CERN Summer Student Programme 2025

29 July 2025
pablo.pomaresv@alumno.buap.mx

CMS Roman Pots (PPS) Automatic DAQ Report System and Front-End Electronics Powercycle Procedure

Pablo Jibrán Pomares Valdés

Supervisor:
D. Figueiredo - *Istituto Nazionale di Fisica Nucleare (Pisa)*

Keywords: Compact Muon Solenoid (CMS), Precision Proton Spectrometer (PPS), PPS DAQ.

Abstract

The Compact Muon Solenoid (CMS) Roman Pots are near beam movable detectors installed along the Large Hadron Collider (LHC) beam pipe, located 200 m away from the CMS Interaction Point (IP5). The Precision Proton Spectrometer (PPS) is an experiment designed for measuring the momentum loss of the survived protons in the very forward region. The Roman Pots data acquisition system (DAQ) is fully integrated with CMS.

The primary objective of this project is to create an automated DAQ error reporting system for PPS detector operators. This system enables operators to quickly access actionable information necessary to diagnose and resolve issues. With the developed tools, relevant data from the digitizers' front-end electronics and back-end hardware log files are extracted and presented in a user-friendly web application. The interface opens with a summary page that displays the overall status of the FEDs (front-end data), including fiber locking status for data transmission and frame counters. A detailed digitizer overview follows, highlighting specific warnings and pinpointing any detected issues.

Finally, a power-cycle procedure for the front-end electronics was designed using a finite-state machine (FSM) structure. This action is necessary because some front-end electronic chips are affected by single event upsets (SEUs). The procedure also incorporates hardware protections and ensures that communication between the back-end and front-end remains properly established and stable throughout the process.

Contents

1 Introduction

1.1 The Precision Proton Spectrometer

The Precision Proton Spectrometer (PPS) is a subsystem of the Compact Muon Solenoid (CMS) experiment, designed for measuring survived protons after a bunch crossing collision at interaction point 5 (IP5). Historically, PPS originated as a joint project between CMS and TOTal Elastic and diffractive cross section Measurement (TOTEM)[1] Collaborations.

PPS is composed of a set of mechanical near-beam movable units, which host tracking[2] or timing[3] detectors and are sealed under a secondary vacuum in respect to the beam line. In the literature, this type of detectors are known as Roman-Pots (RPOTs). PPS RPOTs detectors are symmetrically installed in stations along the LHC beam pipe and localized at about 200 m from the CMS Interaction Point (IP5), as shown in Figure 1. PPS operates with different technologies for tracking (CMS 3D pixels), used for proton kinematics measurements, and timing (artificial diamonds sensors) for pileup rejection.

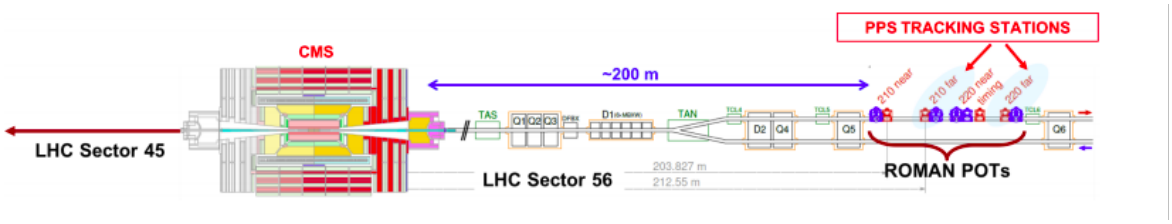

Figure 1: schematic diagram of one of the LHC sectors where PPS is installed. The PPS timing station is placed between two PPS tracking stations. See ref.[3].

The PPS detectors exploit some advantages of being installed at a very forward region due to its acceptance, such as:

- to measure high masses exotic states produced via gamma-gamma interactions, due to the large suppression of quark-gluon (qg) and/or gluon-gluon (gg) cross-sections.
- to measure Anomalous Quartic Gauge Couplings (AQGCs) in Beyond Standard Model (BSM) Physics, increasing the sensitivity of the central detector.
- to measure emerging intact protons from colourless exchanges with small momentum loss ($\xi = \triangle p/p$ between 2 and 10%) and providing data for forward physics processes.
- to monitor the LHC beam optics, stability and to provide feedback for the LHC teams operations.

1.2 PPS Data Acquisition System (DAQ)

The Precision Proton Spectrometer (PPS) readout system is fully integrated into the CMS experiment to enable unified data acquisition. This integration utilizes specialized software tools that automate the control and configuration of the detector's finite state machines (FSMs), implemented within the CMS online C++ framework, XDAQ [4].

