The MVD Data Acquisition System

(7-1-98,very preliminary)

This note contains a description of the MVD DAQ system, from the readout side of the HELIX chip to the EVB TPM and offline monitor jobs. Clearly the contents can only represent the current understanding and too much is missing for a complete specification.

1 Contents

1 CONTENTS 1

2 HARDWARE 4

2.1 Overview 4

2.2 CC sub-system 5

2.3 RL sub-system 6

2.4 ADC sub-system 6

2.5 GSLTI sub-system 7

2.6 SLT 7

2.7 EVBI sub-system 7

2.8 SC sub-system 8

2.9 VME CPUs 8

3 SOFTWARE 9

3.1 Generalities 9

3.2 Network communication 9

3.3 XDR message format 9

3.4 Service brokering 9

3.5 Process control 9
3.6 Code and management 9
3.7 Databases 10
3.8 Partitioning 10
3.9 Debugging 10
3.10 Monitoring 10
3.11 Histogramming 10
3.12 Operator 10
3.13 RC software 10
3.14 RC component software 11
3.15 SLT software 12
3.16 SC software 12
3.17 SC component software 13
3.18 Monitor software 14
3.19 Operator software 15
3.20 Database software 16
3.21 WEB software 17
3.22 Conclusions 17
4 CALIBRATION 17
5 MONITORING 18
5.1 Online 18
5.2 Offline 19
6 RUN SEQUENCING 19
6.1 Overview 19
6.2 IDLE 20
6.3 DO CALIB 20
6.4 CALIB 20

6.5 SETUP 20

6.6 READY 22

6.7 ACTIVATE 22

6.8 ACTIVE 23

6.9 PAUSE 25

6.10 PAUSED 25

6.11 RESUME 25

6.12 END 25

6.13 ABORT 26

7 ERROR CONDITIONS 26

7.1 ADC sub-system 26
7.1.1 Timing errors 27
7.1.2 Mismatch errors 27

7.2 Next… 27

8 ERROR HANDLING 27

8.1 Re-initializing Helix chains 27

8.2 Remove a chip from a chain 28

8.3 Disable a chain at the ADC unit level 28

8.4 Download ADC constants whilst ACTIVE 29

Figure 1 Schematic of MVD DAQ system. 5
Figure 2 Schematic of Clock and Control sub-system. 6
Figure 3 ADC Crate system. 6
Figure 4 Control Signal. 7
Figure 5 RC process connectivity. 11
Figure 6 RC component process connectivity. 12
Figure 7 SC process connectivity. 13
Figure 8 SC component software process connectivity. 14
Figure 9 Online and offline monitoring process connectivity. 15
Figure 10 Operator process connectivity. 16
Figure 11 Database process connectivity. 17
Figure 12 MVD RC state transition diagram. Ovals are states and transitions are boxes. 20
Figure 13 Data processing timing. 27

2 Hardware

2.1 Overview

The DAQ hardware is shown schematically in Figure 1. Major sub-systems are:

- Clock and control (CC)
- Readout link (RL)
- ADC digitizers (ADC)
- Second level Trigger (SLT)
- Event builder interface (EVBI)
- GSLT interface (GSLTI)
- Slow Control (SC)

The associated electronics is mounted in crates located in the cryo-tower, ca. 5m from the detector, or in the rucksack, ca. 45m distance, by cable, from the cryo-tower.

A common feature of the VME crates is the presence of, at least one, board computer. These machines are used to provide the functionality required by the DAQ system (initialization, data gathering and transport, etc.) and are connected over the network.

A brief overview of the sub-systems and VME CPUs is given below. More detail can be found in the references given.

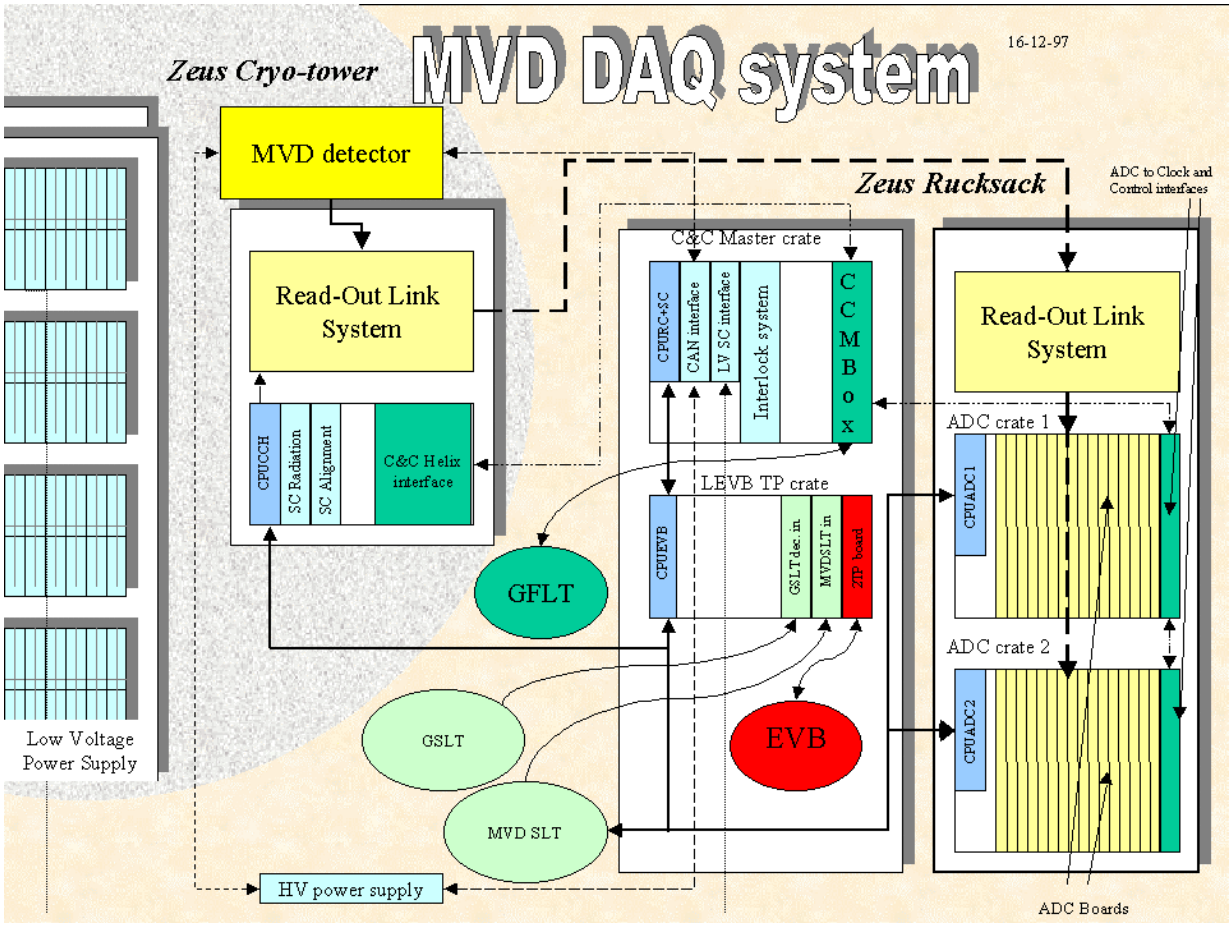

Figure 1 Schematic of MVD DAQ system.

