TT6 Analysis Package for Tracker Test Beam Analysis

(User's Guide)

Abstract

The TT6 analysis package for the Beam Test data is described in this note. The analysis algorithm and the framework is discussed in detail together with the general schema of the analysis.

Contact : CMS TIB Test Beam software developers (cms-tib-tb-dev@cern.ch)

1 Introduction

The performance of CMS Silicon Tracker Modules has been evaluated through a large number of tests using charged particle beams. The TT6 package was developed for the analysis of the data taken in beam tests. This package initially was written in FORTRAN language back in 1995 and has been evolved to fulfill the needs of module performance studies. Present version is written in C++ and it is a part of the Object Oriented Reconstruction and Analysis(ORCA) [1] framework. The TT6 analysis is one of the concrete implementation of the ApvAnalysis framework. The analysis algorithm is explained in Section 2, in Section 3 the analysis output is described and in Section 4 instructions and different options to run the analysis program are outlined.

2 Analysis Algorithm

The analysis consists of the calculation of Pedestal, Noise of individual strips and Common Mode of a group of strips in an iterative way. During the calculation, strips which have possible signal from the incident particle need to be discarded for that event.

2.1 Initial Pedestal and Noise (TT6PedestalCalulator)

The initial pedestal values are calculated using $\mathrm{N_{init}}$ events where $\mathrm{N_{init}}$ is a flexible number. For a given strip the initial pedestal is defined as the average ADC counts.

$\mathrm{Pedestal_{init}} = \sum_{\mathrm{i=1}}^{\mathrm{N_{init}}}(\mathrm{ADC_i})/\mathrm{N_{init}}$

and the noise is defined as the variance of the ADC counts

$(\mathrm{Noise_{init}})^2 =< (\mathrm{ADC_i})^2 >$

The events on each strip are sorted in ADC counts and 10% events with higher ADC counts are removed before doing the average to avoid signal events.

2.2 Updated Pedestal and Raw Noise (TT6PedestalCalulator)

The Pedestal and the Raw Noise are calculated in iterative way using $\mathrm{N_{iter}}$ events after cumulation where $\mathrm{N_{iter}}$ is a flexible number.

$\mathrm{Pedestal} = \sum_{\mathrm{i=1}}^{\mathrm{N_{iter}}}(\mathrm{ADC_i})/\mathrm{N_{iter}}$

$(\mathrm{RawNoise})^2 =< (\mathrm{ADC_i})^2 >$

During this calculation possible signal events are rejected using a cut

$(\mathrm{ADC} - \mathrm{Pedestal}) > \mathrm{n_{signal}} \times \mathrm{RawNoise}$

where Pedestal and RawNoise are used from the previous iteration (from initial values in case of first iteration).

2.3 Common Mode

The common mode is the instantaneous electronic condition and is defined for a group of strips connected to same chip. The Common Mode is calculated event by event. The number of strips, N_{strip}, in the group is variable parameter in the analysis. In the TT6 analysis the Common Mode is defined in two ways ;

(a) Mean Common Mode (TT6CommonModeCalculator): defined as the weighted average over a group of strips and the weight is defined from the noise.

$\mathrm{CommonMode} = (\sum_{\mathrm{i}=1}^{\mathrm{N_{strip}}} (\mathrm{ADC_i} - \mathrm{Pedestal_i})/\mathrm{Noise_i^2})/(\sum_{\mathrm{i}=1}^{\mathrm{N_{strip}}} \mathrm{Noise_i^2})$

In this calculation the possible signal strips are avoided using $(\mathrm{ADC} - \mathrm{Pedestal}) > \mathrm{n_{signal}} \times \mathrm{Noise}$

(b) Median Common Mode (MedianTT6CommonModeCalculator): defined from the median of the $(\mathrm{ADC_i} - \mathrm{Pedestal_i})$ values of the group of strips.

