Publishing ATLAS CTP per-bunch-crossing ID and BPTX Monitoring Data

Erez Zimmerman

30th August 2018

Supervisors: Patrick Czodrowski, Antoine Marzin

Abstract

The ATLAS central trigger is responsible for forming the Level-1 trigger decision based on the information from the calorimeter and muon trigger processors, as well as forward detectors. As part of my project as a summer student at CERN, I was asked to develop a software application able to extract per-bunch-crossing ID and BPTX monitoring information from databases, and publish it in a useful way. To that end, I developed a python script that creates summary ROOT files that contain histograms from the monitoring data. Two subsequent scripts then create a useful and interactive webpage to publish the data. These scripts are to be run daily to achieve daily publication of the data.

1 Introduction

As part of the CERN summer student program, I worked for the Level 1 central trigger (L1CT) Group in the ATLAS experiment and was given a project to visualise and publish per-bunch-crossing ID and BPTX monitoring data. This data is useful to monitor potential malfunctions and configuration errors of the experiment. This report will include a short introduction of the central trigger processor (CTP), an introduction to the per-bunch-crossing ID and BPTX (Beam Pick-up Timing system for Experiments) monitoring information, and a review of the code and script I developed in the course of this project.

1.1 ATLAS Level-1 Central Trigger Processor [1]

The CTP is the component responsible for making the Level-1 Accept (L1A) decisions based on information from the calorimeter and muon trigger processors and send it to the high level trigger (HLT) and data acquisition system. In addition to making a L1A decision the CTP also creates a 8-bit trigger type word which identifies the type of trigger the L1A decision was made upon.

To maintain synchronisation of the CTP and the LHC, each trigger event is associated with two identifiers by the CTP; an event identifier (L1ID) and a bunch crossing identifier (BCID). The L1ID is formed in counting the L1A, while the BCID gives the corresponding bunch crossing (BC) number within one turn of the LHC. A bunch counter then resets every turn of the LHC to maintain synchronisation. Overall each turn of the LHC contains 3564 BCs.

In order to prevent front-end buffers overload the CTP also generates dead time in 2 different ways: a fixed number of BCs that would not trigger after each L1A and 4 leaky-bucket algorithms which limit the number of L1As in a given period of time.

The CTP is hardware based and made of several modules:

- up to 3 CTP_IN modules for input data
- CTP_CORE - a core module that makes trigger formations, read-out, time stamping and monitoring data
- CTP_MON - a bunch-by-bunch monitoring module
- up to 4 CTP_OUT - a modules that sends data to the sub-detectors
- CTP_MI - a machine interface module for timing signals
- CTP_CAL - a module that receives calibration requests from the detectors

In this project I was working on data from the CTP_MON module and the CTP_CORE module, both receiving signals from the CTP_IN module.

1.2 Per-bunch-crossing ID and BPTX monitoring

I was asked to work on two sources of monitoring data: CTPMon data, which is produced by the CTP_MON module in the CTP, and the per-bunch monitoring data, which is produced by the CTP_CORE module.

The CTPMon data is the monitoring data produced by the CTP_MON module which counts trigger information on a bunch-by-bunch basis. The data is saved in histograms of events per BCID, which are created for every lumi-block (LB, a time period of one minute in a run). The CTP_MON module has an output capability of up to 160 trigger inputs times 3564 bunch crossings.

The per-bunch monitoring (PBM) data is created in the CTP_CORE module. The data is taken by 64 PBM counters, of which 56 are sampled for every lumiblock (low frequency monitoring data) and 8 are sampled every 2 seconds (high frequency monitoring data). Again this data is saved as histograms of events per BCID. The PBM also contains high frequency CTP dead-time data and L1A rates, which are saved as histograms as well.

Both sources contain BPTX data as well, which is brought from a separate system - the ATLAS beam pick-up based timing system [2]. The BPTX system provides information about the beams pattern. The properties of the individual bunches are measured and the structure of the beams is determined. The data allows one to, for example, differentiate bunches with colliding beams.