2.2 CC sub-system

The CC's task is to interface the DAQ system to the GFLT and HELIX chips. A detailed description of the CC is available [1]. The functionality provided is shown in Figure 2.

The **Masterbox** is a 9U VME module located in the rucksack. The **Helix interface** consists of 16 6U VME modules in the cryo-tower. One **CC-ADC interface** module is present in each ADC crate. The modules can be accessed, over VME, using the crate CPU.

The Helix interface allows configuration data to be downloaded (it cannot be read back for validation) to the HELIX chips, via one serial control line per token. The Helix interface is controlled over VME by CPUCCH.

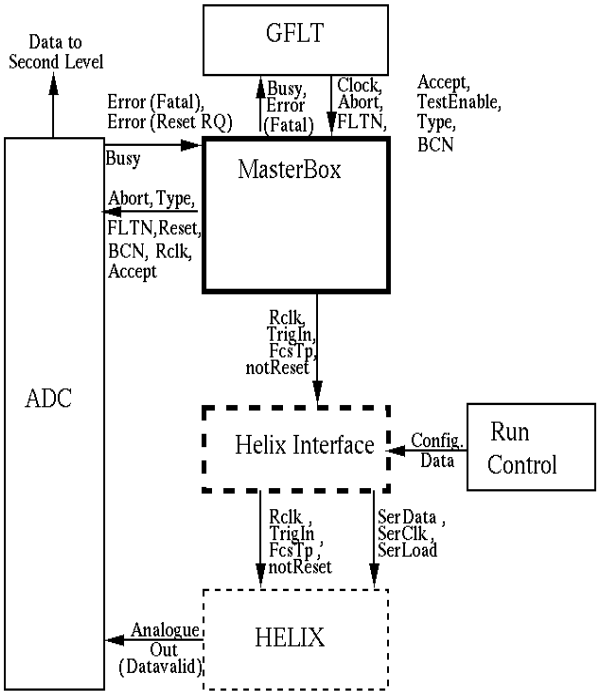

Figure 2 Schematic of Clock and Control sub-system.

2.3 RL sub-system

The readout link connects the HELIX readout with the ADC sub-system. A description [2] of the RL sub-system is available.

The implementation requires cryo-tower and rucksack space.

2.4 ADC sub-system

The ADC's task is to digitize data from HELIX chips and store the results for down-stream. A detailed description of the ADC sub-system [3] is available. A block diagram and the control signals of the ADC system are shown in Figure 3, Figure 4.

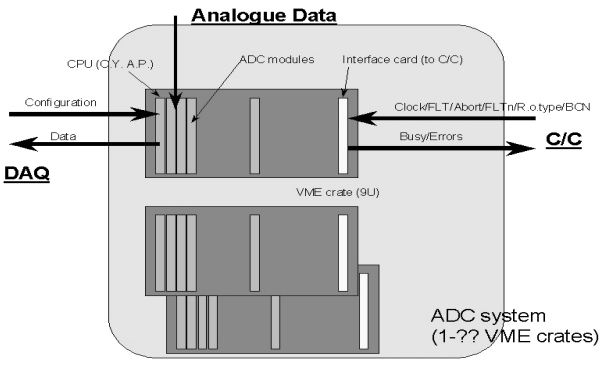

Figure 3 ADC Crate system.

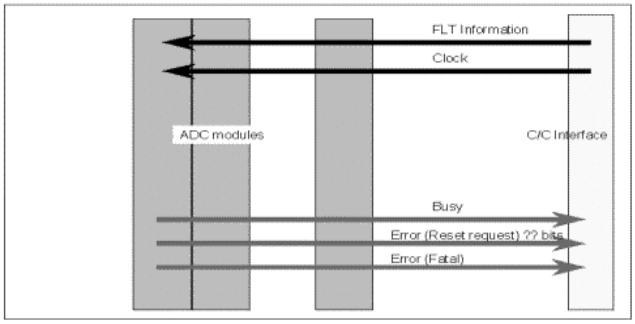

Figure 4 ADC/CC Control Signals.

Two ADC crates are foreseen, each containing a 6U CPU, ca. 16 9U ADC boards and a CC ADC interface.

The CC-ADC interface receives the GFLT clock, number, accept, run-type, abort, bunch crossing number and allow the busy handling between ADC and CC.

The presence of two CC-ADC interfaces may require special handling at CC sub-system if an error state has to be the logical OR of both.

In order to have fast data transfer at GFLT accept and error handling the CPUADCs need to be interrupt driven. These interrupts have to be generated directly by the CC/ADC system or an additional VME board driven by the CC/ADC system.

2.5 GSLTI sub-system

The GSLTI is used to send SLT trigger data to the GSLT and receive GSLT accept decisions.

The GSLTI hardware is currently a transputer in the NIKHEF 2TP (6U) VME module shared with the EVBI.

Data from MVDSLT and GSLT could be handled also through a transputer interface connected to a CPU controller device (TP to FIFO). MVDSLT output could also be a not TP based link. Specification needed.

2.6 SLT

The SLT system remains undefined. The connection link technology to be used may be bus, ring or switch based. For the purpose of this note a TLT farm implementation is assumed.

2.7 EVBI sub-system

The EVBI's task is to receive event data associated with a GSLT accept decision and transfer this to the EVB transputer TPM.

The EVBI sub-system is currently a transputer in the NIKHEF 2TP (6U) VME module shared with the GSLTI.

2.8 SC sub-system

The slow control consists of the following separate components:

- **Detector.** Temperature and humidity monitors mounted on the detector hybrid cards. CAN readout.
- **Alignment.** Laser based detector position monitoring system. VME readout.
- **Radiation.** Radiation rate monitor. VME readout.
- **Low voltage.** Low voltage supply based on LPS design. VME readout.
- **Bias voltage.** Bias voltage supply. CAN readout.
- **Interlock.** Not yet specified interlock system.
- **Test.** TTL input/output slave modules for use in the template component. CAN readout.

The CAN systems are directly connected to CPUSC CAN 2.0A interfaces. VME systems are accessed via the CPU in the relevant crate.

2.9 VME CPUs

At least one VME CPU is required per VME crate. The following list defines the names (not the IP name) of the systems likely to be present.