2.4 Common Mode Subtracted Noise (TT6NoiseCalculator)

The common Mode subtracted noise is defined from the variance of the quantity (ADC - Pedestal -Common Mode). This is also defined in the iterative way from a group of N_{iter} events for a given strip and avoiding signal events. $\mathrm{Noise_{CM}}^2 =< (\mathrm{ADC_i} - \mathrm{Pedestal} - \mathrm{CM_i})^2 >$

2.5 Noisy and Dead Strips (TT6ApvMask)

The faulty strips are classified as noisy or dead accordingly from their noise behavior. A strip is defined 'dead' when

$\mathrm{Noise_{CM}} < \mathrm{F_{dead}} \times < \mathrm{Noise_{CM}} >$

where $\mathrm{F_{dead}}$ is the percentage cut and $< \mathrm{Noise_{CM}} >$ is the average noise over all strips connected to a given readout chip.

Noisy strips are defined in three ways

(i) from rms cut : $\mathrm{Noise_{CM}} >< \mathrm{Noise_{CM}} > +\mathrm{F_{noisy}} \times \sigma_{\mathrm{noise}}$

here σ_{noise} is the rms spread of the noise over strips in given readout chip. The rms and the $< \mathrm{Noise_{CM}} >$ is calculated after a truncation, $\mathrm{F_{truncation}}$, of the noise values.

(ii) from percentage cut : $\mathrm{Noise_{CM}} > \mathrm{F_{noisy}} \times < \mathrm{Noise_{CM}} >$

(iii) fixed cut : $\mathrm{Noise_{CM}} > \mathrm{F_{noisy}}$

The strips identified as noisy or dead are then masked accordingly. The strip mask is defined at each iteration of the analysis. For noisy and dead strips the masks are 2 and 3 respectively and for good strips it is 1. This mask is used in subsequent common mode calculation and cluster finding to avoid a faulty strip. The mask is calculated from $\mathrm{Noise_{init}}$ and $\mathrm{Noise_{raw}}$ as well. But in order to identify the source of faulty strip different flag values are used here. Noisy and dead strips are marked by 5/4 from $\mathrm{Noise_{init}}$ and by 7/8 from $\mathrm{Noise_{raw}}$.

2.6 Cluster Finding

The clusters are searched in all strips after subtracting pedestal and the common mode from the ADC counts of strips. The cluster is selected using a three threshold algorithm, where after finding a seed with seed threshold the neighbouring strips are searched with neighbour threshold. Finally overall cluster threshold is used to select the cluster.

Instead of the default standard ORCA clusterizer, a TT6Clusterizer can be used which allows a maximum of one hole due to masked strips in a cluster.

3 Analysis Output

The useful informations from analysed data are stored in form of histograms as well as ntuples. This output can be produced either using HBOOK [2] or using ROOT [3] packages.

Histograms are available in the readout chip level as well as in the module level. The booking and filling of histograms are done in the classes TT6HistosForAPV (or TT6RootHistosForAPV) and TT6HistosForModule (or TT6RootHistosForModule).

