Performance Evaluation of the ATLAS IBL Calibration

Performance Evaluation of the ATLAS IBL Calibration

M. Kretz[1] on behalf of the ATLAS collaboration

[1] ZITI Institut für technische Informatik, Ruprecht-Karls-Universität Heidelberg, Mannheim, Germany

E-mail: `moritz.kretz@cern.ch`

Abstract. The Insertable-B-Layer (IBL) has recently been commissioned at the ATLAS Experiment, adding 12 million channels to the existing Pixel Detector. The front-end chips (FE-I4) are connected to newly designed readout hardware situated in a VME crate. In order to take data under uniform conditions, one needs to periodically tune the detector in short breaks between data-taking sessions to accommodate for radiation damage and aging effects. Tuning involves a variety of components, ranging from high-level steering and analysis software (PixLib) running on commodity hardware, to embedded components situated inside the VME crate that feature only a minimal or no operating system at all. Understanding the interactions between these components is key in debugging and optimizing the tuning procedures to become more efficient.
We therefore implement an instrumentation framework aimed at all major components. It features a uniform interface to the user and is able to take instrumentation data with μs-precision. A central server application is used to gather the instrumentation data of a tuning session. It processes the data and saves it into an SQLite database for later analysis.

1. Introduction

Many new hardware and software components were developed for the IBL at the ATLAS Experiment [1, 2]. The existing calibration architecture was improved and the bottleneck of VME-based transfer of calibration histograms has been evaded by using Gigabit Ethernet network interfaces to ship data to a processing farm [3, 4]. Therefore, many parts of the already existing Pixel Detector calibration software needed to be adapted or newly written to handle the IBL without influencing the already existing system.
This involved adjusting and extending parts of the present Pixel software library (PixLib) and also implementing software from scratch for the new embedded hardware components being used in the detector readout and calibration.
As the time slots for detector calibration between data-taking runs at the LHC are sparse and valuable one is interested in making the calibration procedures as efficient as possible. In order to optimize and debug these operations, an instrumentation framework for the IBL is implemented that is able to supply timing information for the various components involved in calibration scans and tunings.

1.1. Detector Calibration

A regular detector calibration consists of a sequence of scans and tuning procedures that optimize the detector response for a data-taking run.
The most prominent parameters that characterize a pixel are its threshold value and the time over threshold (TOT) of a signal associated with a minimum ionizing particle passing through it. These parameters can be manipulated by changing front-end register values that affect the threshold and TOT on a per-front-end as well as a per-pixel basis. A calibration usually consists of initial scans in order to get the state of the detector before tuning, then a series of tunings that optimize the threshold and TOT response, followed by a group of scans to verify the results. Figure 1 shows an example of calibration scan histograms before and after tuning the modules.

2. Calibration Architecture Overview

In this section we will give a brief overview of the relevant hardware and software layers that are used for calibrating the IBL. Figure 2 depicts these components and their interaction.

2.1. Hardware Components

The IBL detector front-end chips are grouped in 14 staves, each containing 32 chips with 26880 pixels. The IBL VME crate in the counting room hosts 14 pairs of Back-Of-Crate (BOC) and Read-Out-Driver (ROD) cards, each pair servicing one stave.
While the BOC mainly acts as an optical interface to the detector front-ends and forwards decoded 8b/10b data to the ROD, the ROD re-formats the incoming event data, sends commands to the front-end chips, and performs histogramming tasks for detector calibration. The ROD features two slave FPGAs (Spartan-6) and one master FPGA (Virtex-5) that each have access to Gigabit Ethernet interfaces to efficiently communicate with the higher level parts of the infrastructure.
The slave FPGAs are used for data-processing. Their hardware design contains a MicroBlaze soft-CPU for sending out histogram data via the network as well as for control purposes. The master FPGA acts as a controller for the slaves and generates commands for the front-end