- **CPUCCH** located in the cryo-tower crate containing the CC Helix interface card.
- **CPUCCM** located in a rucksack crate containing the CC Masterbox.
- **CPUADC1** located in the first ADC crate.
- **CPUADC2** located in the second ADC crate.
- **CPUEVB** located in the crate containing the EVBI and GSLTI sub-systems.
- **CPUSC** located in the rucksack and used for slow control.
- **CPUSPY** located in the CPUEVB crate and used by monitoring.

Note that no CPU is specified for run control, this will probably run on CPUSC, and that CPUSPY may be a PC within the unspecified implementation of the SLT.

A MVME2600 has been ordered with a LYNX-OS operating system. This combination might be a candidate for the other CPUs whereby the MVME2300 (2600 without SCSI) is considerably cheaper and might be more applicable in a network boot environment.

3 Software

3.1 Generalities

The MVD is a component of the ZEUS DAQ and must adhere to the application interfaces, which exist there. In this respect the software required is well understood and working examples, i.e. templates and other components, already exist. This section describes some of the concepts relevant to the software design.

The assumption is that the existing CDAQ software should be used throughout and that as little effort as possible be invested in software development. Details specific to the MVD hardware will, of course, require individual solutions.

3.2 Network communication

It is clear that the MVD DAQ system, at the software level, will consist of numerous processes running on different CPUs, and that these processes will communicate by passing messages over IP. Depending on the final DAQ architecture, the data will also be transferred on an IP network.

3.3 XDR message format

In order to avoid problems associated with data representations on different hardware platforms only XDR [4] encoded messages will be transferred. The required level of XDR support is already available on the platforms in use at ZEUS and can be easily provided on others.

All MVD messages will be XDR tagged union structures and will be defined in a unique XDR definition file. The definition file is used to automatically generate the encoding and decoding (serializing and de-serializing) routines. Incoming messages failing the decoding routine are discarded as illegal. The automatic generation of encoding and decoding routines removes the reliability hazard which exists when every user is forced to provide their own versions.

Exceptions to this rule are those messages exchanged with CDAQ software packages (RC use strings and the new SC and daemon system use XDR).

XDR encoded messages will be transmitted using ZMP[5].

3.4 Service brokering

Processes are required to advertise the services they offer. The advertisement of services allows crashed, aborted or otherwise restarted process to find and reconnect (self-synchronize) to the services they require. On acquiring the service a register message is sent to the service. On resigning the service a corresponding de-register message is sent.

The name server associated with named ZMP communications interfaces should be used for service brokering.

3.5 Process control

The ability to start, stop, and acquire the current state (live, dead, run time, open files, etc.) of a process is essential. This feature can be used to tailor dynamically which processes are running and the number of instances according to the requirements of the DAQ.

The ZMP based daemon system should be used for process control.

3.6 Code and management

C code is sufficient given the small amount of new code required. Using the MVD DAQ environment

as a platform for learning C++ should not be recommended.

A code management environment is required were all updates can be made with a single make operation and were some degree of versioning is foreseen. More enlightened approaches should be looked at, but initially nothing more than a make file is required.

3.7 Databases

A DAQ wide database is required for storing related data (configuration information, run history, monitoring information, calibration, etc.).

Microsoft Access is being used in the new CDAQ slow control system and should be used as the MVD DAQ database. A process offering the service to other DAQ processes will handle the SQL based communication with the database. This process de-couples SQL from the XDR message definitions and can be implemented immediately.

3.8 Partitioning

The MVD DAQ system normally will run within the context of the ZEUS DAQ system and the partition is the entire MVD system. Partitioning into local RC components may be required during debugging, an ADC test program may need CC functionality, this can be provided by service brokering.

3.9 Debugging

Logging processes are required to view the system behavior. These will be connected to services being debugged and can have their output routed to display or storage devices.

3.10 Monitoring

Data quality monitoring processes are foreseen in both the online and offline environments. Clearly these processes must be capable of interacting with other DAQ processes even though they may be written in FORTRAN and use offline tools such as ADAMO.

The evaluation of calibration data will be performed by an online monitor task.

3.11 Histogramming

Any process can provide histograms. These should be accessible via a CDAQ server.

3.12 Operator

This section should have been called Expert system however, this phrase has been somewhat debased, so lets call it Operator. The idea is that this process will be available to make decisions based on information received from other DAQ processes. An example would be that the monitoring system finds an error. The operator process would then decide what to do.

The importance of the operator is initially as a receiver of messages which cannot be processed by the sender because their context is not wide enough.

3.13 RC software

The MVD-RC process manages the state of the MVD DAQ system. The MVD-RC process behaves like the CDAQ-RC and has, itself, no state, but holds the global state of the components. The existing component ZMP template software will be used to provide the basic MVD-RC functionality. The MVD-RC process environment is shown in Figure 5.

CDAQ-RC is the ZEUS run control process. During normal running this process controls the MVD-RC process. The communication consists of transition requests and acknowledgements as defined in

the CDAQ documentation.

LOCAL-RC is used, typically during debugging, to control the MVD-RC. Only one LOCAL-RC instance can exist and during this time the MVD-RC state seen by CDAQ-RC is offline. Initially the terminal and script implementations of the process are required.

Communication with the MVD-CC, -ADC, -EVBI, -SLT, -GSLTI, and –SC processes is required, by MVD-RC, to carry out the state transitions required. These processes have their own environments as described later in the RC component software section.

LOGGER processes connect to MVD-RC and receive a copy of all messages generated by the system. There is no restriction on the number of LOGGER processes.

The EVENT processes receive notification of state transition and errors. They could be implemented as Loggers, with message filtering, but have been allocated a separate interface at the moment. Again any number of EVENT process can exist.

The MVD-RC manages the global state of the DAQ by sending and receiving messages. It has no directly connected hardware and the TRANSITION processes shown are redundant.

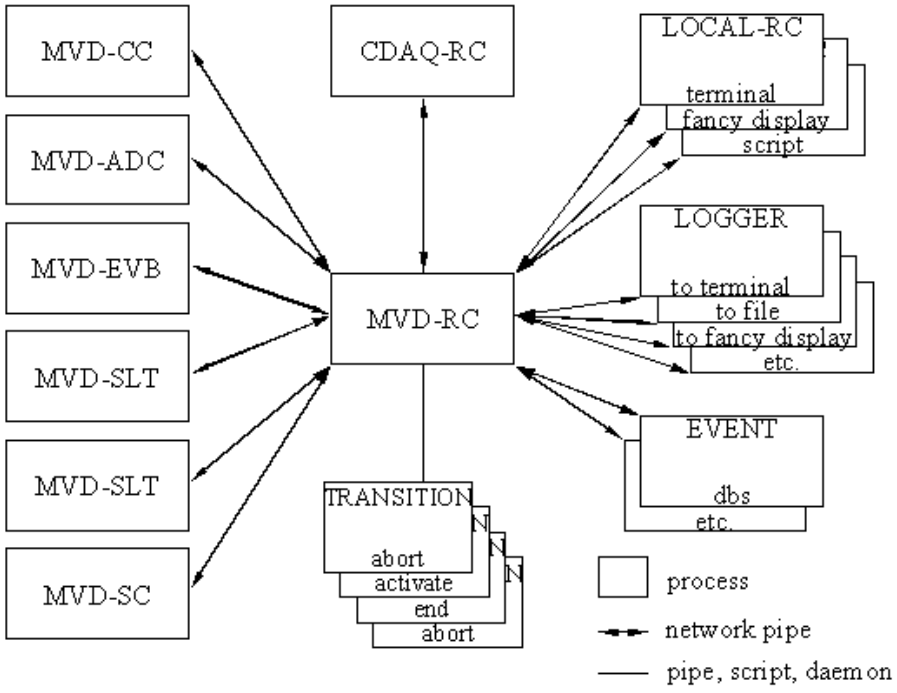

Figure 5 RC process connectivity.

