

Database implementation for time dependent module by module gains

Ermias ATOMSSA

Stony Brook University

December 20, 2011

Database side: PdbHbdAdcCalib

- Members of this class:
 - float PedestalConst[NHBDPADS];
 - float PedestalErr[NHBDPADS];
 - float GainConst[NHBDPADS];
 - float GainErr[NHBDPADS];
- This class is designed to contain the gain and pedestals of each pad
- ps: After Irina's request, I've replaced all occurrences of the literal 2304 by a member static const called NHBDPADS. (not committed yet)
- This class is designed to be stored in table calibhbdadc of calibrations database

In memory (Hbd calibration code) :hbdAdcCalib

- Three bank ids are used in calibhbdadc
 - bank1: an instance of PdbHbdAdcCalib is used to store the 2304 pedestals
 - bank2: another instance of PdbHbdAdcCalib is used to store the 2304 pad by pad gains
 - bank3: yet another instance of PdbHbdAdcCalib is used to store 24 module by module gains
- This organization wastes a lot of space
 - In bank ids 1 and 2, only 1/4 of the values stored actually contain information
 - In bank id 3, only 24 out of the 2304 of 1/4th of the values stored contain any information

A new container class for database storage: PdbHbdModuleGains

- Temporary location
`/phenix/plhf/tujuba/diel/src/commit/offline/database/pdbcal/base/PdbHbdModuleGains.[cc,hh]`
- The new proposed class to contain time dependent module gains has
 - A vector of floats, which by design was made to grow only to 24 indexes one for each module
 - Two unsigned ints for the start clock tick of the validity range for the gains stored in it (the end is inferred from the next instance). The database can not store long long types, so the clock ticks have to be split into two ints.
- Multiple copies (one for each validity region) of this same object are stored as a vector using a single bank id.
- Irina already crated an independent table called `hbdmodulegain` to hold this object in the database

Implementation of the read back

- Temporary location `/phenix/plhf/tujuba/diel/src/commit/offline/packages/hbd/hbdAdcCalib.[cc,hh]`
- The code was rewritten to
 - Store as many copies of this new container as needed for each run. Function to read gain constant from the files provided by Mihail was added. One copy of `PdbHbdModuleGain` object is committed for each time validity range on the same bank id.
 - Fetch the new container objects from database, as many as there are stored
 - Point to the right container object for each event. The pointer is moved only when the new event is not in the same validity range as the previous event to save time. Each time this happens, the old arrays used for the correction are updated.
- The same architecture can be used for runs where we don't have time dependent calibrations by storing just one copy of the container object initialized to the constant gains.
- Still need to do some testing, but should be ready to commit in a few days.

Questions

- Are the clock ticks stored in the gain files that were supplied absolute clock ticks or relative clock ticks from the beginning of the run?
- Is there any thing that one can do to test the setup?
- Can I commit the constants and the code? (Once I finish testing...)
- One can save some space and gain efficiency by rewriting the code and constants for the rest of the calibration (pad by pad gains and pedestals) or just leave it as it currently is. Should we keep it the way it is now?