milliseconds of one correlator tick in auto-correlation mode. This value is 1.30172
3	Maximum data rate	The fastest rate at which data can spew out of the correlator in units of results/second. Raw lag results will be 32-bit integers while spectral points will be 32-bit floating-point numbers
4	Bandwidth modes	A list of the available bandwidth modes. These are 800 MHz and 100 MHz. An asterisk next to the mode indicates that a Xilinx FPGA personality image needs to be downloaded for this mode, e.g., “100MHz*”.
5	Mode Change Time	This describes the time (in seconds) it takes to perform a correlator mode change. For 800MHz mode, the value is 100 ms and for narrow band mode (100MHz) the value is 40 seconds.
6	Minimum Dump Time	This is the shortest dump time (in milliseconds) for the correlator hardware. This value is 96 milliseconds for synchronized short-term integrations and ~70 milliseconds for non-synchronized integrations
7	Computer Type	This is a simple string identifying the correlator computer type. It is “Motorola MV2700 MPC750”.
8	Monitor point list	A list of monitor points names, refer to Table 1 for details.

Table 4 – Test Correlator Characteristic Information

11 Diagnostics

Diagnostics refer to tests that can be run on the CHW to verify that it is operating correctly. Each test returns a success or failure flag with any errors being logged to the ErrorLogger (described below). In most cases the correlator computer must be reset after diagnostic tests complete. The execution of diagnostic tests are managed by the CC_Manager.

These diagnostic tests include:

- Read and write to scratch memory (up to 1024 bytes) on the system monitor, correlator control, and LTA cards. This test validates the serial links between the correlator computer and the system monitor card, and between the system monitor card and the other correlator cards.
- Check that the FIFO memory on VME interface card can be read. This ensures that the VME address is correct and the bus handshaking between the VME interface card and the VME computer is being done correctly.
- An additional diagnostic check is the check of the sampler threshold levels for the plus and minus bits. To do this check, a special correlation system mode is set, a short integration is made and the zero channel of the raw lags are sent to the ACC. This check is performed eight time, once for each bit for each of four samplers. This is the only test which does not require the correlator computer to be reset.

11.1 Error Logging

Error logging is accomplished by all objects having access to a single instance of an ErrorLogger object. As other software objects in the system encounter run-time errors, they log the errors with the ErrorLogger. The ErrorLogger maintains a circular buffer of ErrorLogEntrys enabling it to keep track of errors for a limited period of time. This circular buffer prevents unlimited memory usage and is adjustable in size. This error history can be retrieved by the ACC getting the entire circular buffer or just the errors since the last retrieval – referred to as "new errors".

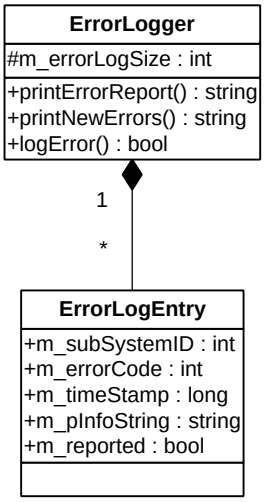

Figure 12 – ErrorLogger Class

When an error occurs, ErrorLogger::logError() creates an ErrorLogEntry object and places it into the m_errorHistory vector. The members variables of the ErrorLogEntry are:

- **M_subSystemID** – which is a uniquely generated identifier for each object that logs an error. This subsystem id is based on a name-lookup so that the object calling ErrorLogger::logError() supplies a characters string name which is then converted to an integer
- **m_errorCode** – this is the subsystem specific error number.
- **m_timeStamp** – the array time when the error is logged.
- **m_pInfoString** – this is a pointer to a string which contains optional error information.
- **m_reported** – a boolean which determines if this error item has been reported via ErrorLogger::printNewErrors() allowing one to only access new errors that have occurred since the previous request.

The ErrorLogger class members are:

- **m_errorHistory** – the vector of ErrorLogEntrys which is a circular buffer to avoid growing forever. Entries can be overwritten.
- **m_errorLogSize** – the size of the m_errorHistory array which can be varied allowing for resizing of the error history. The default size is 500 entries.
- **logError()** – the interface to log an error. Input parameters are the subsystem name, the error code and any optional information specified using a C printf variable argument string.
- **printNewErrors()** – the interface to obtain an error report of new errors. A new error is one which has not been previously obtained.
- **printErrorReport()** – allows one to obtain an error report of any group of errors. The group can be all errors, errors for a specific subsystem, and errors before or after a specific time.

The format for each line of the error report is:

`Sys: `*`Subsystem Name`*` Code: `*`Error Number`*` Time: `*`Array Time Additional info`*

12 Dynamic Model

The dynamic model for the test correlator control software utilizes a multi-threaded architecture which allows for concurrent execution of control, monitoring and data processing functions. This section details this functionality. Much of this analysis is based on the ADARTS (Ada Design Approach for Real Time Systems) model as outlined in [11]. The main goals of the ADARTS design methodology are twofold: to structure the system into concurrent tasks and to utilize the reuse of software through information hiding.

12.1 Multi-tasking Model

Table 5 depicts the various tasks envisioned for this design. There is a single instance of each task. The first two columns show the task name and purpose. The remaining columns pertain to the results of a rate monotonic analysis to determine if these tasks are schedulable which is further discussed later in § 12.2.

Task Name	Purpose	Execution Time (C_i) ms	Period (T_i) ms	Utilization[1]	Cumulative Utilization	Priority (decreasing)
ArrayTime	Process 48 ms Timing Events	0.5	48	1.04	1.04	10
GetRawLags	Obtain raw lags from CHW	46.0	70	65.71	66.76	20
DataProcessor [2]	Process raw lags to spectral results	10.0	70	14.29	81.04	30
DataCollector Communications[2]	Transfer spectral results to Data Collector	4.1	70	5.88	86.92	40
ACC Communications	Bi-directional communication to/from ACC	1.0	48	2.08	89.01	50
IntegrationManager [2]	Configure and control CHW	2.0	96	2.08	91.09	60
CC_Manager[2]	Coordinate ACC cmds. & responses to other tasks	5.0	96	5.21	96.30	70
MonitorManager	Update monitor data	5.0	2500	0.20	96.50	80

Table 5 – Task Description

Figure 13 shows a concurrency diagram, which follows the model used by [12], to provide a view of the tasks of the entire real-time system. It is an elaboration of a collaboration diagram, but shows the communications between the tasks. It is based on a Data Flow Diagram which tracks data as it moves through a system. One can envision commands coming in from the ACC and flowing through to various tasks for processing.

A task is identified by a parallelogram with lines between tasks. The lines represent communications with the arrows defining uni-directional messaging. A bold line implies communication between a task (or device driver) and some external interface. As many tasks can have multiple blocking mechanisms, e.g., VxWorks pipes, event triggers, and semaphores, the specific blocking mechanism is omitted except in the case where a single, specific trigger allows a task to execute. VxWorks pipes allow a specific task to wait for multiple messages of different sizes from other tasks. The priority of a

[1] Utilization is defined as the ratio of execution time to period, i.e., C_i/T_i

[2] Aperiodic task which is assumed to have a pseudo-period defined by the deadline time

task is show in each task parallelogram as P-1, P-2, or P-3 for high, medium, or low priority. A small box with two flags represents semaphores. Events are small rounded-edge boxes with a single flag.

There are four external systems shown that are outlined in bold, rounded rectangles: the ACC, the Data Collector, the 48 ms array-wide timing events and the correlator hardware. Communication flow is shown between these devices and the tasks that interface with them as wide arrows. The three communication mechanisms shown are TCP/IP, e.g., ACC to ACC Communication task, serial e.g., Serial task to the correlator hardware, and VME memory accesses e.g., correlator hardware to the Get Raw Lags task.