3.14 RC component software

The MVD DAQ component (-X) system: MVD-CC, -ADC, -EVBI, -SLT, and -GSLTI processes are responsible for the underlying hardware components of the system and communicate with the MVD-RC process. They must implement the state transition diagram show in Figure 12. The similarities between the MVD-RC and MVD-X processes are clear, see Figure 5, and processes should be reused accordingly.

MVD-RC controls the component during normal running. The communication consists of transition

requests and acknowledgements.

LOCAL-X is used, typically during debugging, to control MVD-X. Only one LOCAL-X instance can exist and during this time the MVD-X state seen by MVD-RC is offline. Initially the terminal and script implementations of the process are required.

LOGGER processes connect to MVD-X and receive a copy of all messages generated by the system. There is no restriction on the number of LOGGER processes.

The EVENT processes are similar to Loggers, but receive notification of state transition and errors. Again any number of EVENT process can exist.

One implementation of the X processes is as spawned (forked) sub-processes (children) connected to MVD-X by mailboxes (pipes) or a script or daemon based script, provided the operation returned a meaningful exit status. Its task is to carry out the transition required. Only one X process can exist at a given time. Another implementation foresees the transition being carried out entirely within the MVD-X process. The choice is driven by the requirement of having no blocking operation w.r.t. the ability of MVD-X to respond to status and other requests.

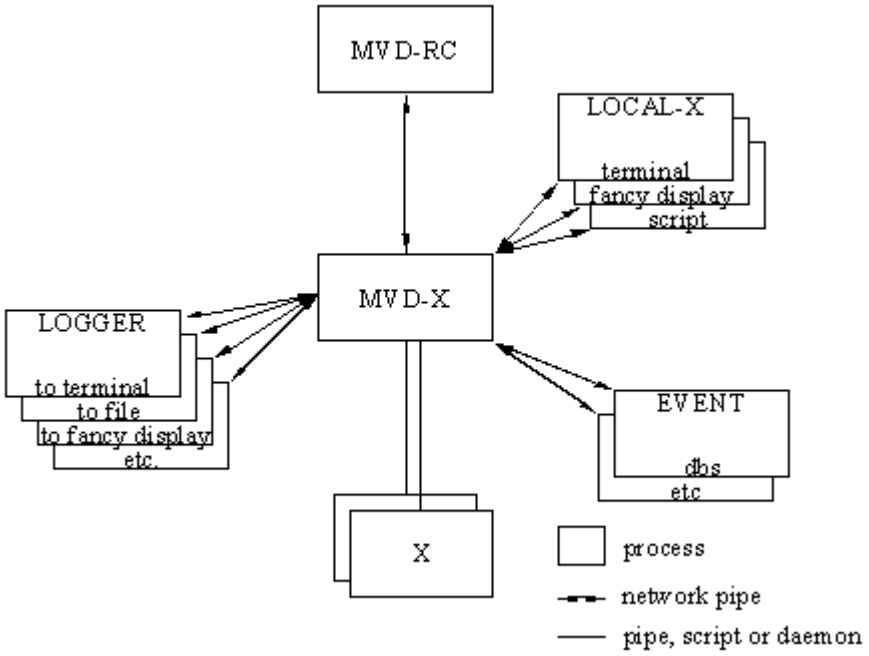

Figure 6 RC component process connectivity.

3.15 SLT software

The implementation of the SLT is uncertain. For the purposes of this note the SLT software is assumed to be similar to the TLT, i.e. the SLT is a processor farm, and the MVD-RC process will communicate with a MVD-SLT process, which in turn is communicating with SLT analyzers. MVD-SLT will implement the global state control used in MVD-RC.

3.16 SC software

The process connections to the MVD-SC process are shown in Figure 7.

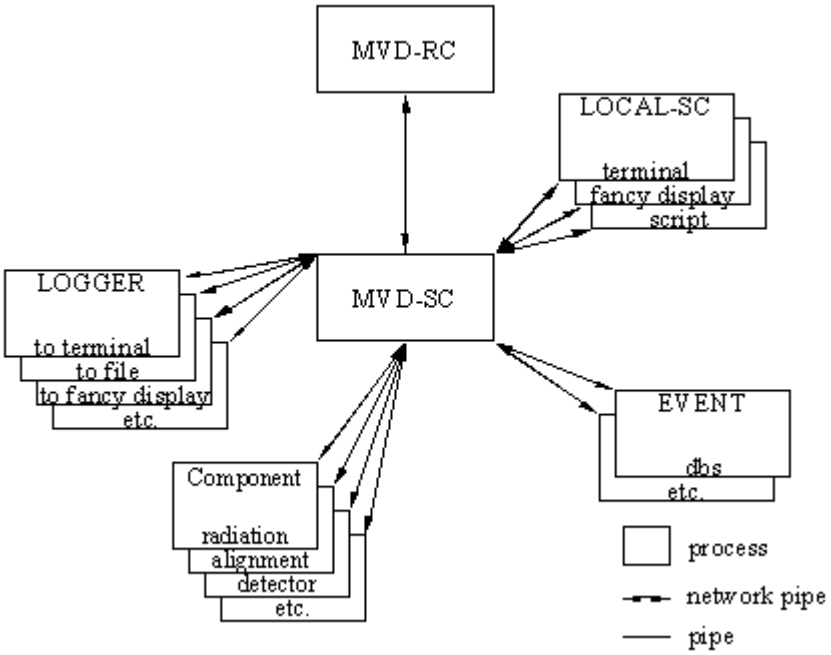

Figure 7 SC process connectivity.

The LOGGER, EVENT, LOCAL-SC and MVD-RC connectivity is identical to that described above.

The SC components (radiation, alignment, etc.) exist within the CDAQ-SC [6] system and connection to MVD-SC uses ZMP communications definitions defined.

MVD-SC has no connection to the CDAQ-SC system and implements same global state transition diagram as the MVD-RC process.

3.17 SC component software

The Environment of the SC sub system is shown in Figure 8.

