Conversion from text files to root files

This is not for physics data! Marlin is used in that case with slcio files. Use this only with txt files!

Without event building

The data is saved, by the Labview program that run the electronic setup, in txt ASCII format. This part of the code converts these files to more manageable root files. A choice can be made to perform **event building or not (so, no sorting by BXID**).

In order to have consistent data output format, you need to run the code in the convert_root_eventbuilding folder. This is not only necessary for physics data (cosmics, testbeam...) but also commissioning data.

Instructions

Here are some step-by-step instruction to run the program:

1. Go to convert_root folder

   ```
   cd YourDestinationFolderName/convert_root
   ```

   This folder contains two other folders: src/ and Debug/. The first one contains the source files and the second one contains the executables and the makefile to produce them.

   The ASCII files produced by the Labview do not contain any header giving any explanation of what is the format of the data, so you need to know it before hand. The data is usually delivered in two different formats, depending if the data is taking using the HBU DAQ or the HDMI DAQ connexion. For the HBU DAQ Labview, the text file contains 12 columns: `BunchXID, CycleNr, ChipID2, ASICNr, EvtNr, chn, TDC, ADC, xPos, yPos, Hit_Bit, Gain_Bit`

   `Bunch crossing ID`: clock count

   `Cycle Number`: readout cycle (the readout continues until all memory cells are filled, then the data needs to be processed and another readout cycle starts)

   `ChipID2`: identity of the chips, is not hardcoded in the chip itself but set in the slow control files (don't change it!)

   `ASICNr`: Position of the asic in the HBU (0-3). Not used.

   `EvtNr`: Memory cell.

   `chn`: channel (0-35)

   `TDC`: relative time of the hit within a BunchXID (uncalibrated)

   `ADC`: charge, in ADC counts, collected by the channel/memory cell in one hit (uncalibrated)

   `xPos, yPos`: position of the chip/channel, not filled.

   `Hit_Bit`: 1 if the channel signal was over the threshold, 0 if not. In laboratory mode (forced trigger) is always 1.

   `Gain_Bit`: 1 if the ADC is delivered in High_gain mode or 0 if is in low gain.

   The data format which depends of your labview. Here you have the two standard formats.

   for HDMI DAQ: `sscanf(line.c_str(), "%i %i %i %i %i %i %i %i %i", &CycleNr, &BunchXID, &ChipID2, &EvtNr, &chn, &TDC, &ADC, &Hit_Bit, &Gain_Bit);`

   for USB DAQ: `sscanf(line.c_str(), "%i %i %i %i %i %i %i %i %i %i %i %i", &BunchXID, &CycleNr, &ChipID2, &ASICNr, &EvtNr, &chn, &TDC, &ADC, &xPos, &yPos, &Hit_Bit, &Gain_Bit);`
2. Compile the code

```
cd YourDestinationFolderName/convert_root/Debug
make clean
make convert_root
```

3. Finally, you can run the program. It only needs two arguments: folder where the text files and the folder where the rootfiles will be (you need to create it).

```
./convert_root [INPUT FOLDER] [OUTPUT FOLDER][Data format: USB or HDMI]
```

Output data format

The output is a root file with TTree that contains an entry for each line of data (an entry for each channel that has been stored in the data file). In LED runs with forced trigger in high gain mode, the Hit_bit and Gain_Bit will be always 1 and all 16 memory cells are filled for all chips even though the memory cell 0 (16 if we use USB labview) is filled with zeros.

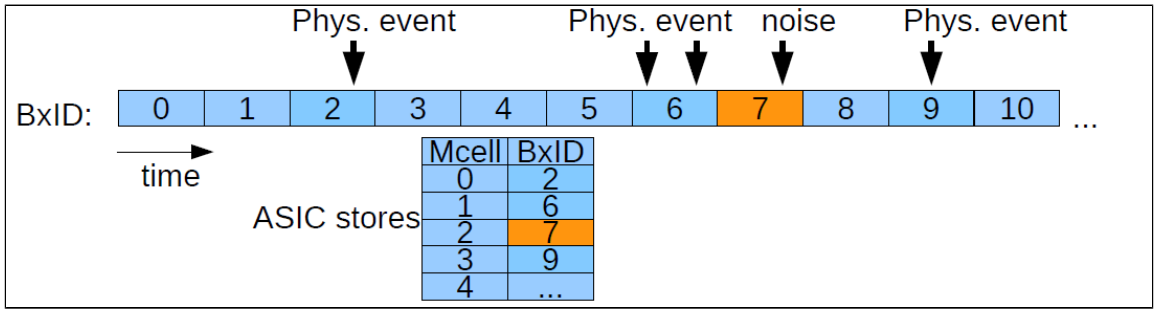
```
int CycleNr = 0;
int BunchXID = 0;
int ChipID2 = 0;
int ASICNr = 0;
int EvtNr = 0;
int chn = 0;
int TDC = 0;
int ADC = 0;
int xPos = 0;
int yPos = 0;
int Hit_Bit = 0;
int Gain_Bit = 0;

TFile* output_tree_file = new TFile(output_file.c_str(),"RECREATE");
TTree *myTree = new TTree ("tree", "tree");
myTree->Branch ("CycleNr",&CycleNr,"CycleNr/I");
myTree->Branch ("BunchXID",&BunchXID,"BunchXID/I");
myTree->Branch ("ChipID2",&ChipID2,"ChipID2/I");
myTree->Branch ("ASICNr",&ASICNr,"ASICNr/I");
myTree->Branch ("EvtNr",&EvtNr,"EvtNr/I");
myTree->Branch ("chn",&chn,"chn/I");
myTree->Branch ("TDC",&TDC,"TDC/I");
myTree->Branch ("ADC",&ADC,"ADC/I");
myTree->Branch ("xPos",&xPos,"xPos/I");
myTree->Branch ("yPos",&yPos,"yPos/I");
myTree->Branch ("Hit_Bit",&Hit_Bit,"Hit_Bit/I");
myTree->Branch ("Gain_Bit",&Gain_Bit,"Gain_Bit/I");
```

