ATLAS Level-1 Calorimeter Trigger 28 February 2001
Software Note 001 Version Draft 0.3

Local Controllers

Murrough Landon

1 Introduction

This document sets out the requirements and suggested implementation of the Local Controllers for the ATLAS Level 1 Calorimeter Trigger [1].

The Run Control [2] package of the ATLAS Online Software [3] establishes the concept of a Local Controller. This is a software component responsible for implementing the state transitions required to bring a well defined part of the ATLAS readout system from the initial, powered on state into the running state (and back).

1.1 Hardware Overview

The Calorimeter Trigger consists of a number of VME crates containing many different types of module.

- Eight preprocessor (PPr) crates: containing a CPU, up to 16 preprocessor modules (PPMs), one timing control module (TCM) and two preprocessor readout driver modules (PPRODs).
- Four cluster processor (CP) crates: containing a CPU, 14 cluster processor modules (CPMs), two common merger modules (CMMs) and one timing control module.
- Two jet/energy processor (JEP) crates: containing a CPU, 16 jet/energy modules (JEMs), two CMMs and one TCM.
- Two readout driver (ROD) crates: containing a CPU, up to 13(?) common processor readout driver modules (CPRODs), maybe one CERN BUSY module (if not in the TTC crate) and perhaps a TCM.
- One TTC crate: contain a CPU, one TTCvi module, one PPROD BUSY module, possibly one BUSY module for each of the two ROD crates (if not located within those crates) and one overall BUSY module (unless spare ports on one of the other BUSY modules are used for this purpose).

The trigger electronics chains will also contain

- Eight receiver crates (two for TileCal, six for LAr): containing up to 16 receiver modules and one SPACbus interface module. We will be responsible for the TileCal crates while the LAr groups will responsible for their receivers. However as the control of these systems is likely to be via a SPACbus controller (which can manage several SPACbus interface cards) sitting in some other crate, it may be worth having a single control crate for all eight receiver crates.

- One receiver control crate (perhaps): containing a CPU and a SPACbus controller module.

Additionally, in test setups, we may have:

- One or more DSS crates: containing a CPU and up to 6 DSS modules. In test setups, this crate may be combined with either the TTC crate or the crate containing prototype RODs.

The CPUs in each crate will be connected via a fast (or gigabit) ethernet switch. The system will be controlled by one or more workstations. The workstations will run various DAQ server processes (at least in standalone mode), monitoring, display and user interface programs.

2 Requirements

The L1Calo Local Controllers

- shall be part of the ATLAS Run Control system and satisfy all its requirements. This will be achieved by basing them on the standard Local Controller skeleton.

- shall be able to initialise and configure all the types of module found in the Calorimeter Trigger including test modules, loading all kinds of calibration and threshold data.

- shall be able to handle the complete final ATLAS system and arbitrary subsets for tests.

- shall be able to start and stop a normal run.

- shall be able to start, operate and stop any of our various types of calibration and test run.

- shall be able to change between run types.
- shall monitor the modules in each crate and report their status via the IS.
- should be designed to allow new run types to be added with minimal effort.

3 Design Issues

3.1 Run Control System Overview

The ATLAS Run Control scheme implements a hierarchy of controllers underneath a single root controller. It is expected that there will be single overall controller for each major subdetector and that each subdetector will consist of one or more lower levels of controllers. A subdetector with more than one TTC partition might have one controller per partition. At the lowest level there would generally be one controller per crate or readout system.

The run control system [2] defines a number of states and state transitions. The main states are Initial, Loaded, Configured and Running. There is also a Paused state, and a concurrent Fault state indicating the presence or absence of an error condition.

Each controller executes an action method on making each transition. When making a given state transition, a parent controller completes its transition action before any of its child controllers start their actions. In the normal case, all child transition actions occur in parallel (asynchronously). It is also possible to request, per controller per transition, that they occur one by one (synchronously) either in forward or reverse order, according to the order the children are listed in the database.

The above ordering is the only synchronisation mechanism available. It is not expected that actions in one crate should depend on actions in another crate and no communication between child controllers is implemented. Synchronisation issues in the L1Calo system are discussed in section 3.4.