MVD-Y is the slow control component seen by the CDAQ-SC and MVD-SC. It is assumed that the implementation of MVD-Y will be exactly identical to that described in the CDAQ proposals for the upgraded slow control. The histogram shared memory section, shown, is part of that specification.

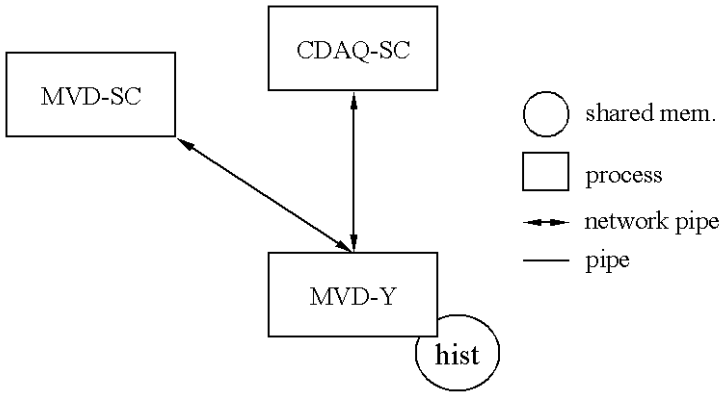

Figure 8 SC component software process connectivity.

3.18 Monitor software

Two monitor processes are likely to exist, an online monitor and an offline monitor.

The online MONITOR receives data (events) from the DATA SERVER. Errors found in the data (dead strips, etc.) are notified to the OPERATOR for possible correction. CDAQ histogram servers serve the histograms produced to other users. Data storage and retrieval from the database is foreseen.

The online MONITOR job has no direct connection to the MVD-RC system and is started by the process control when required. The analysis job (C or FORTRAN) should be forked from the MONITOR allowing the MONITOR to immediately respond to status requests.

The DATA SERVER process will spy events from the MVD EVB input buffer.

The offline MONITOR is running outside the hall DAQ system on an offline monitor batch queue. Communication with the DAQ will use script-based messages. This mechanism is used already in the data corruption warning system and has the advantage of de-coupling offline jobs with DAQ requirements. The histograms available at the offline MONITOR may be accessible to the DAQ servers.

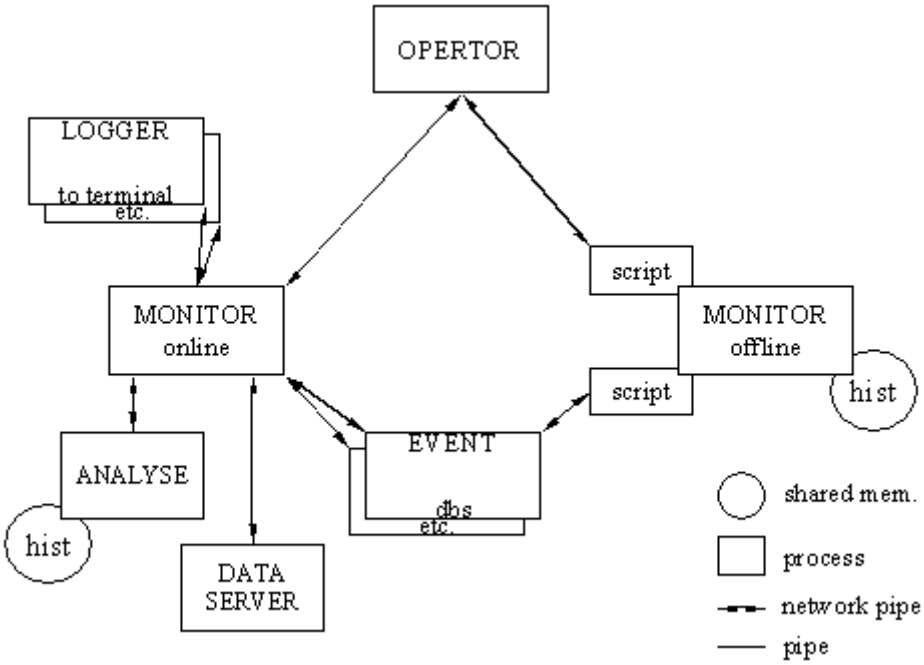

Figure 9 Online and offline monitoring process connectivity.

Provided the monitor tasks work correctly there should be no need to for deputy shift checks of MVD histograms.

3.19 Operator software

The OPERATOR process receives messages, which may require the DAQ system to modify its configuration.

An example would be the notification, from MVD-RC, that a run has been successfully activated. The OPERATOR process would then use the process control system to start an online MONITOR via the connected daemon system.

The process connectivity, Figure 10, has many features in common with other DAQ processes.

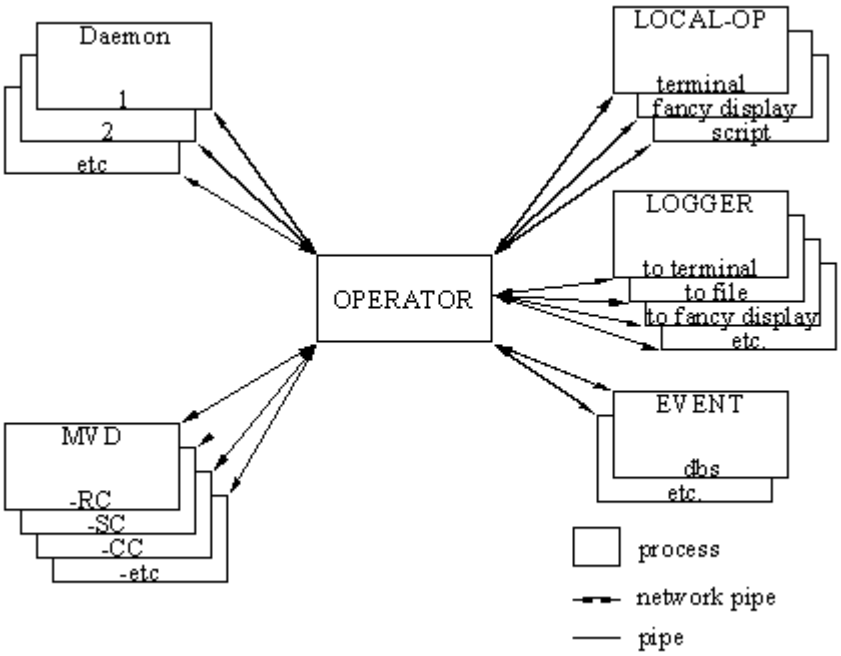

Figure 10 Operator process connectivity.

3.20 Database software

As already mentioned the database is hidden behind a ZMP process. The process environment associated with this is shown in Figure 11. The LOCAL-DBS and LOGGER are identical to that described above.

The REQUEST connections are responsible for query and update transactions with the database.

The EVENT connections are funneling state transitions and error conditions to the database.