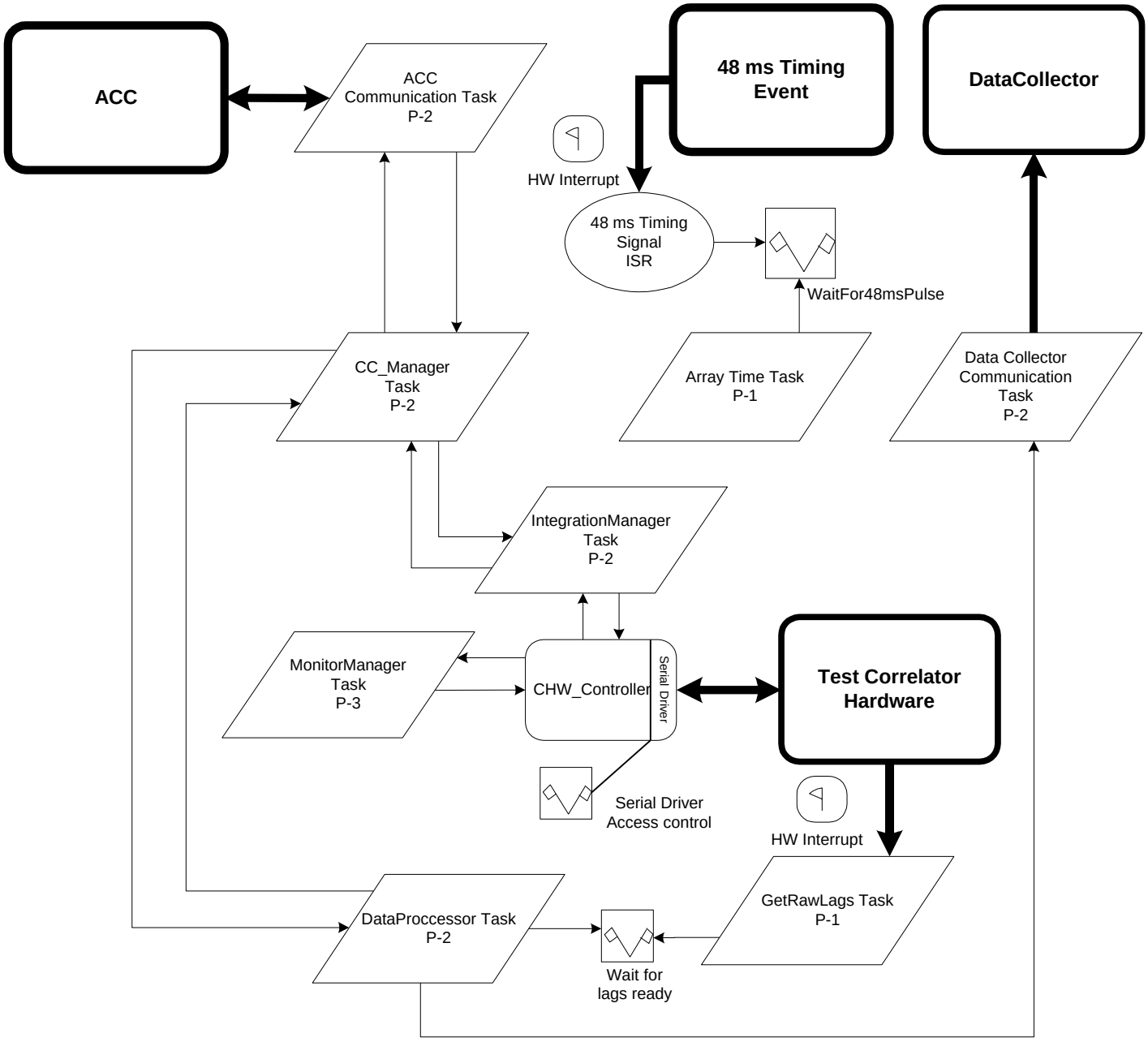

Figure 13 – Concurrency Diagram

These tasks are implemented as classes which provide a wrapper around the VxWorks RTOS functions and enable them to contain instances of objects described in the preceding sections. This approach of encapsulation eliminates the need for global variables, allows initialization of task information in the task object's constructor and facilitates the modularity of tasks. Also many of the tasks implement the "manager" class functions described in earlier sections so one can see the parallels between, for example, the CC_Manager and the CC_ManagerTask. A description of each task follows.

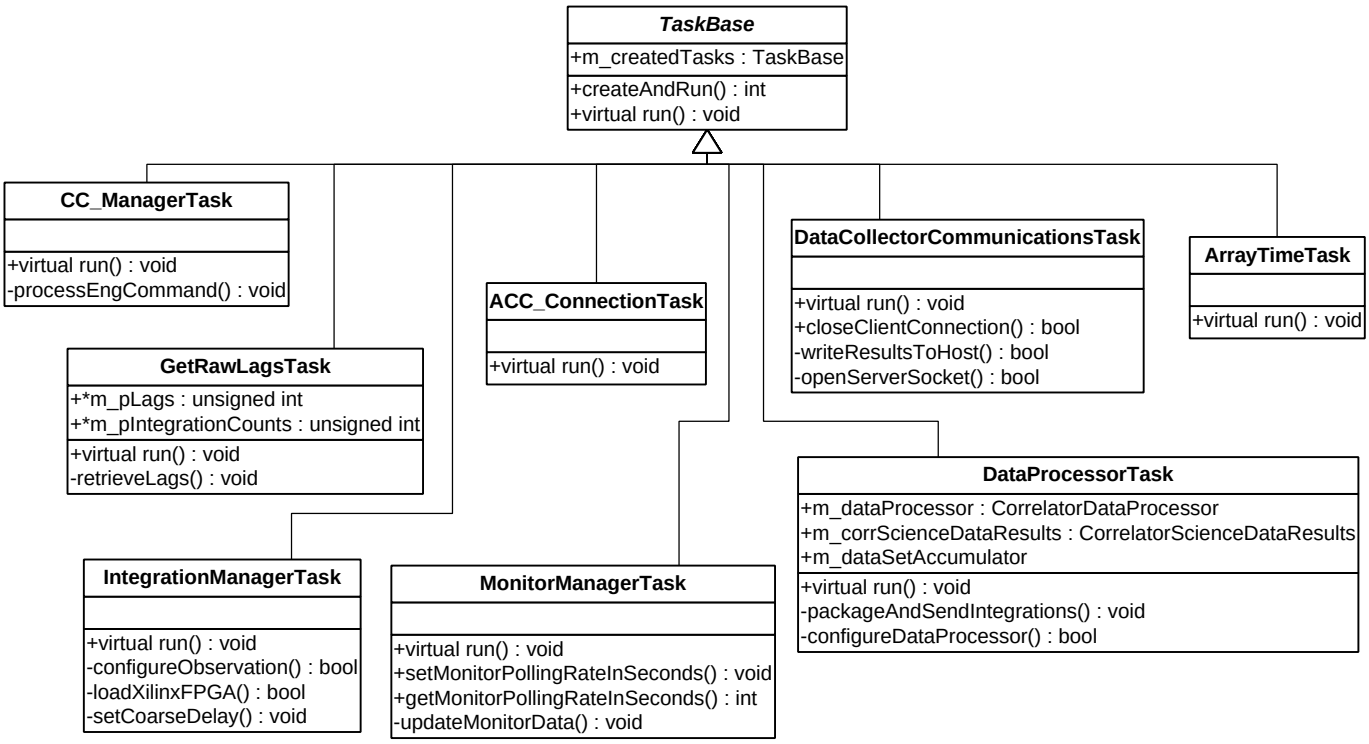

Figure 14 – Class Task Hierarchy

12.1.1 TaskBase

All tasks are derived from this abstract base class which has the pure virtual function, **run()**. TaskBase contains a list of tasks objects which have been instantiated and started via the VxWorks task spawning function **taskSpawn()**. It utilizes a static member function to pass to **taskSpawn()** to deal with the differences of C and C++ when passing function pointers. A one-time instantiation of a special class, StartTasks, constructs the TaskBase-derived classes and starts their execution. StartTasks also performs some system-wide initializations.