3.2 Run Types

Apart from normal physics runs, we envisage a number of different kinds of calibration and test runs. The available run types can be defined for a given DAQ Partition and the IGUI allows them to be selected at any time up to the transition into the Running state.

In particular, the run type can be changed while the system is in the Configured state. Hence any actions which may be dependent on the run type should be done during the transition into the Running state. Examples probably include downloading trigger settings and maybe even calibration data; loading of playback memories with test vectors; etc.

3.3 Actions

The detailed actions for each transition in each crate need to be defined. A first draft of these, **for normal runs**, are given in the appendix in tables 1, 2, 3, 4, 5, 6, 7, 8.

Further work is required to list the additional actions required for the various types of calibration and test run.

In general, each crate will read all necessary information from the database at the Load step. Some run type independent settings and checks can be made at the Configure step. Final downloading of run type dependent information to the modules can be done at the Start step. It may be wise to use the Resource Manager (RM) package to lock crates (or possibly individual modules in some cases?) against simultaneous use by another partition.

Some actions on the whole system require the use of the TTC system. They are the kind of actions which might naturally be allotted to an overall system controller, but as they require access to the TTCvi are naturally implemented in the TTC crate.

Note that the DSS crate (or modules) are only required in some test setups.

3.4 Synchronisation

There are a few cases where we will require synchronisation between actions in different crates. Specifically, some actions must either precede or follow other actions. These requirements dictate the hierarchy of controllers we will need. While it would be possible to use a different hierarchy for different situations, eg calibration runs, it would be most desirable if the same hierarchy could be used in all cases.

The following actions require synchronisation:

- Configure step: TTC broadcasts to initiate LVDS synchronisation should ideally be done before VME actions in the affected crates.
- Start step: loading of playback memories and switch into playback mode must be done before the TTC broadcast to start synchronous playback from all modules (of a given type) in the system.

- Start step: TTC broadcast to start synchronous playback must be before triggers are enabled.
- Start step: configuration of all processor crates should be complete before the overall BUSY is removed. Q: within PP crates, is there an ordering required between PPMs and PPRODs?
- Start step: configuration of all processor crates must be complete before triggers are enabled. For local triggers this is easy as we enable our triggers via the TTCvi. For CTP and calorimeter calibration triggers however, we rely on the BUSY signal to veto triggers.
- Stop step: triggers should be disabled before playback mode is switched off.

Some other possibilities include:

- Start step: configuration of processor modules before/after RODs? Eg CP/JEP crates before/after ROD crates; PPMs before/after PPRODs in the same crate.