The ID's for the HBOOK histograms are defined in such a way that contains module number(m) and the number of the readout chip(i). The Root histograms are self sufficient by their names. The histograms from individual module are stored in directories with respective name module number ("Module_1", "Module_2".....). The full list of histograms follows

```
Histogram Type              ID for HBOOK          ID for ROOT

Pipeline Address           m0i0010 (i-th chip)   M_m_APV_i_pipeLine_address
APV Error Bit              m0i0020 (i-th chip)   M_m_APV_i_error_bit
APV Digital Zero and One   m0i0030 (i-th chip)   M_m_APV_i_zero_one

Initial Pedestal vs Strips m0i0101 (i-th chip)   M_m_APV_i_init_pedestal_vs_strip
                           m000101 (full module) M_m_init_pedestal_vs_strip
Initial Pedestal           m0i0102 (i-th chip)   M_m_APV_i_init_pedestal
                           m000102 (full module) M_m_init_pedestal

Pedestal vs Strips         m0i0301 (i-th chip)   M_m_APV_i_pedestal_vs_strip
                           m000301 (full module) M_m_pedestal_vs_strip
Pedestal                   m0i0302 (i-th chip)   M_m_APV_i_pedestal
                           m000302 (full module) M_m_pedestal

Initial Noise vs Strips    m0i0401 (i-th chip)   M_m_APV_i_init_noise_vs_strip
                           m000401 (full module) M_m_init_noise_vs_strip
Initial Noise              m0i0402 (i-th chip)   M_m_APV_i_init_noise
                           m000402 (full module) M_m_init_noise

Raw Noise vs Strips        m0i0501 (i-th chip)   M_m_APV_i_rawnoise_vs_strip
```

```
 (profile)                  m000501 (full module) M_m_raw_noise_vs_strip
Raw Noise vs Strips         m0i0503 (i-th chip)   M_m_APV_i_rawnoise_vs_strip_lastit
(last iteration)            m000503 (full module) M_m_rawnoise_vs_strip_lastit
Raw Noise                   m0i0502 (i-th chip)   M_m_APV_i_rawnoise
                            m000502 (full module) M_m_rawnoise

CM Noise vs Strips          m0i0601 (i-th chip)   M_m_APV_i_cmnoise_vs_strip
(profile)                   m000601 (full module) M_m_cmnoise_vs_strip
CM Noise vs Strips          m0i0603 (i-th chip)   M_m_APV_i_cmnoise_vs_strip_lastit
(last iteration)            m000603 (full module) M_m_cmnoise_vs_strip_lastit
CM Noise                    m0i0602 (i-th chip)   M_m_APV_i_cmnoise
                            m000602 (full module) M_m_cmnoise
Common Mode                 m0i0800 (i-th chip)   M_m_APV_i_cmode
Common Mode vs Chip#        m000800 (full module) M_m_cmode

Mask vs Strip               m0i0701(i-th chip)    M_m_APV_i_mask_vs_strip
                            m000701(full module)  M_m_mask_vs_strip
Mask                        m0i0702(i-th chip)    M_m_APV_i_mask
                            m000702(full module)  M_m_mask

Number of Clusters          m0i0901(i-th chip)    M_m_APV_i_ncluster
                            m000901(full module)  M_m_ncluster
Cluster Charge              m0i0902(i-th chip)    M_m_APV_i_charge_cl
                            m000902(full module)  M_m_charge_cl
Cluster Position            m0i0903(i-th chip)    M_m_APV_i_position_cl
                            m000903(full module)  M_m_position_cl
Cluster Width               m0i0904(i-th chip)    M_m_APV_i_width_cl
                            m000904(full module)  M_m_width_cl
Cluster Noise               m0i0905(i-th chip)    M_m_APV_i_noise_cl
                            m000905(full module)  M_m_noise_cl
Response($\eta$) Function   m0i0906(i-th chip)    M_m_APV_i_eta_cl
                            m000906(full module)  M_m_eta_cl
Cluster Shape               m0i0907(i-th chip)    M_m_APV_i_shape_cl
                            m000907(full module)  M_m_shape_cl
Charge On Seed              m0i0908(i-th chip)    M_m_APV_i_seed_charge_cl
                            m000908(full module)  M_m_seed_charge_cl
Charge On Neighbour         m0i0909(i-th chip)    M_m_APV_i_neighbour_charge_cl
                            m000909(full module)  M_m_neighbour_charge_cl
```

One can use the following macro to plot a given variable from the output root file.

```
void plot_variable(int imod, int iapv, const char* var_name, TFile* file) {
  if (!file) return;

  char hName[50], dName[10];
  sprintf(dName, "Module_%i", imod);
```

```
  if (iapv <= 0) {
    sprintf(hName, "%s/M_%i_%s", dName, imod, var_name);
  }
  else {
    sprintf(hName, "%s/M_%i_APV_%i_%s", dName,imod,iapv,var_name);
  }
  TH1F* fTh1 = dynamic_cast<TH1F*>(file->Get(hName));
  if (fTh1) {
    cout << "Plotting ==> " << hName << endl;
    fTh1->Draw();
  }
}
```

To plot Pedetal as a function of strip number for module#m and APV#n one has to execute at the root prompt

```
TFile* myfile = new TFile(``file name'')
.x plot_variable.C(m, n, ``pedestal_vs_strip'', myfile)
```

A Graphical User Interface has been developed using Root [3] GUI facilities [4]. This software among it's other functionalities provide an option to plot the output root file from TT6 software.

The cluster informations are stored in the HBOOK ntuple or in ROOT Tree depending on the chosen option. They are booked and filled in the classes TT6Histogrammer/TT6RootHistogrammer.