The PPS DAQ is divided into separate partitions, each acting as an interface between CMS and PPS. One partition manages the 3D pixel detectors, while another oversees the timing detectors. At the hardware level, PPS sends triggered events directly to the CMS data readout units (FEROL).

The 3D pixel partition follows CMS pixel standards and employs the μTCA electronics framework. Two back-end boards, both FC7 hardware but with different firmware, are used: one handles the slow-control CCU25 token ring protocol for configuring PPS portcard integrated circuits, while the other delivers I^2C fast-commands at a 400 MHz clock rate. These fast-commands configure the readout ASICs (PROC600 and TBM10) on each detector plane and synchronize the LHC clock and CMS trigger signals. Triggered events are synchronized and transmitted through POH4 optotransceiver mezzanines connected to the front-end PPS portcard. These events then reach the CMS FEROL via 10 Gb/s s-link optical links.

The timing partition uses the CCU25 token ring to configure Digitizer Board FPGAs equipped with Microsemi SmartFusion2 M2S150-FC1152 devices [3][5]. These FPGAs encode 32-bit data packets and synchronize data transmission with the CMS DAQ at 400 MHz. The FPGAs distribute I^2C commands to embedded components such as QPLLs and Giga Optical Hybrids (GOHs), while JTAG interfaces communicate with the readout ASICs. The slow-control ring's back-end interface is managed by a VME-standard Front-end Controller (FEC) board. Thanks to its modular design supporting various readout ASICs, the Digitizer Board can adapt to different PPS data-taking configurations.

Triggered events from the timing partition are received by the PPS Scalable Readout System (SRS). Data from the Digitizer Board's GOHs are sent to CMS FEROL via 5 Gb/s s-link copper connections. The SRS control software readies the timing partition for data acquisition by transmitting UDP packets.

2 Information Extraction and Classification

This project was primarily developed in Python, with some sub processes implemented in Bash script. All the code can be retrieved on Gitlab [6].

The digitizers and the scalable readout system (SRS) error logs are obtained via the xDAQ system web page. However, the files come with unnecessary information and conflicting syntax so a cleanse of them must be performed before storing its information. In the case of the digitizers, this is done with the script `extractErrMesg.py` and for the SRS it is done from the main file itself.

In order to process the error log files two object classes were created. One called `Digiboard_ErrLog` and the other being called `SRS_ErrLog` which—as the name would infer—correspond to the digitizer and the SRS report files respectively. In the following sections their structure and attributes will be briefly discuss.

2.1 Object `Digiboard_ErrLog`

This object stores all information returned by the digitizer. That is, it stores the finite-state machine (FSM), trigger commands (Bg0, L1a, event reset), BX counters and the state bytes of a given board. By default, all values are 0; however, one can call the class method

`from_file(file)` in order to populate it with the real board values. If needed, a string representation of the class was added that returns all the data contained in the object.

For the I^2C channels of the digitizer's FPGAs status, a subclass was created in order to keep the code organized. Due to each board containing four I^2C channels, a dictionary (hash-map) was created, relating the I^2C class and the channels address ("0x15", "0x16", etc). Additionally, a decoder for the FSM was added in order to explicitly be able to read out its information.

2.2 Object `SRS_ErrLog`

The `SRS_ErrLog` reads the system of a particlar, specified, server. It stores values such as the event and frame counters, the fiber status, and the counter checks. The SRS status files are not as cluttered as the ones from the digitizers, so it was possible to populate the object directly from the source file. In order to do this, the class method `from_file(file)` should be called. Just as before, a string representation of the class was added that returns all the data contained in the object.