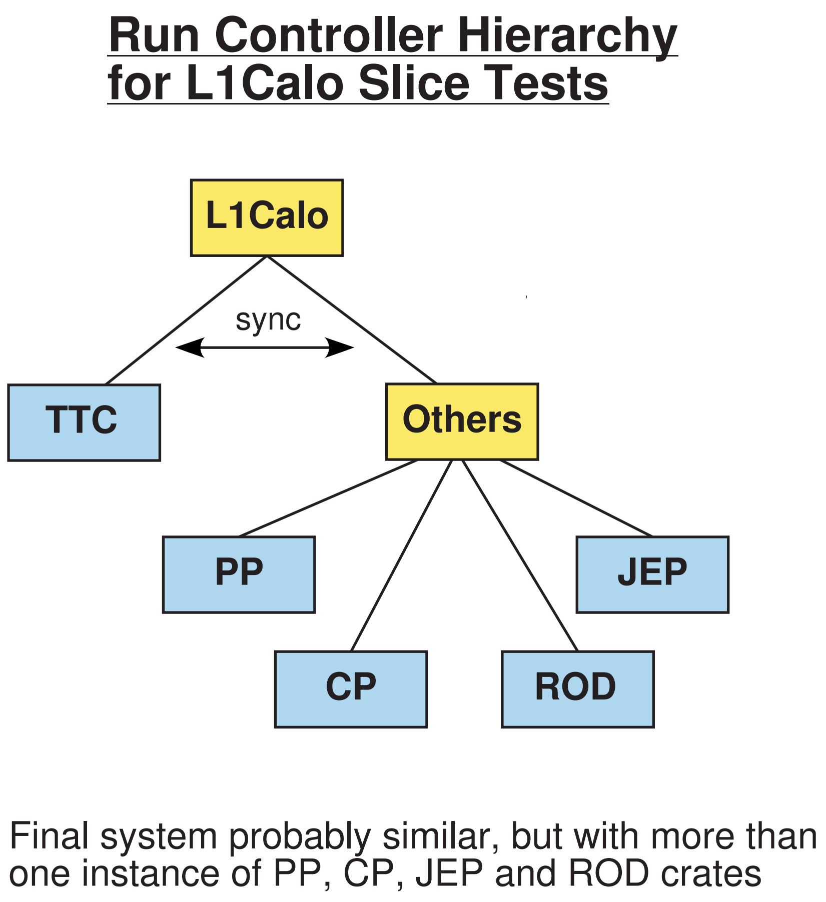
3.5 Hierarchy

The synchronisation issues discussed in the previous section require that some actions connected with trigger, TTC and BUSY signals must be executed either before or after actions in the trigger processor crates.

The specification for the BUSY module allows the output BUSY signal to be set directly via a VME register. Assuming our overall BUSY module is in the TTC crate, all the above can be satisfied (I think!) using a two level run control hierarchy where the TTC crate is separate from all the other crates. It is then a simple matter in the overall L1Calo run controller to select the order in which the actions in the “TTC” and “Others” controllers are executed in each transition.

Apart from possible concerns about actions in the ROD crate, all the other crate actions can run in parallel.

Given these assumptions, a hierarchy of run controllers suitable for the slice tests is shown in figure 3.5. In principle the hierarchy for the final system is identical, except that there will be more of each kind of crate. Also there will be controllers for the receiver crates which will not be present at the slice test.


Run Controller Hierarchy for L1Calo Slice Tests

Final system probably similar, but with more than one instance of PP, CP, JEP and ROD crates

L1Calo run control hierarchy for the slice tests.

3.6 Calibration

[Further work is required here]

Preliminary discussions with the LAr and TileCal groups suggest two possible scenarios for doing calibrations. Typically both involve a scan over a range of values of some parameter.

This scan can be done in a single run. In this case the system has to operate in a run/pause loop with the parameter value being changed during the pauses. Alternatively it can be a series of runs, each of which uses a single value of the parameter.

The multiple runs scheme may be controlled externally by some script. Otherwise there are not many implications for the run control hierarchy or transition actions.

The single run scenario however will require synchronisation between pauses of the trigger (handled in the TTC crate after a specified number have been taken) and changes of the parameter values in other crates. The Paused state (and Pause/Resume) commands can be used for this purpose.

In this scheme, there would be a separate process polling eg the TTCvi to count triggers. This process would then send Pause and Resume commands to the Root run controller at the appropriate times. This "calibration sequencer" process would be started and stopped by the normal run Start and Stop commands.

3.7 Implementation

It is expected that each crate in the system will have its own CPU It is therefore sensible that there be one Local Controller running in each crate CPU to control the modules in that crate.

It is not clear if each crate CPU will have its own disk. If so, the calibration and other data required to initialise each crate should be kept local to that CPU. In most cases, the calibration data will be generated locally. However the threshold settings etc will need to be distributed to each crate from the central database.

Some of the various kinds of crates share the same kind of module. So the Local Controllers in different crate will inevitably share some code. One approach may be to develop a single Local Controller program containing the code relevant to all module types. This program can read from the database which kind of crate it is operating in and can handle the modules it finds there.

Apart from code specific to modules, there may well be other ancillary code common to the Local Controller of different crates which will also be useful to share. Also, in test setups, some crates may contain combinations of modules which will always be in separate crates in the final system.

However if it turns out that the commonalities are exceeded by differences between the various crates, splitting into several different programs may be more manageable.

4 IS server and IGUI

The Local Controller in each crate needs to be able to publish information about its crate. For this purpose we will run a dedicated IS server [4] for the L1Calo system.

This should contain the following information:

- for each module in the full configuration, a status block with entries indicating if the module is actually present, whether its links to other modules are up, detailed error counts and any other useful data.
- for each crate in the full configuration, a status block with entries indicating if the crate is currently enabled, and summaries of error counts from modules within the crate.
- from each PPM, the rate histogram for each channel (is this too much data to ship around? Perhaps each crate just publishes a limited amount of processed information?).
- other histograms from each crate?