Comparison between LHC past and ongoing LHC runs is useful to spot potential hardware failure or misconfiguration which could disrupt data-taking, such as synchronisation issues in the CTP.

2 PBM and CTPMon Data Acquisition and Visualisation

To create summarised monitoring histograms from the PBM and CTPMon data, I developed a python script that fetches the data from EOS, summarises it and creates PNG files with the summarised histograms. Two subsequent scripts create webpages for each run data takings, and a main index page for the user to navigate. The scripts are to be run daily using another script that runs them all together. In this section I will elaborate on the operation of these scripts:

2.1 Data Acquisition and Handling

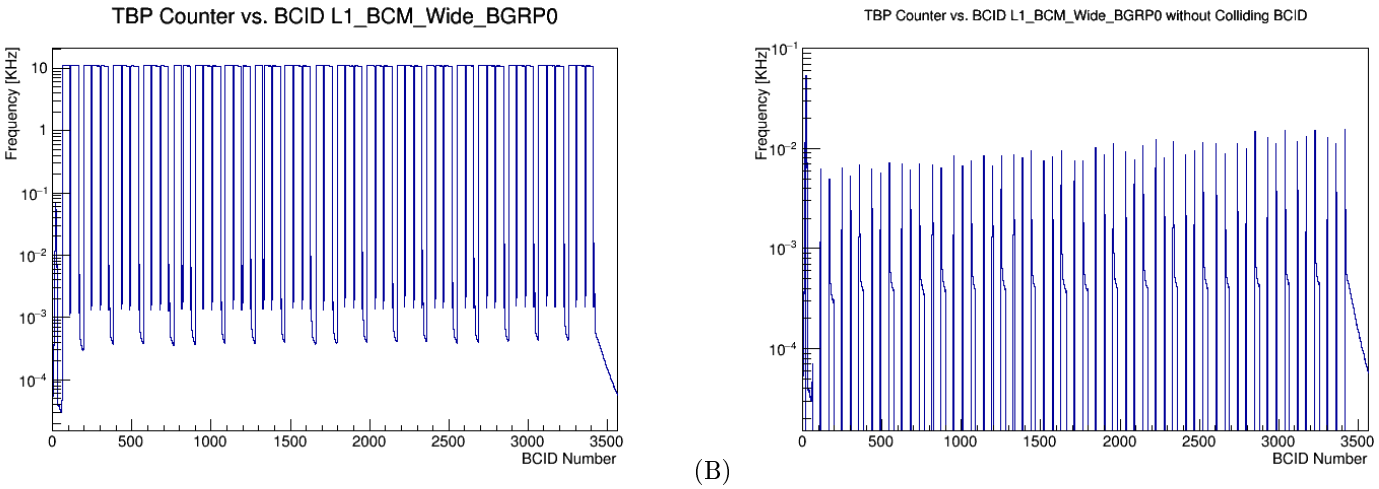
The raw CTPMon and PBM data are saved daily in a tar file to a folder in EOS [3]. These tar files contain ROOT [4] files corresponding to 5 minutes of data taking. These files contain all per-BCID histograms from the CTP monitoring data. by

the relevant modules. The histograms are sorted into different timestamped folders; in the case of the PBM, the histograms are also sorted into low frequency monitoring, high frequency monitoring and dead time per-bunch monitoring (DPM) folders.

To produce summarised data for each run, the script first sorts the different 5 minute ROOT files into runs. To do that the script uses the Beauty interface [5] to fetch information about the latest LHC runs from the pBeast [6] databases. This data contains the start and finish time of each run as well as information on which LBs in a run had stable beams in the LHC. Using the start and finish time of each run and the timestamp on each 5 minute ROOT file, the script then sorts the ROOT files into different run folders. The stable beam information is kept for later use. A summary of the amount of LBs in a run as well as the start and end date of the run is documented in a txt file by the script as well.

After sorting the ROOT files into folders, the script starts opening the ROOT files one by one for each run, reading each histogram's underflow bin. The underflow bin in the histogram is used to store the LB in which the events in the histogram occurred. The histogram's LB is then compared to the stable beam data acquired from pBeast and if the LB has a stable beam, the histogram's data is added to a summarised histogram in a new run-summary ROOT file.