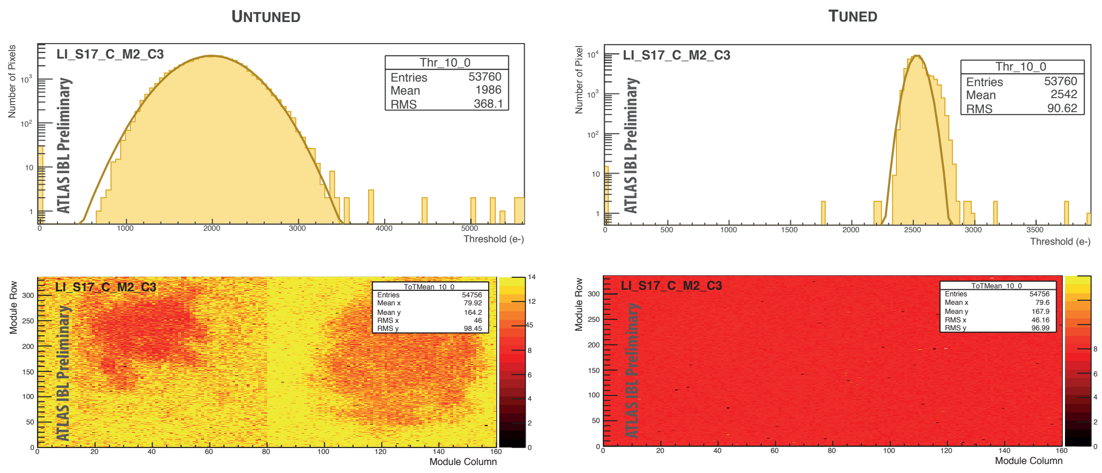


Figure 1. Scan results before and after tuning to a threshold of 2500 e- and TOT target value 8. Histograms on top display the threshold distribution, while 2D maps of the modules on the bottom show TOT values for a MIP charge deposit. Data are for one module consisting of two FE-I4 front-end chips. Scans were run in the Pixel test-stand on one stave. The noticeable tail in the threshold distribution might be avoided by running another tuning iteration.

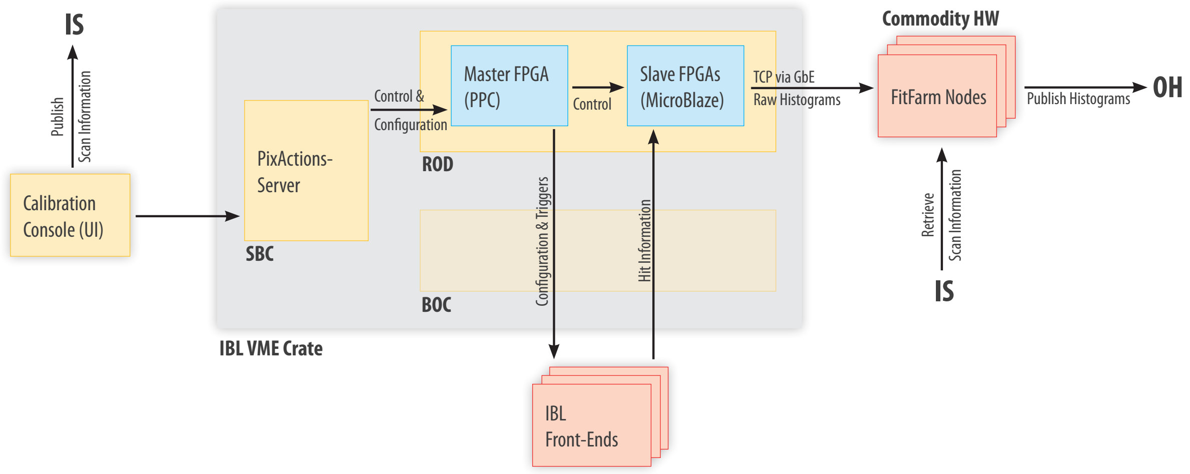


Figure 2. Simplified overview of IBL components involved in calibrating the detector.

modules. It features an integrated PowerPC (PPC) that runs the control software, receiving commands via the network.

The VME crate furthermore contains a Single-Board-Computer (SBC) that runs parts of the high-level PixLib framework. There are other server machines responsible for storing and retrieving front-end configuration data (PixDbServer), as well as processing calibration histograms (FitFarm).

2.2. Software Components
PixLib is based upon the ATLAS TDAQ framework ([5]). Besides supporting the data-taking operations of the detector, its main task is to provide the means for an efficient detector calibration.
The user interface for calibration (CalibrationConsole) is used by shifters and experts to run scans or tunings on the detector as well as look at archived results. Distributed server processes take care of sending commands to the master FPGAs on the RODs.
While the PPC features a minimalistic operating system (XilKernel) that supports, for example, multi-threading, the slave software for the MicroBlaze runs without an operating system layer. These components are independent from the higher-level PixLib framework. Basic scans are steered by the PPC software, yet more complex tunings require feedback from PixLib analysis processes. The slave software receives commands from the PPC to start and stop histogramming, buffers the gathered histogram data, and sends it to the FitFarm via a TCP network connection.

The FitFarm consists of one or more server processes that retrieve the calibration data from the RODs, re-format and process it, and finally publish the results to the Online Histogramming (OH) service, from where they are available to other parts of PixLib.