We will almost certainly want to be able to specify run control parameters to the run controllers – especially for calibration runs. The standard way to do this is via L1Calo specific panels in the Integrated GUI (IGUI). Such panels, written in Java, can set and display run parameter variables in our IS server. These can be read by the run controllers before they execute each transition.

We should also provide an IGUI panel to display the status information read from each module and summarised for each crate.

References

[1] ATLAS Level 1 Calorimeter Trigger: home page
`http://hepwww.pp.rl.ac.uk/Atlas-L1`

[2] ATLAS Online Software Run Control component
`http://atddoc.cern.ch/Atlas/DaqSoft/components/runcontrol`

[3] ATLAS Online Software home page
`http://atddoc.cern.ch/Atlas/DaqSoft`

[4] ATLAS Online Software Information Service component
`http://atddoc.cern.ch/Atlas/DaqSoft/components/is`

A State transition actions

The following tables give the detailed transition actions for each type of module.

Transition	**Actions**
Initial$\rightarrow$Loaded	Read database: – expected configuration of modules – $E_t$corrections Lock crate/modules via Resource Manager?
Loaded$\rightarrow$Config	Check expected modules are actually present Load $E_t$corrections into receiver modules via SPACbus. [Assuming not run type dependent] Update module status in L1Calo IS server?
Config$\rightarrow$Running	None
Running$\rightarrow$Paused	None
Paused$\rightarrow$Running	None
Running$\rightarrow$Config	None
Config$\rightarrow$Loaded	None
Loaded$\rightarrow$Initial	Unlock crate/modules via Resource Manager

Table 1: *Actions for TileCal Receiver Crate*

Transition	Actions
Initial→Loaded	Read database: – expected configuration of modules – default FPGA versions – dead/hot/disabled channel masks – energy calibrations – coarse and fine timing calibrations – BCID pulse shape calibrations – BCID/FIR/saturation settings – readout settings – default rate histogram settings – PPROD settings Lock crate/modules via Resource Manager?
Loaded→Config	Check expected modules are actually present Check FPGA versions: reload and reset if required Assert BUSY on all PPRODs Update module status in L1Calo IS server? Start rate monitoring program/thread
Config→Running	Load all calibrations and settings into PPMs Load settings (if any) into the TCM Load readout settings and default monitoring selections into PPRODs Zero rates and/or other statistics for this run Remove BUSY from all PPRODs
Running→Paused	Assert BUSY on all PPRODs? **Calibration/Test:** Load value(s) for next calibration step?
Paused→Running	Remove BUSY from all PPRODs?
Running→Config	Assert BUSY on all PPRODs Log rates summary and other statistics for this run to database?
Config→Loaded	Stop rate monitoring program/thread
Loaded→Initial	Unlock crate/modules via Resource Manager

Table 2: *Actions for PreProcessor Crate*

Transition	Actions
Initial→Loaded	Read database: – expected configuration of modules – default FPGA versions – dead/hot/disabled channel masks – timing settings – threshold settings – readout settings Lock crate/modules via Resource Manager?
Loaded→Config	Check expected modules are actually present Check FPGA versions: reload and reset if required Update module status in L1Calo IS server? Start crate monitoring program/thread/callbacks
Config→Running	Load thresholds and settings into CPMs Load settings into CMMs Load settings (if any) into the TCM Zero module error counts and/or other statistics for this run
Running→Paused	**Calibration/Test:** Load value(s) for next calibration step?
Paused→Running	None
Running→Config	Log error counts and other statistics for this run to database?
Config→Loaded	Stop crate monitoring program/thread/callbacks
Loaded→Initial	Unlock crate/modules via Resource Manager

Table 3: *Actions for Cluster Processor Crate*

Transition	**Actions**
Initial→Loaded	Read database: – expected configuration of modules – default FPGA versions – dead/hot/disabled channel masks – timing settings – threshold settings – readout settings Lock crate/modules via Resource Manager?
Loaded→Config	Check expected modules are actually present Check FPGA versions: reload and reset if required Update module status in L1Calo IS server? Start crate monitoring program/thread/callbacks
Config→Running	Load thresholds and settings into JEMs Load settings into CMMs Load settings (if any) into the TCM Zero module error counts and/or other statistics for this run
Running→Paused	**Calibration/Test:** Load value(s) for next calibration step?
Paused→Running	None
Running→Config	Log error counts and other statistics for this run to database?
Config→Loaded	Stop crate monitoring program/thread/callbacks
Loaded→Initial	Unlock crate/modules via Resource Manager