12.1.2 ACC Communication Task

This task receives commands from the ACC via a TCP/IP connection which it manages. As commands arrive, they are converted to EngCommand-derived objects (see § 13) and, if valid, sent to the CC_Manager task for processing. Responses to information requests are routed through this task to the ACC.

12.1.3 CC_Manager Task

This task serves many roles, it:

- Acts as delayed handler for the ACC Communication task by being a distribution point for command processing and provides a conduit for command responses.
- Receives EngCommand-derived objects. This EngCommand object is sent to the appropriate task for further processing, e.g., observation configuration, retrieve monitor data, etc.
- Assists in the coordination of the starting and stopping of observations among multiple tasks, namely the Integration Manager and Data Processor tasks.
- Launches diagnostic tests. Request for diagnostic tests arrive from the ACC with the CC_Manager task executing the tests and returning the test results to the ACC.
- Handles characteristic information requests and returns ErrorLogger report requests.

12.1.4 Integration Manager Task

The Integration Manager task performs configuration and execution of observations. It receives configuration commands from the ACC via the CC_Manager task which it uses to configure the correlator hardware using the CHW_Controller including downloading of Xilinx FPGA images for 100 MHz modes. These images reside on a remote file system accessible either through NFS or FTP. If there is a network failure and the IntegrationManagerTask can't download the file, a error is raised and the correlator controller is set to an IDLE state.

The IntegrationManagerTask also receives start and stop observing messages from the CC_Manager based on start and stop observing commands from the ACC. Thus it acts as a delayed handler of commands for the ACC Communications task.

12.1.5 Get Raw Lags Task

This high-priority task is responsible for retrieving the raw lags from the correlator. Each time a set of 1024 raw lags (there are 64 sets per dump per bin) is ready to transfer to the correlator computer, an interrupt handler is triggered. This interrupt handler unblocks a semaphore which the Get Raw Lags task waits on allowing it to perform a DMA transfer to an internal buffer. Once all of the lags are retrieved for the current dump, a pointer to the raw lags is sent to the Data Processor task for processing.

12.1.6 Data Processor Task

The Data Processor task coordinates the processing lags to spectra. It does this in many steps:

- It waits for a pointer to the raw lags from Get Raw Lags task.
- The lags are then converted to spectra using its configured CorrelatorDataProcessor object.
- If there are multiple dumps in an integration, the spectral results for the current dump are accumulated with previously accumulated results.
- Once the integration time, i.e., the number of dumps, has been reached the accumulated results are package into a CorrelatorScienceDataResults object and sent to the Data Collector Communication task for transmission to the Data Collector.

This task has a hard deadline which it must meet before the next set of lags arrive from the GetRawLags task.

12.1.7 Data Collector Communication Task

This task waits for a pointer to a CorrelatorScienceDataResults object sent by the Data Processor Task and transmits it to the Data Collector via TCP/IP. This tasks also monitors the link to the Data Collector to ensure that it is valid, flagging any errors if the connection is down.

Similar to the Data Processor task, this task has a hard deadline which must be met before the next set of spectral results arrive from the Data Processor task. Buffering of multiple sets of results allow for delays due to Ethernet loads or for temporary, short-term breaks in network connections. This task contributes to the minimum correlator dump time, but its impact is not as severe as the Data Processor task and can be simply calculated by the block size times the data rate. Note that this does not account for non-determinacy with *arrival* of the data to the Data Collector due to Ethernet loads.

12.1.8 Monitor Manager Task

This task requests all CHW monitor data on a periodic basis and populates a vector of CorrelatorMonitorPoint objects which are then available for delivery to the ACC.

12.1.9 Array Time Task

The Array Time task is responsible for tracking array time in the correlator computer. Array time is set by a command from the ACC upon startup. After that, timing event pulses are accessible via a digital I/O card on the VME bus. These

pulses generate interrupts which are then counted in the Array Time task updating the correlator computer's internal array time clock. This array time is made available to other software components in the correlator computer.

The Array Time task also plays a role in synchronization. It can receive messages to notify other tasks to execute some functionality, e.g., start observing, at a given timing event.

12.2 Correlator Computer Timing Analysis

Timing deadlines identify the "hard real-time" aspects of a real-time system. For the test correlator and the MVME-2700 CPU these critical deadlines are few.

One critical deadline is the data processing. The raw lag results must be extracted from the LTA, converted to spectral results, integrated and queued to the Data Collector Communication task for delivery to the Data Collector within the dump time. This process defines the minimum dump time.

When the LTA has completed a dump, it sends an interrupt to the VME computer signaling the correlator controller to transfer 64 4096-byte blocks. The speed of this DMA transfer is controlled by the VME interface card, which is approximately 6 MB/sec leading to a duration of ~46 milliseconds.

Next the raw lags are sent to the Data Processor task which converts them to spectral results. According to [3], processing of 1K lags takes about 550 KFLOPS and the MVME-2700 can execute about 170 MFLOPS which results in a duration of ~3 – 25 milliseconds depending on the size of the FFT and the number of FFTs to perform for a given correlator mode.

The other time factor is the transmission time of a spectral results block from the correlator computer to the Data Collector. The spectral results block has a 24-byte header plus 1024 single-precision complex floating point values for a total of 8216 bytes and using a 100 Mbit Ethernet link the transmission time will be less than 1 millisecond, although Ethernet can be highly variable as it is non-deterministic. Therefore the entire duration from the time the correlator controller starts extracting lags to when they arrive at the Data Collector computer should take about 70 milliseconds.

It is important to note that these time estimates are for one correlator bin. If multiple bins are used, then these durations for DMA transfers, data processing and transmission must be multiplied by the number of bins' worth of data, i.e., either 2 or 4.

Columns 2 – 6 of Table 5 show the results of a rate monotonic analysis to determine the schedulability of these tasks using the example of processing 1K lags in an auto-correlation mode, correlator mode 3. I followed the method of RMA outlined in [13]. Columns 4 and 5 are the most important. The utilization is the percentage of CPU time taken by a specific task and is simply defined as Utilization = (execution time / period) * 100%. The cumulative utilization shows the sum of utilization of the current task and higher priority tasks. A bold dividing line in Table 5 after the Data Collector Communications task shows all of the tasks in use during the correlation process. From this analysis, one adjusts the period values in column 4 to define the minimum correlator dump time such that the cumulative utilization is less than 100%. In the case of the example shown in Table 5, since the cumulative utilization is less than 100%, all tasks are schedulable.

One caveat with this analysis is that RTOS system overhead is not accounted for, e.g., task context switching times. If we assume a reasonable value of 1 ms per second, this is a negligible load of 0.1% (VxWorks advertises a value much shorter than 1 ms/second).

Using this analysis, the theoretical minimum dump times for each of the correlator modes appear in Table 6.