The structure of the ntuple is the following

```
*****************************************************************
* Var numb * Type * Packing *    Range     *  Block   *  Name    *
*****************************************************************
*      1   * U*4  *         *              * COMMON   * RUN
*      2   * U*4  *         *              * COMMON   * EVENT
*      3   * U*4  *         *              * COMMON   * SPILL
*      4   * R*4  *         *              * COMMON   * PLL_DELAY
*      1   * I*4  *         * [0,12]       * CLUSTER  * nModule
*      2   * I*4  *         *              * CLUSTER  * number_cl(nModule)
*      3   * R*4  *         *              * CLUSTER  * charge_cl(nModule)
*      4   * R*4  *         *              * CLUSTER  * position_cl(nModule)
*      5   * R*4  *         *              * CLUSTER  * noise_cl(nModule)
*      6   * U*4  *    5    * [0,20]       * CLUSTER  * width_cl(nModule)
*      7   * I*4  *         *              * CLUSTER  * maxstr_pos_cl(nModule)
*      8   * R*4  *         *              * CLUSTER  * maxstr_charge_cl(nModule)
*      9   * R*4  *         *              * CLUSTER  * charge_np_cl(nModule)
*     10   * R*4  *         *              * CLUSTER  * charge_nm_cl(nModule)
*      1   * I*4  *         * [0,50]       * ALLCLUST * totCluster
*      2   * U*4  *    4    * [0,12]       * ALLCLUST * moduleId(totCluster)
```

```
 *      3   * U*4  *    3    * [0,4]        * ALLCLUST * clusId(totCluster)
 *      4   * R*4  *         *              * ALLCLUST * clusCharge(totCluster)
 *      5   * R*4  *         *              * ALLCLUST * clusPos(totCluster)
 *      6   * R*4  *         *              * ALLCLUST * clusNoise(totCluster)
 *      7   * U*4  *    5    * [0,20]       * ALLCLUST * clusWidth(totCluster)
 *      8   * R*4  *         *              * ALLCLUST * clusEta(totCluster)
 *****************************************************************
```

In the first block (COMMON) the general event parameters are stored, in the second block (CLUSTER) the cluster quantites (charge, position ....) are stored for the most energetic cluster in each module. For detailed study up to 4 clusters, ordered in energy, are stored in the last block (ALLCLUST). A couple of examples are given below to plot

Cluster Charge of most energetic cluster in module #2 : nt/pl 1001.charge_cl(2) number_cl(2).gt.0

Cluster Charge of 2nd most energetic cluster in module #2 : nt/pl 1001.clusCharge moduleId.eq.2.and.clusId.eq.2

The Root Tree also contains similar variables and are stored in branch Clusters.

```
runNumber
eventNumber
bunchXNumber
pllDelay
nCluster
idCluster
idModule
clCharge
clPosition
clNoise
clWidth
clEta
clMaxPosition
clMaxCharge
clNeighbourChargeL
clNeighbourChargeR
```

The following macro can be used to plot basic cluster properties

```
#include <iostream>
void plot_cluster_properties(int m_id, int cl_id, TFile* file) {
  char hTag[10], hName[30],  hTitle[40], tSelect[50];
  sprintf(hTag, "M%i_Cl%i",m_id, cl_id);

  TCanvas* tc = (TCanvas*) gROOT->FindObject("c1");
  if (tc) tc->Clear();
```

```
TTree* fTree = (TTree*) file->Get("TT6Tree");
if (!fTree) return;

sprintf(tSelect, "idModule==%i && idCluster==%i", m_id, cl_id);

// Cluster Charge
sprintf(hName, "Charge_%s",hTag);
sprintf(hTitle, "Cluster Charge (%s)",hTag);
TH1F* fThCharge = (TH1F *) gROOT->FindObject(hName);
if (fThCharge) {
  fThCharge->Reset();
} else {
  fThCharge = new TH1F(hName, hTitle, 200,0.0,400.0);
}
fTree->Project(hName, "clCharge", tSelect);

// Cluster Width
sprintf(hName, "Width_%s",hTag);
sprintf(hTitle, "Cluster Width (%s)",hTag);
TH1F* fThWidth = (TH1F *) gROOT->FindObject(hName);
if (fThWidth) {
  fThWidth->Reset();
} else {
  fThWidth = new TH1F(hName, hTitle, 21, -0.5,20.5);
}
fTree->Project(hName, "clWidth", tSelect);

// Cluster Position
sprintf(hName, "Position_%s",hTag);
sprintf(hTitle, "Cluster Position (%s)",hTag);
TH1F* fThPosition = (TH1F *) gROOT->FindObject(hName);
if (fThPosition) {
  fThPosition->Reset();
} else {
  fThPosition = new TH1F(hName, hTitle, 512, 0.5, 512.5);
}
fTree->Project(hName, "clPosition", tSelect);
// Cluster Noise
sprintf(hName, "Noise_%s",hTag);
sprintf(hTitle, "Cluster Noise (%s)",hTag);
TH1F* fThNoise = (TH1F *) gROOT->FindObject(hName);
if (fThNoise) {
  fThNoise->Reset();
} else {
  fThNoise = new TH1F(hName, hTitle, 50,0.0,20.0);
}
```

```
 fTree->Project(hName, "clNoise", tSelect);


 TCanvas* tc = new TCanvas("Cluster_Canvas","Cluster Canvas",10,10,900,700);
 tc->Divide(2,2);
 tc->cd(1) ; fThCharge->Draw();
 tc->cd(2) ; fThNoise->Draw();
 tc->cd(3) ; fThWidth->Draw();
 tc->cd(4) ; fThPosition->Draw();
}
```