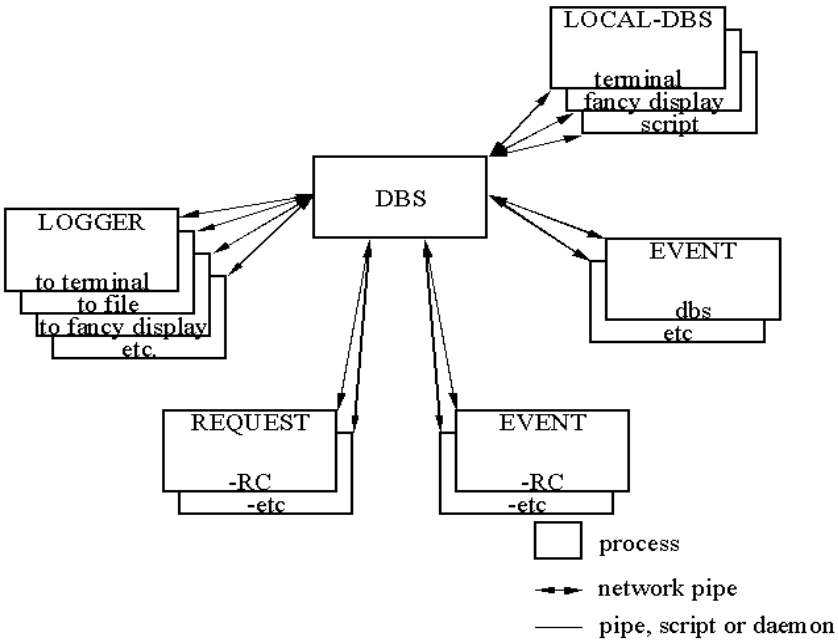

Figure 11 Database process connectivity.

3.21 WEB software

The use of ZMP within the hall process environment is sensible. WEB access should be foreseen for data access and control from outside.

3.22 Conclusions

The key features are unique XDR message definition and the reuse of code and processes.

4 Calibration

Calibration data is required for:

- Evaluating pedestal values
- Evaluating threshold values
- Evaluating chip gain
- Identifying faulty hardware (dead strips, etc.)
- Helix test runs
- Optical link gain
- etc.

No experience exists concerning the stability of the MVD electronics and in particular the

understanding of how much time is required after powering on for the system to reach a stable state. In the following discussion it is assumed that the electronics has reached a known stable state.

Four sources of calibration data are foreseen:

- **Offline standalone calibration**. The MVD DAQ system is configured to run with CC generated triggers and is not available to ZEUS DAQ. Offline standalone runs can be done at anytime by a person taking control of the MVD DAQ system, via LOCAL-RC. This should be the exception, rather than the rule, since we do not want to man calibration shifts.

- **Inline standalone calibration**. The MVD DAQ system is configured to run with CC generated triggers and is available to ZEUS DAQ. Inline standalone runs are configured whilst the MVD DAQ system is available to the ZEUS DAQ. Configuration is triggered either by the MVD itself or by a request from ZEUS DAQ. A policy is required on when and how these runs should be terminated on receiving a new configuration request.

- **In transition calibration**. The MVD DAQ system is configured to run with CC generated triggers during a ZEUS DAQ transition. The time available is limited by the targeted transition time.

- **In run calibration**. The MVD DAQ system is configured to run with GFLT generated triggers during the ZEUS DAQ active phase. In standalone and in transition runs only calibration events are transported through the DAQ. In run calibration can utilize GFLT test triggers or any other trigger type.

In all cases a MONITOR process will analyze the data. Hardware errors detected during calibration invalidate the results and require a correction to be made before making another calibration run. The sensibility of the calibration results must be checked before updating the database and downloading the constants.

5 Monitoring

5.1 Online

As previously stated, monitoring is performed by MONITOR processes running on CPUSPY and the daemon system is used to ensure that the correct process is running.

The principal tasks carried out by the ANALYSER of the in run MONITOR would include:

- Identification of bad strips (noisy, inefficient, etc.).

- Identification of errors associated with multiple strips (faulty digitization, incorrect clustering, etc.).

- Evaluating in run calibration data.

- Monitor up-stream error flags (ADC readout errors, etc).

- Fill histograms (strip occupancy, etc.)

Actions initiated by the MONITOR would be updates of the database (bad strips etc.) and indirectly, via the OPERATOR, downloading of new constants to the ADC system.

The strip data gives a handle on the efficiency. By evaluating the occupancy (strip hits/event %) at fixed event intervals dead, noisy or low efficiency strips can be identified. Similarly the cluster information (hit map, multiplicity, etc.) can be provide useful information. All useful values should be plotted, whereby the large number of strips (256k) requires some thought.

Access to monitoring results is provided by histogram servers or statistics servers. The deputy shift should not need to scan MVD histograms provided the ANALYSER works correctly.

5.2 Offline

The existing CTD offline monitor (TCBOL) can be used as a template for the MVD monitor.

6 Run sequencing

6.1 Overview

The state transition diagram of the MVD run control is shown in Figure 12. The MVD-RC process behaves like the CDAQ-RC and has, itself, no state, but holds the global state of the components. Thus the **global Consensus State**, **global Majority State** and **global Error State**, as defined in the CDAQ run sequencing note [7], are used. Other processes may have to implement the diagram or hold global states such as the MVD_SLT process. The error handling described in the note defines an ERROR state, which is entered on an error occurring. In the MVD software the error condition is a qualifier to the current state or transition. In the following sections the transitions and states will be described in detail.

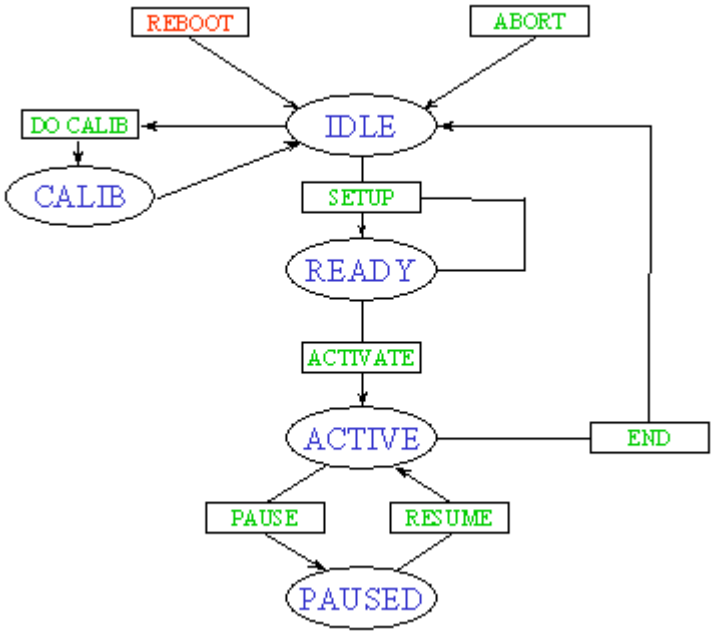

Figure 12 MVD RC state transition diagram. Ovals are states and transitions are boxes.