Correlator Mode	Minimum Dump Time (ms)
1	60
2	60
3	70
4	100
5	100
6	100
7	100

Table 6 – Theoretical Minimum Dump Times

To ensure that time-critical functions meet their deadlines, two schemes are utilized. First, deferred handling allows for time-critical functions to occur at high priorities and then pass off the further processing to lower priority tasks and secondly, buffering is used. For example, the GetRawLags task obtains the raw lags at a high priority. Once it has the lags, they are sent to a medium priority task, DataProcessorTask, for conversion to spectral results via a message queue which can hold pointers to many raw lag sets.

Some tests were run to determine actual data processing and transfer times. These total times are shown in Table 7. Comparing them to the theoretical values shows that they are within reason and validates the RMA model.

Correlator Mode	# of Pol'n Prod. per Dump	Processing Time per Pol'n Prod.	Processing Time per Dump	DMA Time	Ethernet xfer. Time	Total Time
1	4	2.5 ms	10 ms	46 ms	0.3 ms	58 ms
2	2	5.1 ms	10.2	46 ms	0.4 ms	59 ms
3	4	4.2 ms	16.8	46 ms	1.0 ms	69 ms
4	4	12.2 ms	48.8	46 ms	1.0 ms	102 ms
5	2	19.1 ms	38.2	46 ms	1.5 ms	98 ms
6	2	19.1 ms	38.2	46 ms	1.5 ms	98 ms
7	2	19.1 ms	38.2	46 ms	1.5 ms	98 ms

Table 7 – Empirical Data Processing Times

Another critical deadline is to ensure that the CHW starts an integration on a specific timing event. This was discussed in § 5.1. As long as the ACC provides the start observing command at least 96 ms (2 timing events) beforehand, there should be no problem meeting this deadline. If this deadline is not met, then an error is logged.

12.3 Blocking and Deadlocks

The correlator computer software must deal with hardware and software blocking. Hardware blocking occurs at three points which specify the time-critical blocking for the correlator computer (deadline times specify the timeout expiration value):

- Waiting for the 48 ms timing events, deadline = 48 ms
- Waiting for the raw lags from the LTA, deadline = dump time
- Waiting for a response on the serial port connection to the CHW when requesting data, deadline = 0.5 second

All three of these are monitored via watchdogs which raise alarms if they have timed out past their deadline time. VxWorks have watchdog functions that run at interrupt-level and are specifically designed for this type of application.

Software blocking occurs when a task waits for messages from other tasks. All tasks in this design wait indefinitely for messages avoiding problems when a message doesn't arrive in a given time. Problems only arise if a task cannot meet its

processing deadline before a new message arrives. RMA and timing tests are used ensure that each task can meet its processing deadline.

Deadlocks occur when access to shared resources are not properly guarded. The only shared resource that falls into this category is the serial connection between the correlator computer and the CHW and when two tasks want to use this port. A VxWorks mutual exclusion semaphore which can correctly protect against priority inversion is used to arbitrate the access to the serial port.

12.4 Run-time State Information

The correlator software contains state information corresponding to its execution. A simple class, CCC_State, tracks the state information of the correlator device and allows (or prevents) transitions from one state to the next. Refer to the state diagram in Figure 15 for the following discussions of the correlator computer state machine.

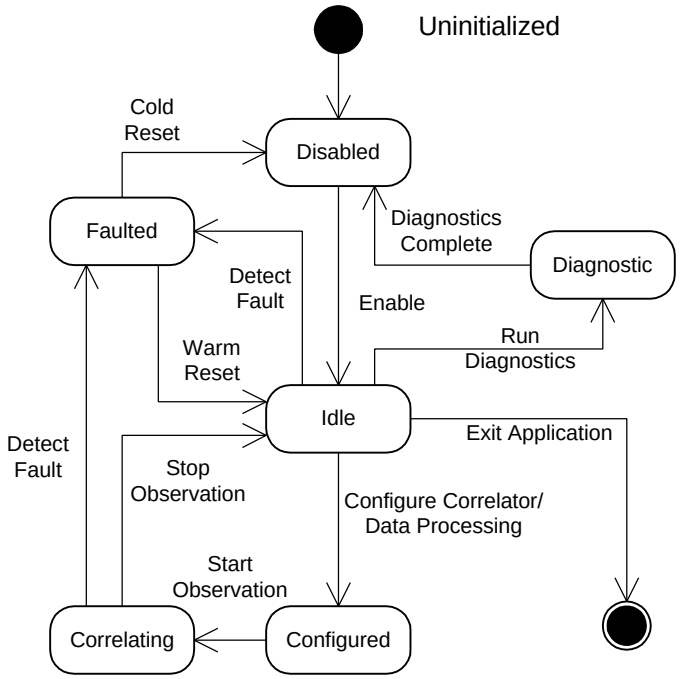

Figure 15 – Correlator Computer System State Diagram

12.4.1 Allowed States

- **DISABLED** – The equipment is in power-off or rebooting state.
- **IDLE** – The correlator computer is ready to accept commands and is basically idle. This is the “ready” state.
- **CORRELATING** – The correlator computer is correlating, i.e., integrating, processing lags and sending them to the Data Collector
- **CONFIGURED** – The correlator computer has configured itself and the CHW for an observation.
- **FAULTED** – A serious unexpected run-time error has occurred requiring operator intervention. Note that this is not the same as an error returned by some task or software function call.
- **DIAGNOSTIC** – This is a special state which allows for diagnostic tests to be run on the correlator computer. Normal operational commands are disabled.

12.4.2 Allowed State Transitions

- **ENABLE** – move into the **IDLE** state with a power-on reset or from a cold boot as in the case of a transition from the **DIAGNOSTIC** state. Exiting action of **DISABLED** is to run initialization functions.
- **RUN DIAGNOSTICS** – move to the **DIAGNOSTIC** state from the disabled state when unable to move to the idle state due to some fault. This transition from the **IDLE** state can also occur under command from the ACC
- **DIAGNOSTICS COMPLETE** – this transition to the **DISABLED** state always performs a power up reset.
- **CONFIGURE CORRELATOR & DATA PROCESSING** – this transition occurs as the result of a command from the ACC to configure the correlator controller for correlation.
- **START OBSERVATION** – this transition to the **CORRELATING** state occurs when an observation starts.
- **STOP OBSERVATION** – this transition occurs upon exiting the **CORRELATING** state and returning to the **IDLE** state.
- **DETECT FAULT** – this transition to the **FAULTED** state occurs when any serious error (alarm) is encountered requiring a reset.
- **COLD RESET** – this transition to the **FAULTED** state is due to a fatal error and requires full initialization of the correlator computer and CHW.
- **WARM RESET** – this transition allows for a return to the **IDLE** state without resetting physical hardware.

12.4.3 Tracking the Correlator Computer State

The allowed states and transitions are managed by a simple, globally accessible singleton class, CCC_State. CCC_State ensures that transitions are valid before they are made and provides an error reporting mechanism if they aren't allowed. CCC_State can provide current state information to the ACC upon request.

13 Command Processing

Now that the static and dynamic designs have be presented, it is time to discuss the actual commands to and responses from the ACC. This design is an interim one which will be replaced by a CORBA interface that will be utilized throughout the test interferometer control software.

13.1 Command Processing Description

Command processing is performed at two levels, first by the ACC_Communications task and then by the CC_Manager task. The command travels from the ACC as an ASCII byte stream to the ACC_Communications task where an Eng-Command-derived object is constructed using a static member function that parses the ASCII byte stream. This Eng-Command object is then passed to the CC_Manager for distribution to the correct task for execution.

