DAQ/Online and the DHCAL

Paul Dauncey

Overview of existing DAQ/online system
H/w requirements for DHCAL
S/w requirements for DHCAL
DAQ overview

• Like most DAQ systems, based on a state machine
  • Transitions between states driven by transfer of “records”

• Records are both dynamical agents and data storage elements
  • Record type tells each node of system which state to go to
  • Record contains data needed by node to complete transition…
  • …and/or stores data generated by node during transition

• E.g. configurationStart record
  • Tells a node to configure some hardware
  • Contains configuration values to do this
  • Stores the configuration values read back by node as crosscheck

• Records are simple contiguous arrays in memory
  • Transmitted between PCs using sockets
  • Written to (and read from) “binary” raw data files
A run is set of nested levels of transitions

- runStart
  - configurationStart
    - slowReadout
    - acquisitionStart
      - spillStart
        - trigger
        - trigger...
      - spillEnd
    - transferStart
      - event
      - event...
  - transferEnd
  - acquisitionEnd
- configurationEnd
- runEnd
DAQ requirements/constraints

• **ECAL** very front end (VFE) ASIC
  • Preamp and CR-RC shaper, peaking time ~ 190ns
  • Sample-and-hold for analogue signal, timed for peak
  • No pipeline or buffer

• Requires
  • Total trigger latency to reach VFE ASIC < 190ns
  • Only one trigger in system at any time
  • Digitisation complete before next trigger allowed, takes ~ 200µs

• General system requirements
  • Trigger rate ~1kHz during spills (theoretical maximum ~ 5kHz)
  • Buffer 2000 events during spills
  • Overall average event readout rate ~100Hz
  • Handle ~50kBytes/event ~ 5MBytes/s
  • Partitionable for ECAL and HCAL to run separately or together
DAQ hardware layout

Minimum DAQ system

ECAL Crate
ECAL CPU
Control CPU
Gbit Switch
Network
3TB

HCAL Crate
HCAL CPU
Monitor CPU

HCAL Slow
ECAL Power
ECAL Stage
ECAL readout based on CRC

• “Calice Readout Card”; 9U VME board, 216×8=1728 channels
  • Modified from CMS silicon tracker readout FED board
  • Firmware in FE, BE, VME
DHCAL: Readout requirements

• Needs to be able to satisfy general DAQ requirements
  • Trigger at 1kHz
  • Buffer 2000 events
  • Read out event data at average 100Hz, including data transfer through hardware
• Also, must deliver all data for each trigger in order
  • Keep counter of triggers for synchronisation checks; read out at ~kHz
  • Must identify empty events; keep list of trigger timestamps?
• N.B. There is no FCT system in CALICE
  • All CRCs run independent clocks
  • Trigger rising edge provides system-wide synchronisation
DHCAL: Trigger issues

• How to handle trigger is major issue
  • Could use single CRC (readout TCMT also)
  • Make your own trigger handler

• CRC trigger handler has several features
  • Oscillator and software trigger
  • S/w programmable AND/OR combinations
  • Spill signal detection
  • Trigger input history ~6µs around trigger

• Trigger history used to remove events offline
  • If particle a few µs before trigger, significant signal from shaper remains
  • For RPCs, would also need much longer history ~10ms
  • Not clear this can be added to CRC handler
VME controller

- Currently use SBS620 VME-PCI bridges
  - HAL interface; easy to use
  - Slow for single word I/O; 10μs overhead per access
  - Not important for block transfer which is major part of ECAL data
- Wanted to try CAEN 2718 system
  - CMS have moved to these; unclear if better performance
  - Discovered last week; need modified VME firmware which is incompatible with the CALICE BE firmware
  - Not worth major BE firmware upgrade needed as benefit is uncertain
- If DHCAL will use CRC for trigger (and TCMT readout)
  - Needs a HAL-compatible controller; only choice is SBS620 (or 618)
  - Otherwise major rewrite of CRC-HAL interface class
- If not using CRC, then free to choose any VME controller
Software overview

• Runs are labelled by runTypes
  • E.g. emcNoise, beamData, ahcLedScan, etc.
  • Each has a version number to select choices, e.g. number of calibration points in scan

• Together, runType and version determine
  • Number of configurations in run
  • Number of acquisitions in each configuration
  • Number of spills and events in each acquisition

• No databases are used
  • Too inflexible and lack of widespread knowledge

• Code requires
  • Any standard C++ compiler on standard Linux PC
  • Installation of ROOT
  • VME interface; we use HAL (from CERN)
DHCAL: Required software

• Several pieces of code will be needed
  • Interfaces to existing DAQ well-defined

• **Hardware** communication and readout
  • New C++ class: DhcReadout.hh
  • New C++ classes: data storage in record

• **Conversion** from runType/version/configuration number to configuration setup
  • New C++ class: DhcConfiguration.hh

• Addition of **DHCAL runTypes** to central list
  • Modify C++ classes: DaqRunTypes.hh, DaqConfiguration.hh

• **Monitoring and analysis**
  • Online and semi-online: scope of work unclear but structure defined
  • Offline analysis and LCIO converter: ditto

• If implementing trigger, need to do most of the above again
**DhcReadout**

- **Hardware interaction class**
  - Gets a record indicating a transition and response appropriately

- **Major records** and how to handle them:
  - **runStart/End**: read all R/O values constant for the run, e.g. serial numbers, firmware versions, cable connector locations, etc.
  - **configurationStart**: write all R/W values for configuring hardware and read back for crosscheck
  - **acquisitionStart**: flush all buffers and zero all counters
  - **trigger**: read all unbuffered data related to the event
  - **event**: read all buffered data related to the event
  - **acquisitionEnd**: check for remaining buffered data
  - **configurationEnd**: read back all R/W values as crosscheck
  - **slowReadout**: read “slow” data; e.g. temperatures, voltages, currents, etc.

- Ideally, every byte accessible in hardware appears somewhere in the above list!
DhcConfiguration

- **Converts** between
  - runType, version and configuration number…
  - …and DHCAL configuration values
- **Resulting values** written into the record
  - No direct interaction with hardware
- **DhcReadout** only uses values it reads from record later
  - Ensures complete configuration always recorded for offline use
- **Actual implementation** is arbitrary
  - ECAL is mainly a large switch statement in code
  - AHCAL has flat files which it reads in
  - Offline only resulting values in record are visible
Partitioning

- System designed to be easy to split
  - Run ECAL and DHCAL independently standalone
  - Run both synchronised as one system
- Use same DHCAL code for both
  - Simply start different programs for each case
  - DHCAL debugging done on separate PC before ECAL arrives
  - First step; simple standalone program to drive DhcReadout
Summary

• The main issue to decide on is how to handle the trigger
  • May constrain the VME controller choice

• Amount of software needed is not huge
  • Main part can be written independently of the main DAQ
  • I will be available (at least remotely) to help get it going

• We have four running DAQ systems
  • DESY, CERN, NIU, UCL
  • Adding a fifth should not be a big problem…
  • …although the new hardware will be the biggest difference