3. Performance Evaluation
As there are many potential parameters that influence the overall performance of the calibration procedure, it is challenging to get a coherent overview of the operation and the time spent for certain parts of it. For example, bottlenecks might exist due to hardware shortcomings in the FitFarm, network speed limitations, inefficient use of sleep commands, lock contention and so forth. Therefore we introduce a framework that enables instrumentation of relevant parts of the

software stack to gain a better understanding of the calibration procedure, as well as to optimize and debug it.

```
#include  PixUtilities/PixInstrument.h

/* Optionally initialize at non-critical code section. */
INSTRUM_INIT();
/* Default instrumentation point with unique integer id and description. */
INSTRUMPOINT(id, description);
/* Logs only every n-th event. */
INSTRUMPOINT_SCALE(id, description, n);
/* Generates objects and logs its lifetime depending on the scope. */
INSTRUMENT_DURATION(id, description); // PixLib/C++ only
```

Listing 1. Macros used for instrumenting code.

3.1. Requirements & Approach
We require precise and synchronized timing information in all parts of the system. For the commodity Linux machines running PixLib this is already provided by using the network time protocol (NTP) in combination with an internal counter.
The PPC as well as the MicroBlaze do not provide an internal clock, therefore we implement a 64-bit counter in the FPGA fabrics that is used for time-keeping purposes. To initially synchronize the time of the ROD FPGAs a custom-built NTP client is deployed on the PPC. Also, the master and slave counters are synchronized by sending a PPC command to the slaves and thus absolute timing information is provided for all components.
As most components have multiple instances or employ multi-threading, unique identifiers are needed to distinguish instrumentation data that is generated by the same pieces of code, but by different instances. A combination of network address, process and thread identifiers, and randomized identifiers for the ROD components is used for that purpose.
In order to flexibly instrument relevant code regions, unified C/C++ macros are provided for the ROD master and slave software as well as for PixLib. An example of the instrumentation software interface is provided in Listing 1.
To minimize the impact on the calibration procedure, the implementation is kept to a bare minimum. The instrumentation data is not written out instantly, but rather buffered. It is then send out during less critical phases (when the PPC is not processing a command, for example). For the PixLib instrumentation the output is produced at fixed intervals, thus minimizing the associated overhead.

3.2. Design
A flexible and scalable design is used to provide the instrumentation infrastructure. Figure 3 gives an overview of the architecture.
A central server process (PixInstrumentServer) is used to gather and process the instrumentation data and synchronization information from the different components. Data sources are either plain-text log-files, which are periodically scanned for updates, or UDP datagrams sent to the server by the respective components. The data are then buffered and re-formatted with absolute timing information, before being written to an SQLite database file.

The instrumented source code is examined by a script that extracts the instrumentation points and checks for inconsistencies. This data is then exported to a database file, so that both this file and the runtime instrumentation data can later be loaded by an analysis tool that provides visualization features as depicted in Figure 4.