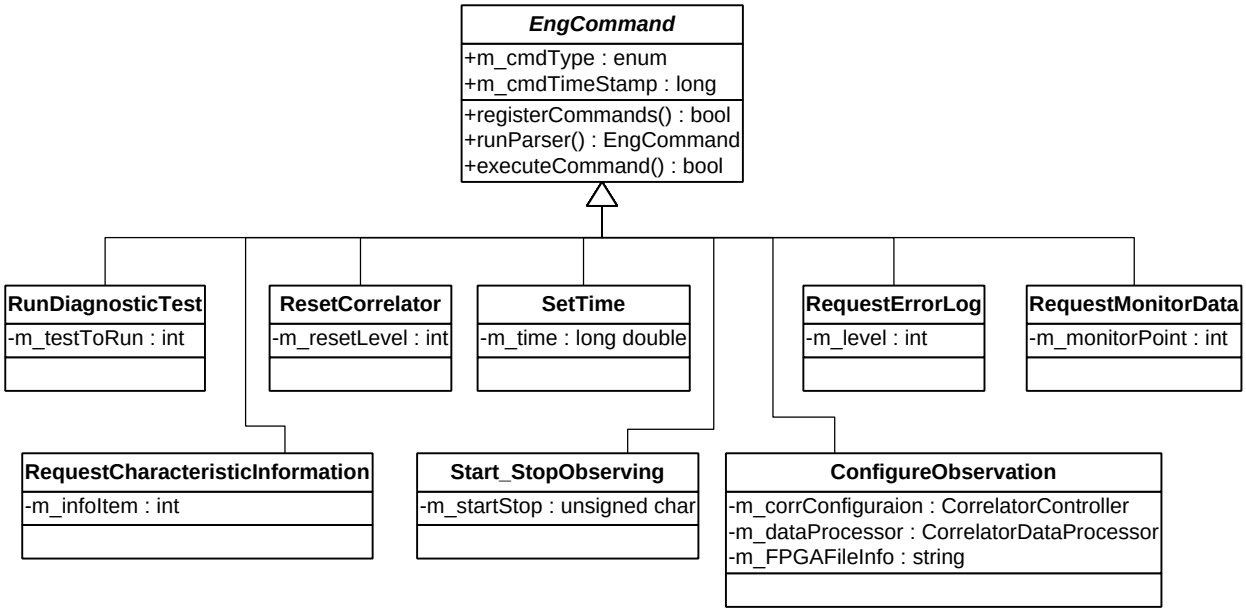

Figure 16 – EngCommand base & derived classes

Figure 16 shows the command hierarchy with each specific command derived from the base class EngCommand. Commands can have request and response information. Request data correspond to parameters needed by a given command. Response data include an acknowledgement that the command was correctly (or not) processed and can include requested information. Any error information is recorded in the ErrorLogger as described earlier. A description of the members of EngCommand are:

- **m_cmdType**

  An enum which defines what command this is. This allows for correctly constructing the derived command from an ASCII byte stream in runParser().

- **m_cmdTimeStamp**

  This is the time tag of when the command was received by the ACC_Communications task.

- **runParser()**

  This static function converts the command from an ASCII byte stream format to a EngCommand-derived object. It returns a reference to an EngCommand-derived object to be passed onto the CC_Manager task.

- **executeCommand()**

  Provides a mechanism for the CC_Manager task to perform a syntax check on the command and sets the specific child object's data members.

The next section lists the specific commands. The format of the description for each command lists its ID, then a brief description of what the command does, any optional request data sent to the correlator computer, optional response data, and finally a description of specific errors.

13.1.1 RESET_CORRELATOR

- **Description:** This command resets the correlator computer and/or the CHW. Any initialization sequences are executed for the given device. The CHW reset is a command which resets the microprocessors on the correlator cards.
- **Request Data:**

  byte:

  1 reset CHW ("warm boot")

  2 reset correlator computer ("cold boot" which reloads the VxWorks OS, test correlator application and starts execution)

  3 reset correlator computer & CHW

- **Response Data:** OKAY/ERROR
  Errors
  - The CHW is unresponsive after reset. If so, then human intervention is required.

13.1.2 CONFIGURE_OBSERVATION

Description: Configure correlator and data processor for an observation. This allows a one-shot configuration of all the parameters required to set up the correlator to commence observing.
- The dump time is an integer representing the number of 1.31072 ms correlator ticks
- The integration duration is in correlator dumps.
- The integration duration, i.e., the number of integrations to perform.
- The bin switch time is in units of 48 ms timing events as bin switches occur on these boundaries

- **Request Data:**
  - dump time: uint16 1 - 2^{15}-1
  - integration time: uint32 1 – 2^{32}-1
  - integration duration: int32 -1 - 2^{31}-1 (-1 means forever)
  - number of bins byte 1, 2 or 4
  - bin switch time uint16 0 - 2^{15}-1
  - correlator mode id: byte 1 – 7 (see Table 2 for details)

  - FFT byte 0 – 1 (false/true)
  - Hanning smoothing: byte 0 – 1 (false/true)
  - van Vleck correction: byte 0 – 1 (false/true)
  - spectral averaging: byte 0 – 1 (false/true)
  - starting spectral point: uint16 0 – N - 1
  - stopping spectral point: uint16 0 – N - 1
  - decimation of spectra: uint16 0 – 10
  - delay model coefficients
    - Antenna ID byte 0 – 1
    - Coefficients float[5] -FLT_MAX – +FLT_MAX
- **Response Data:** OKAY/ERROR
  - Errors
    - Invalid values for numeric parameters.

13.1.3 START_STOP_OBSERVING

Description: Starts or stops the correlator integrations – recall that configuration of an observation specifies the stopping time of the correlator integrations. Short-term integrations can be synchronized to the 48 ms timing events.

- **Request Data:**
  - Start/stop type: byte 1 start observing – non-synchronized
    2 start observing with short-term integrations synchronized to the 48 ms TE
    3 stop observing at end of current integration
    4 abort observing immediately returning partial results
  - Start/stop time long long Array time at which the integration is to start/stop.
- **Response Data:** OKAY/ERROR
  - Errors
    - Correlator not configured for an observation (start observing error)
    - Correlator state is not observing (stop observing error)
    - Start (or stop) time has passed – in this case, integrations still start (or stop)

13.1.4 REQUEST_ERROR_LOG

Description: As errors are recorded by the correlator controller and/or CDP execution, they are preserved in an error history log. The contents of this log may be retrieved via this command.

- **Request Data:**
  - Error type: byte 1 = get new errors, 2 = get all errors
- **Response Data:** OKAY/ERROR. If OKAY, then return the requested error messages according to desired severity

13.1.5 REQUEST_MONITOR_DATA

Description: Get the current monitor data for the correlator. Note that this command is only implemented for testing purposes. It requests monitor data whereas in the actual test interferometer control software, monitor data will be "pushed" via CORBA functionality.

- **Request Data:** Monitor point index 1 – 8 (see Table 1 for details)
- **Response Data:** OKAY/ERROR. If OKAY then the requested monitor point is returned. Each value is a floating point number in the appropriate units.

13.1.6 REQUEST_CHARACTERISTIC_INFORMATION

Description: Obtain correlator characteristic information.

- **Request Data:** Data index, 0 – 8 where 0 means return all following points, 1 – 8 acts as an index into the following list of characteristic information and allows one to request characteristic information for **one** specific item (see Table 4 for details):

  1 - Cross-correlation tick time
  2 - Auto-correlation tick time
  3 - Maximum data rate
  4 - Bandwidth modes
  5 - Mode Change Time
  6 - Minimum Dump Time
  7 - Computer Type
  8 - Monitor point list
- **Response Data:** OKAY/ERROR. If OKAY then the appropriate characteristic information in is returned.
  - Errors
    - Invalid characteristic information index

13.1.7 RUN_DIAGNOSTIC_TEST

Description: This command runs diagnostic tests on the CHW.