Table 4: *Actions for Jet/Energy Processor Crate*

Transition	Actions
Initial→Loaded	Read database:
	– expected configuration of modules
	– default FPGA versions in each CPROD
	– monitoring sampling settings
	Lock crate/modules via Resource Manager?
Loaded→Config	Check expected modules are actually present
	Check FPGA versions: reload and reset if required
	Configure crate BUSY module (if in this crate)
	Assert BUSY on all CPRODs
	Update module status in L1Calo IS server?
	Start crate event monitoring program(s)
Config→Running	Load settings into CPRODs
	Load settings (if any) into the TCM (if present??)
	Zero statistics for this run
	Remove BUSY from all CPRODs
Running→Paused	Assert BUSY on all CPRODs?
Paused→Running	Remove BUSY from all CPRODs?
Running→Config	Assert BUSY on all CPRODs
	Log statistics for this run to database?
	Save local monitoring histograms for this run to database?
Config→Loaded	Stop (or pause?) crate monitoring program(s)
Loaded→Initial	Unlock crate/modules via Resource Manager

Table 5: *Actions for ROD Crate*

Transition	**Actions**
Initial→Loaded	Read database: – expected configuration of modules (TTCvi, BUSY, CORBO, etc?) – expected configuration of PPr and ROD crates – TTCvi settings Lock crate/modules via Resource Manager?
Loaded→Config	Check expected modules are actually present Assert overall BUSY Load settings into TTCvi Send TTC broadcast to start/stop LVDS synchronisation? Configure PPr BUSY module to enable PPRODs in PPr crates Configure CP/JEP BUSY modules (if located in TTC crate) to enable CPRODs in ROD crates Configure system BUSY module to enable the other BUSY modules (if enabled in the configuration) Update module status in L1Calo IS server? Start crate monitoring program/thread/callbacks
Config→Running	(Re)load settings into TTCvi? **Calibration/Test:** Send TTC broadcast to start synchronous playback Remove overall BUSY **Calibration/Test:** Enable local triggers
Running→Paused	**Calibration/Test:** Disable local triggers
Paused→Running	**Calibration/Test:** Enable local triggers
Running→Config	**Calibration/Test:** Disable local triggers Assert overall BUSY
Config→Loaded	Stop crate monitoring program/thread/callbacks
Loaded→Initial	Unlock crate/modules via Resource Manager

Table 6: *Actions for TTC Crate*

Transition	Actions
Initial→Loaded	Read database: – expected configuration of modules and their daughtercards – sets of test vectors to be loaded Locking crate/modules via Resource Manager probably not required for test systems?
Loaded→Config	Check expected modules are actually present Load test vectors into DSS modules Update module status in L1Calo IS server? Start crate monitoring program/thread/callbacks (if any)
Config→Running	Load test vectors into DSS modules NB start of playback initiated by TTC broadcast from TTC crate
Running→Paused	None
Paused→Running	None
Running→Config	None (stop of playback signalled by TTC broadcast?)
Config→Loaded	Stop crate monitoring program/thread/callbacks (if any)
Loaded→Initial	Unlock crate/modules via Resource Manager (if locked)

Table 7: *Actions for DSS Crate*

Transition	Actions
Initial→Loaded	Read database: – expected configuration of crates – default run parameters? Lock L1Calo system via Resource Manager?
Loaded→Config	Set synchronous transition order: TTC before others Update system status in L1Calo IS server? Start overall monitoring program/thread/callbacks?
Config→Running	Set synchronous transition order: others before TTC Zero systemwide statistics/histograms/etc? **Calibration/Test:** Start calibration sequence process?
Running→Paused	None
Paused→Running	None
Running→Config	Set synchronous transition order: TTC before others Save systemwide statistics/histograms/etc to database?
Config→Loaded	Stop overall monitoring program/thread/callbacks?
Loaded→Initial	Unlock L1Calo system via Resource Manager

Table 8: *Actions for Overall Controller*