The properties of nth (<5) cluster for the module#m can be plotted by executing follwing commands at the root prompt

```
TFile* myfile = new TFile(``file name'')
.x plot_cluster_properties.C(m, n, myfile)
```

4 TT6 program options

The analysis algorithms described in section 2 are available in the "ApvAnalysisConcrete" sub-package of the "Tracker" package of ORCA [1]. The steering code to run TT6 analysis "TT6Analysis.cc" together with histogrammer and cluster finder are kept in the TestBeam/TkX5bMay2003 package. The analysis is triggered by default by the general purpose program "General.cpp", unless GenEventStudy:useTT6Analysis is set to false in the .orcarc file.

The analysis parameters which can be changed in the .orcarc file are listed here

Parameter	Default Value	Definition
GeneralApvAnalysisFactory : pedestalCalculator	TT6	Pedestal Calculator
GeneralApvAnalysisFactory : noiseCalculator	TT6	Noise Calculator
GeneralApvAnalysisFactory : cmnCalculator	MedianTT6	Definition of CM
GeneralApvAnalysisFactory : theNumCMstripsInGroup	128	N_{strip}
GeneralApvAnalysisFactory:maskAlgorithm	TT6	Mask Calculator
PedestalCalculator : eventsRequiredToCalibrate	500	N_{init}
PedestalCalculator : eventsRequiredToUpdate	500	N_{iter}
NoiseCalculator : eventsRequiredToUpdate	500	N_{iter}
PedestalCalculator : cutToAvoidSignal	3.0	n_{signal}
NoiseCalculator : cutToAvoidSignal	3.0	n_{signal}
TT6ApvMask : noisyStripDefinition	1	noisy definition
TT6ApvMask : cutForNoisyStrips	5.0	F_{noisy}
TT6ApvMask : cutForDeadStrips	0.5	F_{dead}
TT6ApvMask : cutForTruncation	0.05	$F_{truncation}$
TT6Cluster : FindCluster	0	TT6ClusterFinder/RecHit (1/0)
TT6Cluster : cutForSeedStrip	4.0	Seed Threshold
TT6Cluster : cutForNeighbourStrip	3.0	Neighbour Threshold
TT6Cluster : cutForClusterCharge	5.0	Cluster Threshold
TT6Cluster : NStripsToCheck	5	# of neighbours to search
TT6 : UseRootHistos	1	HBOOK/ROOT histograms (0/1)
TT6 : NtupleBookAndSave	1	book&fill the cluster Tree/Ntuple

References

[1] *CMS OO Reconstruction.* http://cmsdoc.cern.ch/orca

[2] Computing and Networks Division, *HBOOK Reference Manual*, CERN Program Library Long Writeups Y250.

[3] Rene Brun, Fons Rademakers *etal.*, *ROOT Users Guide.*, http://root.cern.ch/root/doc/RootDoc.html

[4] *CMS Tracker TIB Integration Test Software*, http://savannah.cern.ch/projects/cmstibintsft/