- **Request Data:**
  - Diagnostic test uint16: 0 Run all tests
    1 Run CHW check read/write memory test on correlator cards.
    2 Run read lag results test.
    3 Run sampler threshold check.
- **Response Data:** OKAY/ERROR
  - Errors
    - The test result failed. A request of the log error provides more details of the failure.

13.2 Engineering Test Interface

The schedule for the ALMA test correlator software is such that it will be available before TICS [1] to which it interfaces. This will allow the test correlator hardware and software to be used for lab tests with other electronic components. In this capacity, the test correlator will run in a "stand alone" fashion.

The Test Correlator Engineering User Interface application (TC_EngUI) allows testing of the test correlator control software by simulating the ACC and Data Collector computers. The TC_EngUI allows for testing and debugging of functionality described in this document including command processing, timing issues, data processing, load testing and general execution correctness. TC_EngUI also allows for testing of the test correlator hardware to ensure that modifications made for the two-antenna ALMA test interferometer are correct.

A typical testing environment is to have an external noise generator be input to the samplers and integrations can be controlled to collect lag results, process them to spectra and viewed in near real-time. This setup comprises a system test for the correlator control software. The main interface is designed around the commands outlined in §13.1. Screen shots for correlator configuration and control appear below.

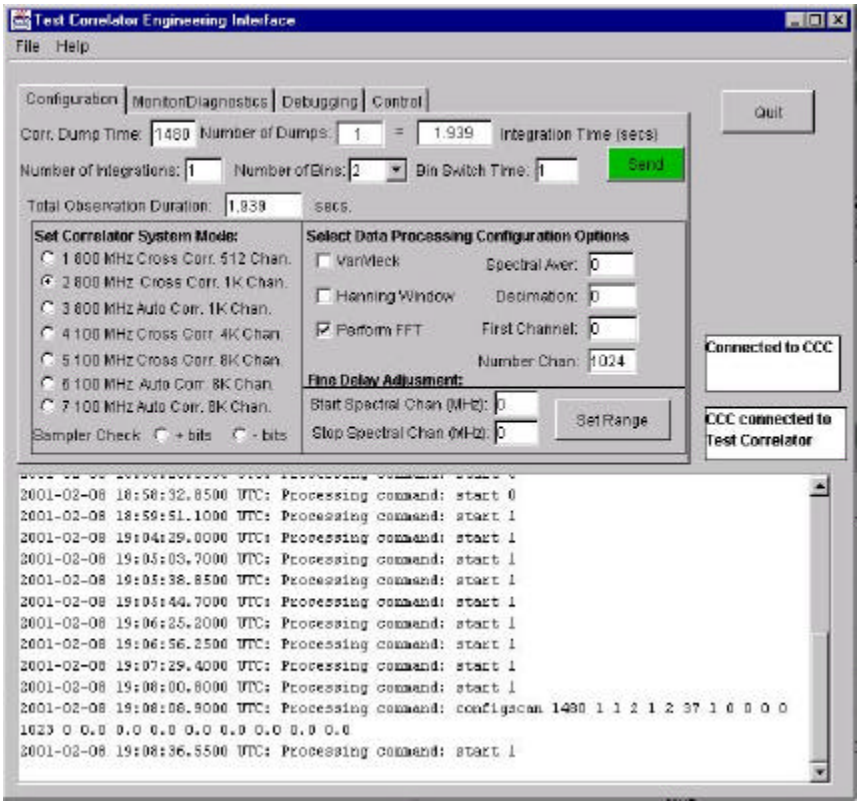

Figure 17 – Correlator Configuration Screen

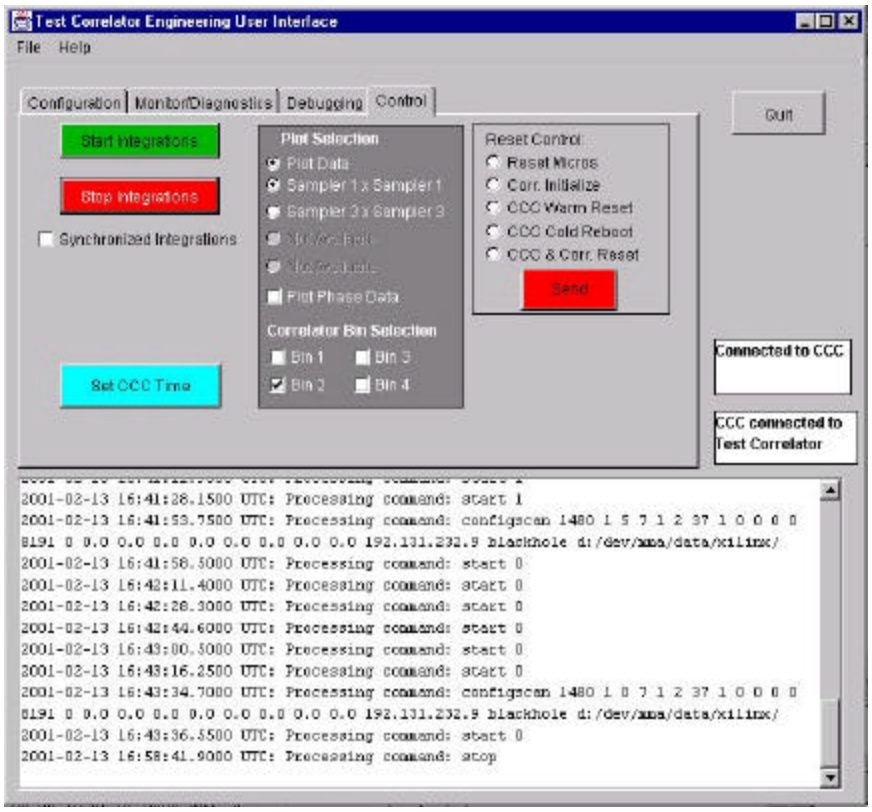

Figure 18 – Correlator Control Screen

14 Glossary of Classes

This glossary lists the classes used in this design with a brief description and their relationships to other classes. This relationship includes the multiplicity. Figure 19 shows the class hierarchy graphically.