6.2 IDLE

The IDLE state is reached after process startup, ABORT or END. In the startup case some system initialization (finding communication partners, initializing required hardware, etc.) may occur. The error qualifier can be set whilst in IDLE.

6.3 DO CALIB

The transition DO CALIB can invoke any inline activity. In the specific case of inline standalone calibration runs the transition is used to setup and start a calibration run.

6.4 CALIB

During the CALIB state the calibration run (or any inline activity) initiated by DO CALIB is performed.

Inline actions have not been implemented within the framework of the other states and transitions because of the requirement of being able to identify and terminate them when required. DO CALIB will have to implement most activities normally associated with SETUP and ACTIVATE and CALIB will handle END and ABORT.

Running the DAQ with the CC generating a local clock and cycling through runs with differing chain configurations to identify faulty chips and chains is a example of a non calibration inline activity.

6.5 SETUP

The SETUP transition is used to move from IDLE or READY to READY. The implementation of the actions required can depend on the current configuration and qualifiers are needed to control what is to be done. For example a warm setup action can return because the result corresponds to the current configuration, whereas a cold setup action would always reconfigure. Note that the SETUP command

can be issued from READY.

The setup would involve:

- MVD-RC broadcasts SETUP to all sub-systems if the global state is IDLE and the global error state is ok.
- MVD-CC carries out the required CC SETUP, by:
    - Masterbox
        - Initialize it
        - set run-type
        - …
    - the Helix interface
        - download control information to the Helix chips
        - …
    - the ADC interface modules.
        - …
- MVD-ADC carries out the required ADC SETUP, by:
    - Reset counters and memory buffers.
    - Download pedestals, bad trip, cluster threshold, constants.
    - …
- MVD-EVBI carries out the required SETUP, by:
    - Initialize TPM memory buffer
    - Handshake with EVB
- MVD-GSLTI carries out the required SETUP, by:
    - Reboot GSLT communication TP.
    - Configure SLT receive GSLT decision mechanism.
    - …
- MVD-SLT carries out the required SETUP, by:

- Initialize the SLT system
- …

- MVD-SC carries out the required SETUP, by:
  - Turning on things that are off
  - …

Transition completion status (success or error) notified to CDAQ-RC, LOCAL-RC and participating sub-systems.

6.6 READY

The READY state is reached from IDLE or READY by SETUP. Traditionally nothing happens in READY, but MVD-RC will broadcast error state changes to CDAQ-RC and the participating sub-systems.

6.7 ACTIVATE

The ACTIVATE transition traditionally carries out few actions. The handling of the Beginning Of Run records (BOR) is required.

The activate would involve:

- MVD-RC broadcasts ACTIVATE to all sub-systems if the global state is READY and the global error state is ok.
- MVD-CC carries out the required ACTIVATE, by:
  - CC specific BOR data (if any) created.
  - …
- MVD-ADC carries out the required ACTIVATE, by:
  - ADC specific BOR data created.
  - …
- MVD-EVBI carries out the required ACTIVATE, by:
  - …
- MVD-GSLTI carries out the required ACTIVATE, by:
  - …
- MVD-SLT carries out the required ACTIVATE, by:
  - …
- MVD-SC carries out the required ACTIVATE, by:

    - SC specific BOR data created.
    - …
- MVD-EVBI receives BOR accept trigger from EVB
- MVD-EVBI puts BOR data into TPM.
- MVD-CC receives INIT from GFLT
- CC de-asserts FLT BUSY.

Transition completion status (success or error) notified to CDAQ-RC, LOCAL-RC and participating sub-systems.

6.8 ACTIVE

The ACTIVE state is reached from READY by ACTIVATE. MVD-RC will broadcast error state changes to CDAQ-RC and the participating sub-systems.

Active is usually characterized by data being taken and the following may be happening:

- On each HERA or CC clock signal:
    - The CC distributes the clock to the HELIX chips.
    - …

On the ADC Read-Out side, in order to be as fast as possible maintaining syncronized data gathering, buffering and error handling, the CPUADC needs to be interrupt driven by an additional board that could be either the ADC itself or the CC interface system. More specifications are needed. The basic idea is that:

- An interrupt I1 has to be generated on ADC data ready.
- An interrupt I2 has to be generated on ADC readout error.

Note that the Fast–Clear ABORT signal is handled by the ADC before data ready.

Looking more into details:

- On GFLT accept CPUADC is interrupted (I1) and
    - gathers and buffers SLT data from ADCs.
    - Extracts the readout type.
    - Check error bank among ADC's for consistency (FLT#, run_type).

- Act depending on read-out type (test triggers, ENV etc.)
- Send ADC data to MVD-SLT. (What to do in case of errors?)
- …

- CPUADC is interrupted (I2) and Error is handled:
  - Fatal error (e.g.…) or non-fatal error rates too high. → Stop the run.
  - Non fatal error (e.g. ADC trailer mismatch) try to re-initialize by:
    - Set hold request (wired or) in CC ADC interface register seen by Masterbox.
    - CC asserts busy after next GFLT accept allowing ADC to empty the Helix buffers.
    - Initiates re-initialization procedure.
    - Set release request (wired or) in CC ADC interface local CC interface.
    - CC de-asserts busy and data taking continues.
    - Update read-out tables; after the error condition has been cleared it might be necessary to flag the data as invalid till next ENV trigger arrival in order to have the correct read-out tables.
    - …

- SLT receive data and compute result to MVD-EVB
- MVD-EVB:
  - Buffer MVD-SLT result.
  - Send result to GSLT.
  - On every GSLT accept:
  - GSLTI receives GSLT decision.
  - Broadcast decision to MVD-ADC
  - Skip data if not accept
  - Forward data to EVBI if accept.
  - Format data if required and insert in EVB TPM.

6.9 PAUSE

The PAUSE transition takes the system from ACTIVE to PAUSED in LOCAL-RC mode.

The CDAQ-RC implements PAUSE by disabling GFLT triggers, but does not broadcast PAUSE to the participating components. Thus CDAQ generated PAUSE, RESUME and PAUSED are never seen.

The MVD should implement, in LOCAL-RC mode, PAUSE, PAUSED and RESUME. The PAUSE will disable the CC clock.

Transition completion status (success or error) is notified to CDAQ-RC, LOCAL-RC and participating sub-systems.

6.10 PAUSED

The PAUSED state is reached from ACTIVE by PAUSE in LOCAL-RC mode.

6.11 RESUME

The RESUME transition takes the system from PAUSED to ACTIVE in LOCAL-RC mode.

Transition completion status (success or error) is notified to CDAQ-RC, LOCAL-RC and participating sub-systems.

6.12 END

The END transition takes the system from ACTIVE to IDLE. The handling of the End Of Run records (EOR) is required.