With event building

When analyzing physics data is important to convert the text files into root file performing, at the same time, the Event Building. Due to the characteristics of the Spiroc readouts, the output is not classified by events (noise or real hit events) but by readout cycle. Let's, for the moment, consider that we only have one ASIC and that we are running in autotrigger mode:

- Whenever 1 channel triggers (autotrigger), all 36 channels are stored in the analogue memory cells (at the end of the BxID latest).
  - We can have noise events
  - We can also, sometimes, miss an event (if one occur at the beginning of the BxID and other at the end, we miss the one at the end)



Acquisition continuous until all memory cells are filled. Then BUSY signal is set and the detector goes blind for up to ~50 ms due to conversion and slow readout. This makes unpredictable the time between readout cycles.

Now, let's consider the more realistic case: we run several chips with the same DAQ interface boards (specifically with same DIF). In this case, the data packets come out-of-order making a bit more complicated the sorting of the events by bunch crossing id.

This is done by the convert_root_eventbuilding code.

Instructions

1. Compile the code

```
cd YourDestinationFolderName/convert_root_eventbuilding/Debug
make clean
make all
```

2. Run the program. It only needs two arguments: folder where the text files and the folder where the rootfiles will be (you need to create it).

```
./convert_root [INPUT FOLDER] [OUTPUT FOLDER][Data format: USB or HDMI]
```

Output data format

The final tree contains one entry for every event (noise or real event). Most of the objects are vectors with a maximum size of nHits == total number of channels in our setup. Remember that our ASICs, when one single channel is over the threshold all channels are readout. To know if a channel was triggered or not (by autotrigger) you need to check the value of the Hitbit (==1 if autotriggered, ==0 if other channel was triggered).

```
int nHits = 0;
int iEvt = 0; //global event number , after sorting

const static unsigned int MAXCELLS  = 10000; /*should be big enough for all detectors!*/

int BunchXID[MAXCELLS];
int CycleNr[MAXCELLS];
int ChipID[MAXCELLS];
int EvtNr[MAXCELLS];
int Channel[MAXCELLS];
int TDC[MAXCELLS];
int ADC[MAXCELLS];
int HitBit[MAXCELLS];
int GainBit[MAXCELLS];

TFile* output_tree_file = new TFile(output_file.c_str(),"RECREATE");
TTree *myTree = new TTree ("myTree", "myTree");

TString _prefix ="ahcal_";

myTree->Branch( string(_prefix + "nHits").c_str(), &nHits,  string(_prefix+"nHits/I").c_str());
myTree->Branch( string(_prefix+"iEvt").c_str(), &iEvt, string(_prefix+"iEvt/I").c_str());

myTree->Branch( string(_prefix + "BunchXID").c_str(), &BunchXID, string(_prefix+"BunchXID["+_prefix+"nHits]
/I").c_str() );
myTree->Branch( string(_prefix + "CycleNr").c_str(), &CycleNr, string(_prefix+"CycleNr["+_prefix+"nHits]
/I").c_str() );
myTree->Branch( string(_prefix + "ChipID").c_str(), &ChipID, string(_prefix+"ChipID["+_prefix+"nHits]/I").
c_str() );
myTree->Branch( string(_prefix + "EvtNr").c_str(), &EvtNr, string(_prefix+"EvtNr["+_prefix+"nHits]/I").
c_str() );
myTree->Branch( string(_prefix + "Channel").c_str(), &Channel, string(_prefix+"Channel["+_prefix+"nHits]
/I").c_str() );
myTree->Branch( string(_prefix + "TDC").c_str(), &TDC, string(_prefix+"TDC["+_prefix+"nHits]/I").c_str() );
myTree->Branch( string(_prefix + "ADC").c_str(), &ADC, string(_prefix+"ADC["+_prefix+"nHits]/I").c_str() );
myTree->Branch( string(_prefix + "HitBit").c_str(), &HitBit, string(_prefix+"HitBit["+_prefix+"nHits]/I").
c_str() );
myTree->Branch( string(_prefix + "GainBit").c_str(), &GainBit, string(_prefix+"GainBit["+_prefix+"nHits]
/I").c_str() );
```