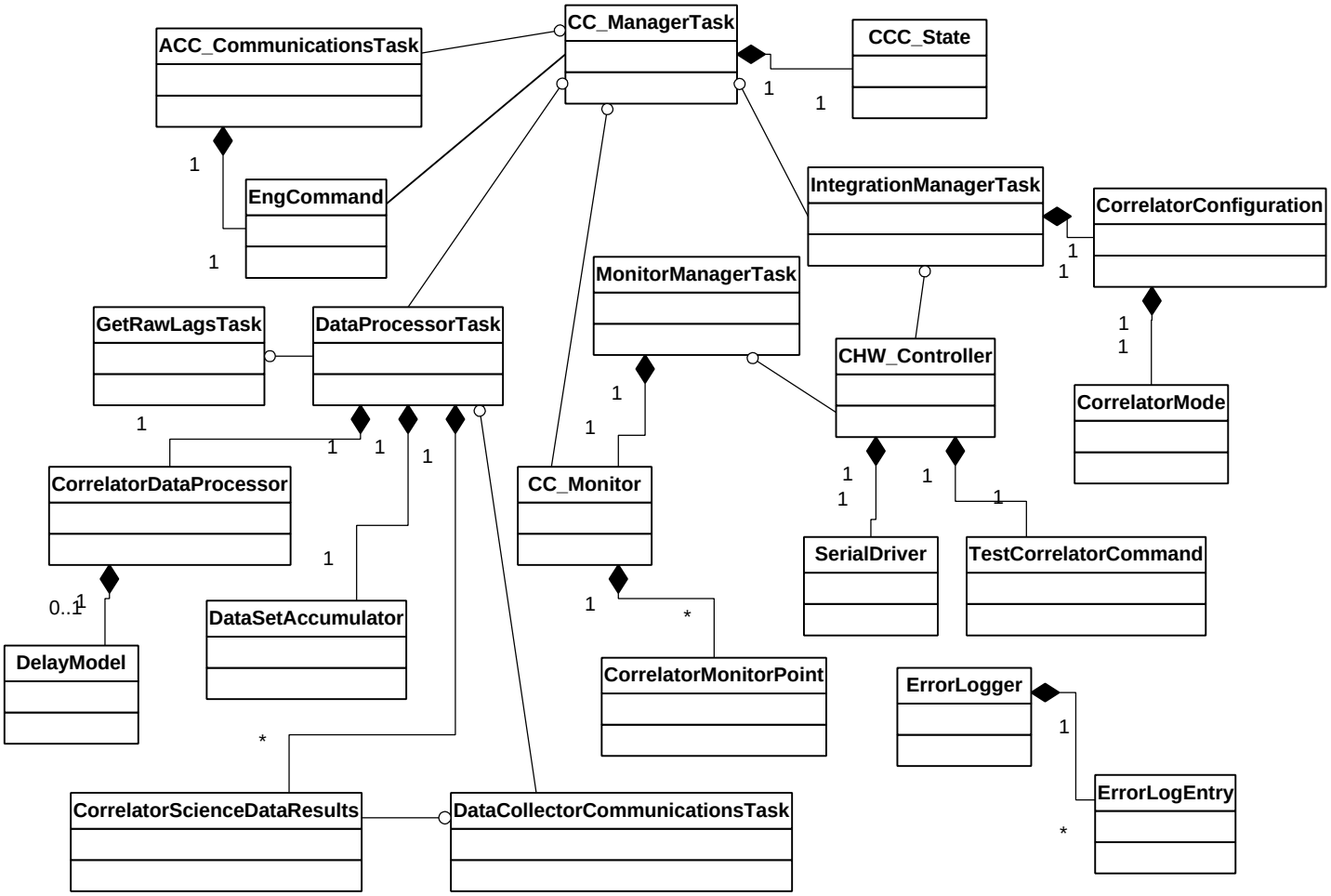

Figure 19 – ALMA Test Correlator Class Hierarchy

14.1 ACC_CommunicationsTask

ACC_CommunicationsTask encapsulates communications between the correlator computer and the ACC and the Data Collector.

ACC_CommunicationsTask communicates with:

- CC_ManagerTask

ACC_CommunicationsTask uses:

- EngCommand to parse flatten ASCII commands and construct EngCommand-derived objects.

14.2 CC_ManagerTask

CC_ManagerTask acts as a central distribution point among other tasks for command execution. Functions that it doesn't send to other tasks are execution of diagnostics, reporting of error logs and providing characteristic information.

CC_ManagerTask communicates with:

- ACC_CommunicationsTask

- DataProcessorTask
- IntegrationManagerTask

CC_ManagerTask uses one instance of:

- CCC_State
- EngCommand-derived object
- CC_Monitor to retrieve current monitor points

14.3 CC_Monitor

CC_Monitor holds all the current monitor values.

CC_Monitor is used by:

- MonitorManagerTask to store the current monitor values.
- CC_ManagerTask to retrieve monitor values to send to the ACC.

CC_Monitor uses:

- CorrelatorMonitorPoint to hold an individual monitor, i.e., there are multiple instances of CorrelatorMonitorPoint.

14.4 CCC_State

CCC_State tracks the state of the CCC and defines valid state transitions

CCC_State is used by:

- CC_ManagerTask

CCC_State uses:

- No one

14.5 CorrelatorConfiguration

CorrelatorConfiguration encapsulates observation configuration information.

CorrelatorConfiguration is used by:

- IntegrationManagerTask to manage integrations for the current observation. A IntegrationManager can have 1 CorrelatorConfiguration.
- CHW_Controller which controls the CHW for the current observation. A CHW_Controller can have 1 CorrelatorConfiguration which is a copy of IntegrationManagerTask's CorrelatorConfiguration.

CorrelatorConfiguration uses:

- CorrelatorMode

14.6 CHW_Controller

CHW_Controller is a singleton object which encapsulates control of the CHW and provides an interface to the CHW.

CHW_Controller is used by:

- MonitorManagerTask
- IntegrationManagerTask

CHW_Controller uses:

- CorrelatorConfiguration for the current CHW configuration defined by the IntegrationManager.
- TestCorrelatorCommand for commands to send to the CHW.
- SerialDriver for sending TestCorrelatorCommands to the CHW.

14.7 CorrelatorDataProcessor

CorrelatorDataProcessor processes correlator lag results for each dump.

CorrelatorDataProcessor is used by:

- DataProcessorTask contains the single instance of the CorrelatorDataProcessor for processing lags.

CorrelatorDataProcessor uses:

- CorrelatorMode defining the current correlator mode configuration.
- DelayModel A CorrelatorDataProcessor can have 0 or 1 DelayModel objects to apply fine delays to each spectral data set.

14.8 CorrelatorMode

CorrelatorMode encapsulates the mode information for a current CHW configuration which includes bandwidth, polarization and the number of lags.

CorrelatorMode is used by:

- CorrelatorConfiguration contains the single instance of the CorrelatorMode
- CorrelatorDataProcessor contains a single instance of the current CorrelatorMode

CorrelatorMode uses:

- No one.

14.9 CorrelatorMonitorPoint

CorrelatorMonitorPoint holds an individual monitor value.

CorrelatorMonitorPoint is used by:

- CC_Monitor

CC_Monitor uses:

- No one.

14.10 CorrelatorScienceDataResults

CorrelatorScienceDataResults encapsulates the spectral results of the correlator after processing of the lags and includes header information identifying the integration sent to the Data Collector. The actual data structure is shown in Table 3.

CorrelatorScienceDataResults is used by:

- DataProcessorTask contains a linked list of CorrelatorScienceDataResults objects, one for each spectral set per integration.
- DataCollectorCommunicationsTask receives a reference to each CorrelatorScienceDataResults object from DataProcessorTask for transfer to the Data Collector.

CorrelatorScienceDataResults uses:

- No one.

14.11 DataCollectorCommunicationsTask

DataCollectorCommunicationsTask manages the transmission of spectral data sets to the Data Collector.

DataCollectorCommunicationsTask communicates with:

- DataProcessorTask from which it receives the spectral data sets.

DataCollectorCommunicationsTask uses:

- CorrelatorScienceDataResults which holds the spectral data set for a correlator integration.

14.12 DataProcessorTask

DataProcessorTask coordinates the processing of correlator lag results for each dump and transferring the spectral results for each integration to the DataCollectorTask.

DataProcessorTask communicates with:

- CC_ManagerTask for configuration and control
- GetRawLagsTask to obtain the CHW's raw lags
- DataCollectorCommunicationsTask to transfer spectral data sets to the Data Collector.

DataProcessorTask uses:

- CorrelatorDataProcessor encapsulates the functionality of spectral processing of lags.
- CorrelatorScienceDataResults encapsulates the spectral data plus header to be transmitted to the Data Collector.
- DataSetAccumulator to sum spectra from multiple dumps.