After a summary ROOT file is created for each run, the script reads the BPTX histograms in the ROOT file and compares the BPTX data of both beams of the LHC, flagging BCIDs that have values (other than 0) in both beams' histogram as BCIDs with colliding beams. The script then creates new histograms excluding BCIDs with colliding beams in the summary ROOT file. This is useful since one expects no trigger events in BCIDs without colliding beams, and so these histograms allow the users to check the correct timing of the experiment.



(A) (B)

Figure 1: A sample histogram from the script (A) with colliding beams BCIDs and (B) without colliding beams BCIDs. One can notice the decrease in 3 orders of magnitude in the frequency of events between the two histograms.

The script keeps a record of runs and tar files it had already handled in a date log file, and will not handle previous runs. It is also worth mentioning that if the script cannot find a tar file for more than 2 days, a warning email will be sent to the group. Lastly, it is worth mentioning that as a user input the script can either update the webpage according to the last date in its date log or go over a user defined number of days of data.

2.2 Visualisation and Publishing

After the summary ROOT file is complete, the script creates PNG files for each histogram in the file. To do that the histograms are first scaled using the overflow bin in each histogram which represents the number of LHC turns during which the events were recorded to the histogram. The scaling for the DPM data is done so that it will represent the dead time percent using the following formula:

$$S = \frac{100}{N_{turns}}$$

where S is the scale factor and N_{turns} is the number of LHC turns in the histogram.

While for the rest of the data the scaling is done using the following formula:

$$S = \frac{f_{LHC}}{N_{turns}}$$

where S is the scale factor and, N_{turns} is the number of LHC turns in the histogram, and f_{LHC} is the LHC frequency of 11.2455 $[KHz]$.

After the histograms are created, a separate script creates a webpage for each run containing all the histograms in a useful way for the user. The webpage contains two columns of histogram, showing each histogram with colliding BCIDs side by side with its equivalent without the colliding BCIDs. The histogram's name is written above the figure, to allow the user to search for the histogram using the browser search bar. Lastly, by clicking on a histogram the user is transferred into another webpage, with a larger figure of the histogram as well as a cursor to allow the user to easily identify the value of each BCID.

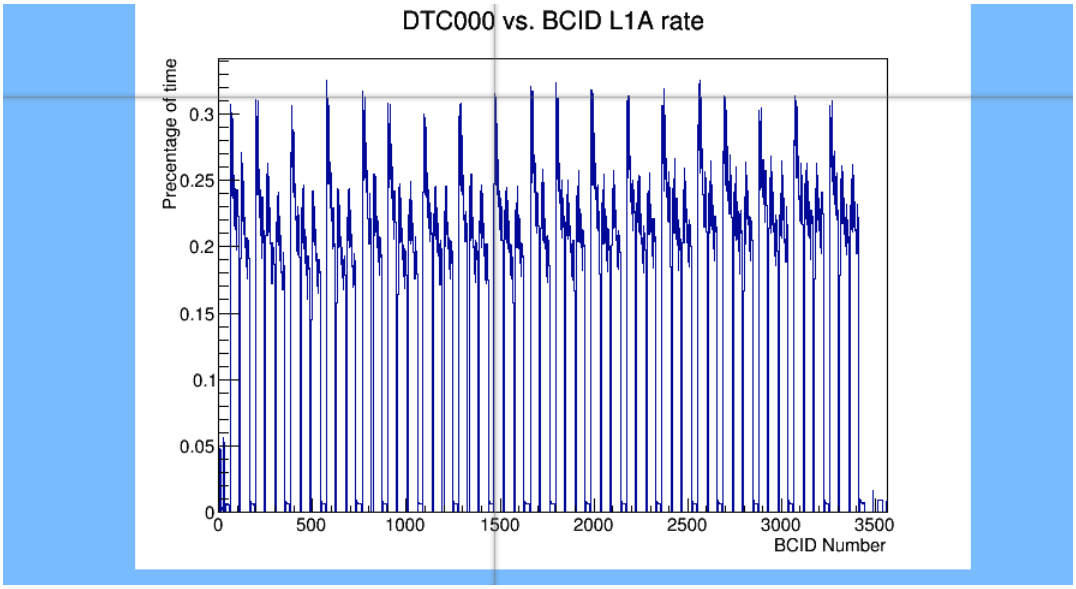


Figure 2: The Histogram page with a cursor.