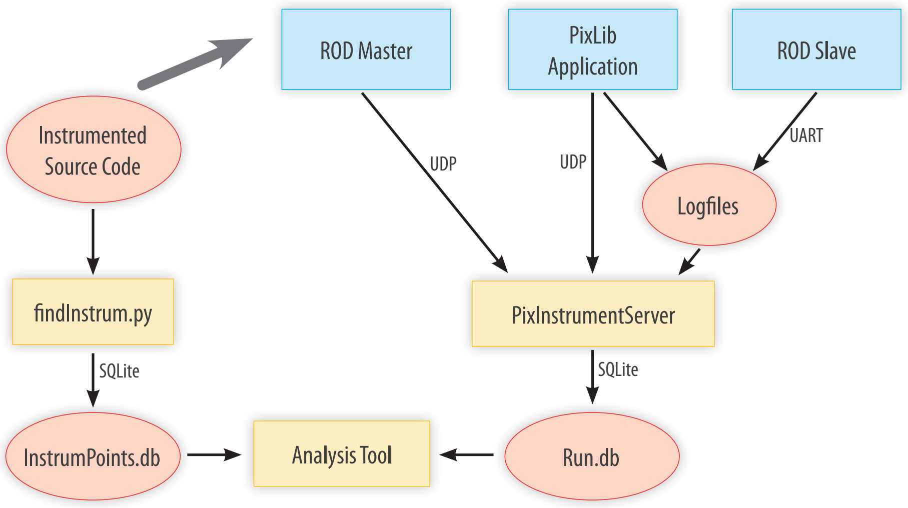


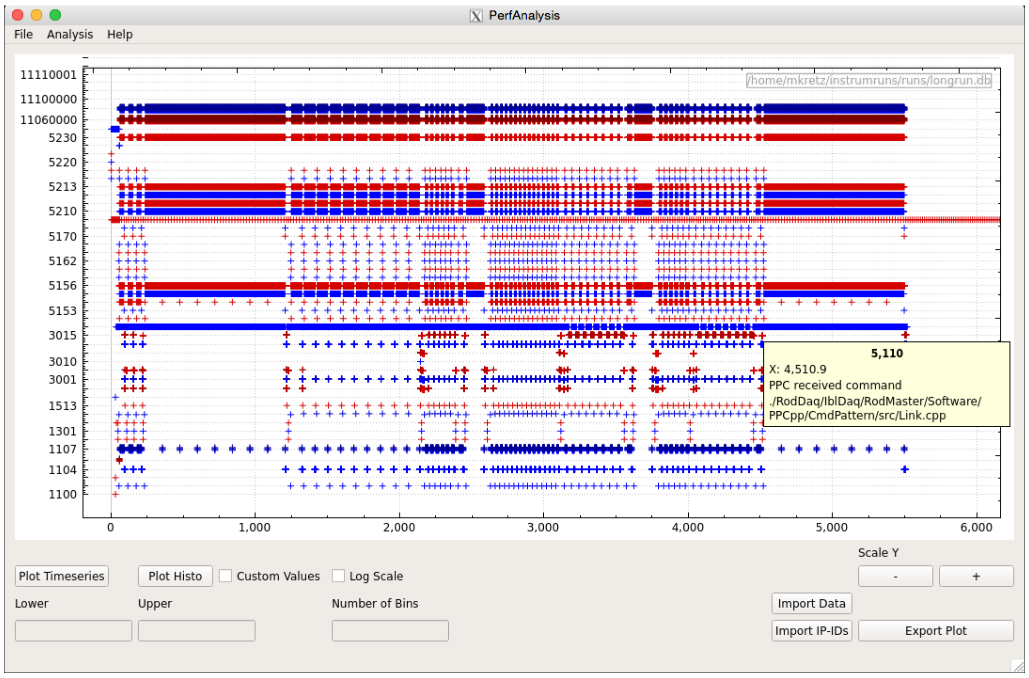
Figure 3. Instrumentation framework design. Various sources provide data to the PixInstrumentServer via different channels. Source code is inspected by a script and instrumentation points are extracted. The analysis tool uses both generated databases to plot the gathered information.

3.3. Example

While calibrating the IBL, the FDAC tuning seemed unexpectedly slow. An FDAC tune optimizes the per pixel amplifier feedback current so that the detector's TOT response becomes more uniform. By analysing the instrumentation data that was generated during an FDAC tuning session it became clear that an unusually long timespan was spent between the major PPC loops that were actually triggering hits on the front-ends. Taking a closer look at the instrumentation data we were able to narrow down the culprit to a routine consisting of a few tens of lines, in which histograms were requested in an inefficient way. Fixing the issue was straightforward from there on.
Optimizing this inefficiency reduced the time spent for an FDAC tuning on seven staves from about 50 minutes down to 15 minutes.

4. Results & Outlook

A scalable instrumentation mechanism with microsecond accuracy has been implemented for the heterogeneous IBL calibration landscape. It supports instrumenting the high-level PixLib components as well as the critical hardware parts on the RODs, that steer a scan and gather the scan data. This enables the developers to accurately investigate the timing of different scans and tunings, debug and optimize them with the help of an analysis interface. It has already provided information that led to a reduction of the time of a basic tuning procedure by about a factor 3.
The approach can also be used for the upcoming L1/L2 upgrade of the Pixel Detector and little to no modification to the implementation should be necessary.



5,110
X: 4,510.9
PPC received command
./RodDaq/IblDaq/RodMaster/Software/
PPCpp/CmdPattern/src/Link.cpp

Figure 4. Analysis tool presenting a time-series of a sequence of scans and tunings spanning a 90 minute period. Instrumentation point IDs are plotted on the y-axis.

References

[1] ATLAS Collaboration. ATLAS Insertable B-Layer Technical Design Report. Technical Report CERN-LHCC-2010-013. ATLAS-TDR-019. CERN, Geneva, September 2010.
[2] ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. JINST 3 S08003. 2008.
[3] J. Dopke et al. The IBL Readout System. ATL-INDET-PROC-2010-031. October 2010.
[4] A. Gabrielli et al. ATLAS IBL: Integration of new HW/SW Readout Features for the Additional Layer of Pixels. ATL-INDET- PROC-2010-031. September 2010.
[5] P. Jenni et. al. ATLAS High-Level Trigger, Data-Acquisition and Controls: Technical Design Report. Technical Report. ATLAS TDR-016. CERN, Geneva, July 2003.
[6] M. Kretz. Studies Concerning the ATLAS IBL Calibration Architecture. CERN-THESIS-2012-67. June 2012.
[7] M. Barbero et al. The FE-I4 Pixel Readout Chip and the IBL-Module. ATL-UPGRADE-PROC-2012-001. Proceedings of Science. 2012.