The activate would involve:

- MVD-RC broadcasts END to all sub-systems if the global state is ACTIVE and the global error state is ok.
    - MVD-CC carries out the required END, by:
    - CC specific EOR data (if any) created.
    - …
- MVD-ADC carries out the required END, by:
    - ADC specific EOR data created.
    - …
- MVD-EVBI carries out the required END, by:
    - …
- MVD-GSLTI carries out the required END, by:
    - …
- MVD-SLT carries out the required END, by:

    - …
- MVD-SC carries out the required END, by:
- SC specific EOR data created.
    - …
    - MVD-EVBI receives EOR accept trigger from EVB
    - MVD-EVBI puts EOR data into TPM
- Statistics, DQM histograms, tables are saved and run-database is updated.

Transition completion status (success or error) is notified to CDAQ-RC, LOCAL-RC and participating sub-systems.

6.13 ABORT

The ABORT transition is guaranteed to move the system to IDLE, from any state or transition.

- MVD-RC broadcasts ABORT to all participating components and:
    - All sub-systems move to IDLE
- Statistics, DQM histograms, tables are saved and run database is updated as in END transitions.

Transition completion status (success) is notified to CDAQ-RC, LOCAL-RC and participating sub-systems.

7 Error conditions

In this section known error conditions [8] associated with the ADC sub-system are described. Additional information concerning other sub-system should be appended when available. Note that the hardware or software mechanism of identifying the error is not described, it is assumed that it can be done (see referenced material).

The correction procedure, or handling, of errors is described in section 8. The bracketed section heading following the description is the error recovery to be initially applied.

7.1 ADC sub-system

It is anticipated [9] that each ADC unit will be able to identify discrepancies in the arrival time and duration of the analogue data (see Figure 13). Additionally mismatches of the Helix pipeline column used within a token chain by an ADC unit. [8.1 Re-initializing Helix chains]

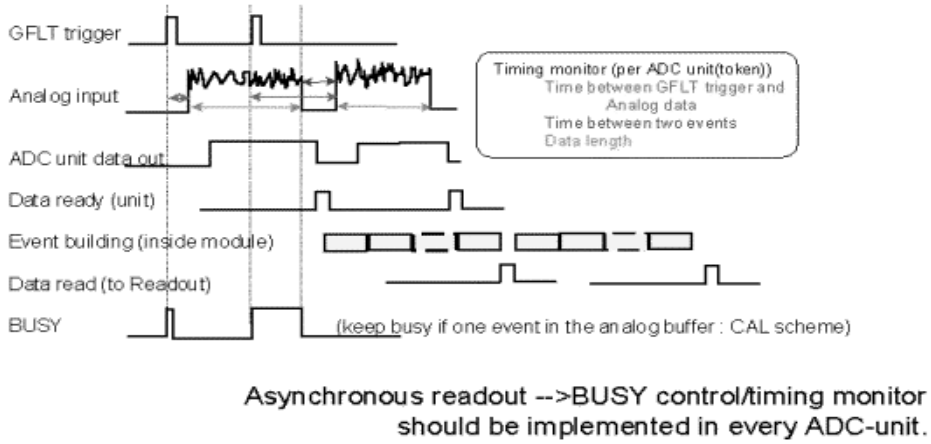

Figure 13 Data processing timing.

7.1.1 Timing errors

The time between a GFLT trigger and the arrival of analogue data, the **DataLatency**, is incorrect. [8.1 Re-initializing Helix chains].

The time lapse between two consecutive events, when the second event trigger is received during processing of the first, is different form the **TokenDelay**. [8.1 Re-initializing Helix chains].

The duration of **DataValid** is incorrect for the data associated with a single event. [8.1 Re-initializing Helix chains].

7.1.2 Mismatch errors

A mismatch exists between the used Helix chip pipeline column, as specified in the trailer, within a token chain.

7.2 Next...

8 Error handling

The error recovery procedures applied should be aimed at returning the system to a working state without user intervention (starting new runs etc.).

A low rate of errors can be handled by simply applying the appropriate recovery procedure. If the error rate increases above a given threshold it may be necessary to carry out additional procedures.

8.1 Re-initializing Helix chains

This procedure applies to all the token chains of the detector and consists of stopping the trigger, clearing out the readout pipeline, resetting the readout chip pipelines and resuming the trigger. The actions required are:

- Re-initialization is requested by the CPUADC (from OPERATOR or ADC I2 interrupt)
- CPUADC sets error request register in CC ADC interface card.

- CC sets GFLT busy on next FLT
- CPUADC senses pipeline empty (GFLT busy and no data) and resets error request register.
- CC issues a **NotReset** to reset the pipelines on the Helix chips.
- CC resets GFLT busy.

8.2 Remove a chip from a chain

This procedure might be invoked if a chip(s) can be identified as responsible for a high rate or errors. The actions required are:

- Removal is requested by the CPUADC (from OPERATOR).
- CPUADC sets error request register in CC ADC interface card.
- CC sets GFLT busy on next FLT
- CPUADC senses pipeline empty (GFLT busy and no data) sends network request to CPUCC to reconfigure chain.
- CPUCC sends reconfigure finished message to CPUADC
- CPUADC updates ADC configuration tables and stores change in ENV.
- CPUADC resets error request register.
- CC issues a **NotReset** to reset the pipelines on the Helix chips.
- CC resets GFLT busy.

8.3 Disable a chain at the ADC unit level

This procedure might be invoked if a chip removal would leave the chain inoperable.

- Disable is requested by the CPUADC (from OPERATOR).
- CPUADC sets error request register in CC ADC interface card.
- CC sets GFLT busy on next FLT
- CPUADC senses pipeline empty (GFLT busy and no data) and sets flag in ADC to disable chain
- CPUADC updates ADC configuration tables and stores change in ENV.
- CPUADC resets error request register.

- CC issues a **NotReset** to reset the pipelines on the Helix chips.
- CC resets GFLT busy.

8.4 Download ADC constants whilst ACTIVE

This procedure might be invoked if the ADC constants needed updating.

- Update is requested by the CPUADC (from OPERATOR).
- CPUADC sets error request register in CC ADC interface card.
- CC sets GFLT busy on next FLT
- CPUADC senses pipeline empty (GFLT busy and no data) downloads constants.
- CPUADC updates ADC configuration tables and stores change in ENV.
- CPUADC resets error request register.
- CC issues a **NotReset** to reset the pipelines on the Helix chips.
- CC resets GFLT busy.

---

[1] Clock and Control from 28/11/97 (J. Butterwoth's web page)?

[2] Optical fiber link description (L. Wigger's web page)?

[3] ADC sub-system description (ask Katsuo which one).

[4] Programming techniques for Remote Procedure Calls. J.R.Corbin

[5] ZEUS Note 94-150

[6] ZEUS Note 97-003

[7] ZEUS-Note 95-008

[8] Minutes of the CC-ADC meeting Oct. 97. (R.Carlin)

[9] K. Tokushuku Note Oct. 97