In the case of PBM data, since 3 different data types exists, a checkbox is added to the webpage so that the user can choose which data to view. A download link to the summarised ROOT file is also available on the webpage for further investigation of the histograms by the user.

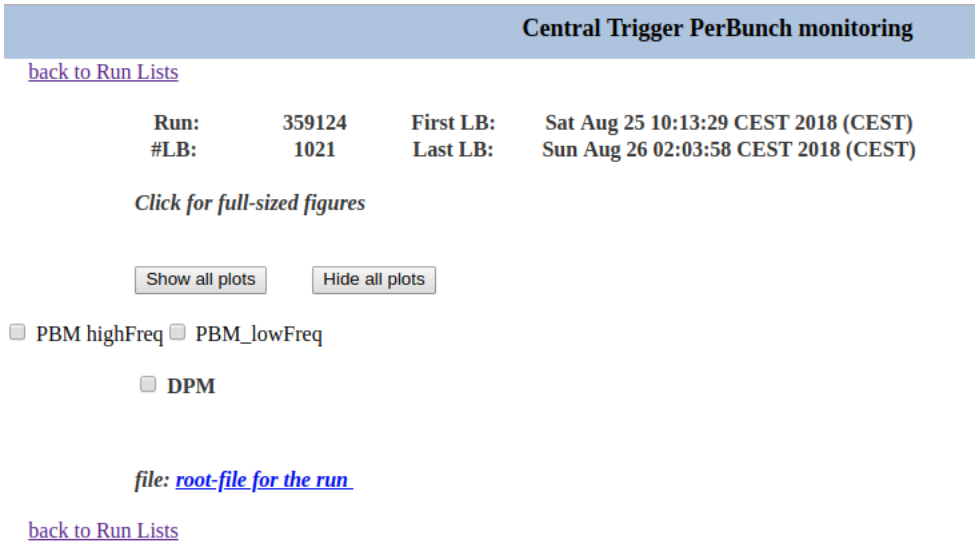


Figure 3: The PBM run webpage menu with the checkbox option. The user can choose which set of histograms to view by checking the relevant checkbox.

After all the run pages are created a last script goes over the run pages and creates a main index page for the user's convenience.

3 Conclusion

During the summer, as part of my project as a summer student at CERN I developed a program for summarising and publishing per-bunch-crossing ID and BPTX monitoring data for the CTP group in ATLAS. The program fetches the data from EOS, writes the useful data for each run into a summary ROOT file, and creates useful histograms with no colliding BCIDs in them. The histograms in the ROOT file are then published in a webpage daily, which will allow the CTP group to monitor the trigger rates and the timing of the experiment.

References

[1] The ATLAS central level-1 trigger logic and TTC system, S. Ask et al. JINST 3 P08002 (2008).

[2] The ATLAS beam pick-up based timing system, C. Ohm, T. Pauly, Nucl. Instrum. Meth. A 623 (2010) 558, arXiv:0905.3648v1 [physics.ins-det].

[3] EOS webpage, url: `http://information-technology.web.cern.ch/services/eos-service`.

[4] ROOT data framework, url: `https://root.cern.ch/`.

[5] W. Vandelli, Beauty project, url: `https://gitlab.cern.ch/vandelli/beauty`.

[6] A. Sicoe et al., A Persistent Back-End for the ATLAS TDAQ Online Information Service (P-BEAST), url: `https://cds.cern.ch/record/1402973/files/ATL-DAQ-PROC-2011-047.pdf`.

[7] CTP PBM Webpage, url: `https://atlas-l1ct.cern.ch/perbunch/`.

[8] CTP CTPMon Webpage, url: `https://atlas-l1ct.cern.ch/ctpmon/`.