14.13 DataSetAccumulator

DataSetAccumulator sums spectra from each dump of an integration into an integration accumulator. When the integration completes, the accumulated spectra are sent to the Data Collector

DataSetAccumlator is used by:

- DataProcessorTask

DataSetAccumulator uses:

- No one.

14.14 DelayModel

DelayModel encapsulates geometric delay model information and evaluation for an antenna.

DelayModel is used by:

- CorrelatorDataProcessor can have 0-1 DelayModel objects.

DelayModel uses:

- No one.

14.15 EngCommand

EngCommand is a base class which encapsulates commands from the ACC (the derived classes are not shown in Figure 19).

EngCommand is used by:

- ACC_CommunicationsTask which contains 0 or more EngCommand -derived classes.
- CC_ManagerTask to execute the EngCommand-derived command.

EngCommand uses:

- No one.

14.16 ErrorLogger

ErrorLogger is a globally-accessible singleton class which records run-time errors and provides access to the ACC.

ErrorLogger is used by:

- All objects which can flag errors.

ErrorLogger is uses:

- ErrorLogEntry holds individual errors.

14.17 GetRawLagsTask

Obtains the raw correlator lags via DMA transfer.

GetRawLagsTask communicates with:

- DataProcessorTask notifying it when raw lags are available for processing.
- An interrupt service routine which notifies GetRawLagsTask when lags are ready to be transferred from the CHW.

GetRawLagsTask uses:

- No one.

14.18 MonitorManagerTask

MonitorManagerTask manages the monitor points.

MonitorManagerTask communicates:

- No one.

MonitorManagerTask uses:

- CC_Monitor to hold the current monitor values.
- CHW_Controller for interfacing to the CHW

14.19 IntegrationManagerTask

IntegrationManagerTask controls and configures CHW for observations.

IntegrationManagerTask communicates with:

- CC_ManagerTask

IntegrationManagerTask uses:

- CorrelatorConfiguration for the current CHW configuration.
- CHW_Controller for interfacing to the CHW

14.20 SerialDriver

SerialDriver is a singleton object which provides an interface to the CC's serial port for communication with the CHW.

SerialDriver is used by:

- CHW_Controller

SerialDriver uses:

- No one

14.21 TestCorrelatorCommand

TestCorrelatorCommand encapsulates the CHW commands for control of the CHW. See [14] for a complete listing of these commands.

TestCorrelatorCommand used by:

- CHW_Controller

TestCorrelatorCommand uses:

- No one

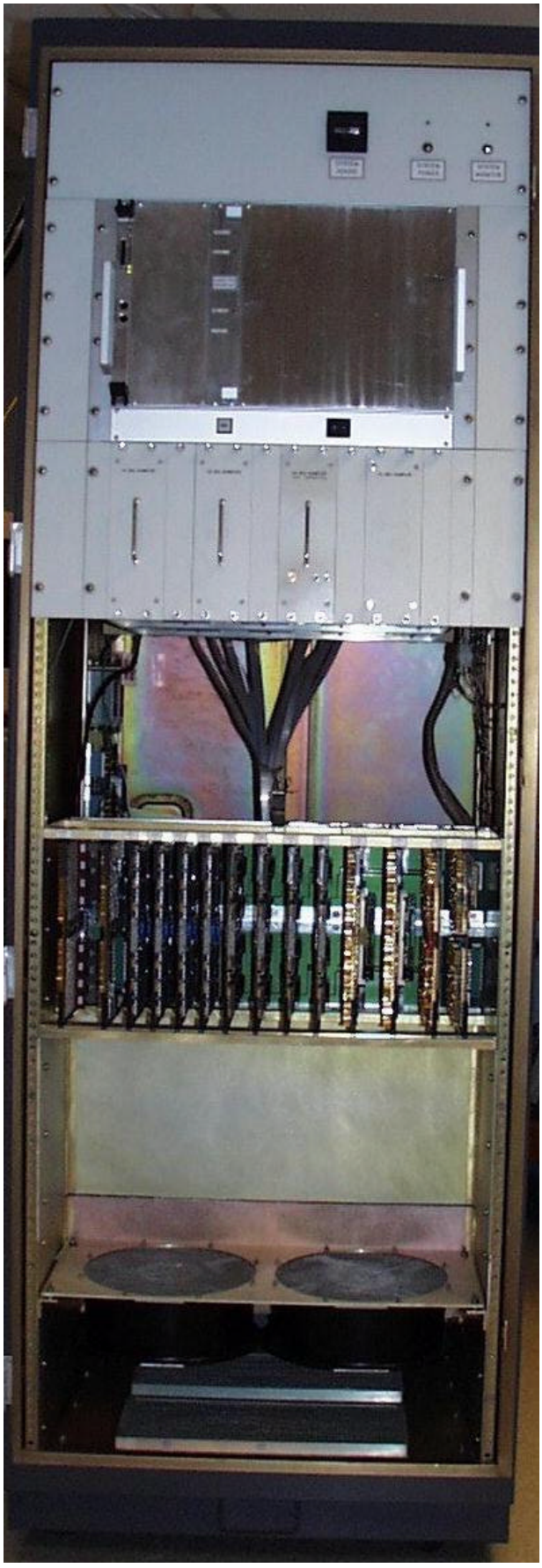

Figure 20 – Test Correlator Front View

15 References

[1] Glendenning, B.E., et. al. *Test Interferometer Control Software Design Concept DRAFT: 2001-02-15* http://www.mma.nrao.edu/development/computing/docs/joint/draft/TICS_Design.pdf, (2001)

[2] Schwarz, J., *ALMA Software Glossary*, http://www.mma.nrao.edu/development/computing/docs/joint/draft/Glossary.html, (2000).

[3] Emerson, D.T., Baars, J.W.M., *ALMA Test Interferometer Project Book*, Chapter 9 – ALMA Test Correlator, http://www.tuc.nrao.edu/~demerson/almapbk/test_int/chap9/chap9.pdf, (2000).

[4] Pisano, J.A., *ALMA Correlator Output Data and Computer Processing Rates*, ALMA Computing Memo 008 (1999).

[5] Emerson, D.T., Baars, J.W.M., *ALMA Construction Project Book*, Chapter 8 – Local Oscillators, http://www.tuc.nrao.edu/~demerson/almapbk/construc/chap7/chap7.pdf, (2001).

[6] Mangum, J.G., *On the Fly Observing at the 12 Meter*, http://www.tuc.nrao.edu/docs/oftdoc.ps, (1999).

[7] D'Addario, L.R., *Timing and Synchronization*, ALMA Memo 298, (2000).

[8] European Southern Observatory: http://www.eso.org/projects/vlt/sw-dev/oowg-forum/ATS/atcsdoc/Model/UseCases/Help/*.html (1999)

[9] Fowler, M., *UML Distilled: Applying the Standard Object Modeling Language*, Addison Wesley Longman, (1997).

[10] Markham, D., et. al, The Mark III analysis software (CALC/SOLVE) , http://www.sgl.crestech.ca/IVS-Analysis/software_tools/calc_solve/ivs_calcmain.htm

[11] Gomaa, H., Structuring Criteria for Real Time System Design. *Proc. 11th International Conference on Software Engineering*, (1989), pp.290-301.

[12] McLaughlin, M.J., Moore, A., "Real-Time Extensions in UML", *Dr. Dobb's Journal*, December 1998.

[13] Briand, L.P., Roy, D.M., *Meeting Deadlines in Hard Real-Time Systems – The Rate Monotonic Approach*", IEEE Computer Society, (1999).

[14] Escoffier, R., *Serial Communication and Control of the ALMA Test Correlator Serial Commands*, Internal memo, (2000-08-31).