3 Web Page Layout

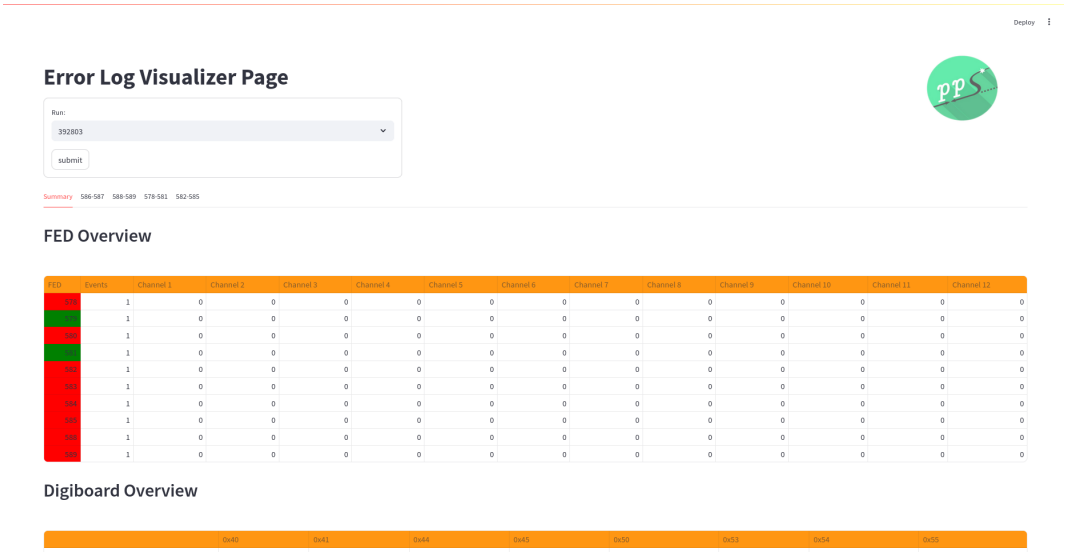

Figure 2: Home page of the web application

When entering the web page the title is located in the left most corner, mirrored on the other side by the PPS logo. As it can be seen in figure 2, below the title a run selection is placed. By default, the most recent event is first shown, however, it is possible to select a specific one with the run selection form. All folders located in the runs directory are automatically listed from the earliest to the latest. Once a run is selected and the submit button is pressed, its the information will be displayed on the web page. A string representation of the class was added that returns all the data contained in the object.

Afterwards, a tab selection can be observed where with the partitions: 'Summary', '586-587', '588-589', '578-581', and '582-585'. The numbers on the four latter tabs allude to FEDs (frontend data) ID and are grouped in servers, i. e., the first tab refers to the FEDs belonging to server 25, the next one to those who belong in server 26 and so on.

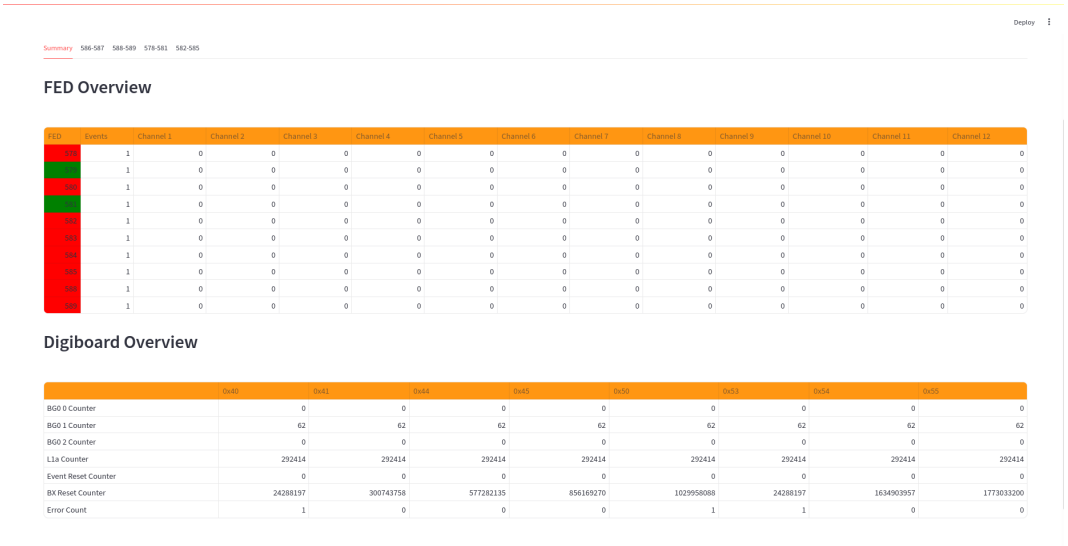

Figure 3: Summary page. On top, the FED Overview is seen, and below, the Digitizer boards Overview is displayed

As it can be observed above, on the 'Summary' tab, a synopsis of all FEDs is placed, displaying the event counter of each fiber. The colors on the FED IDs correspond to the fiber status; if any of them fails then the FED will be highlighted in red, otherwise, it will be highlighted green. Below the FED Overview table, the Digiboard Overview is presented. In there the BG0, L1a, Event reset, RX reset and error counts of all CCUs are shown.

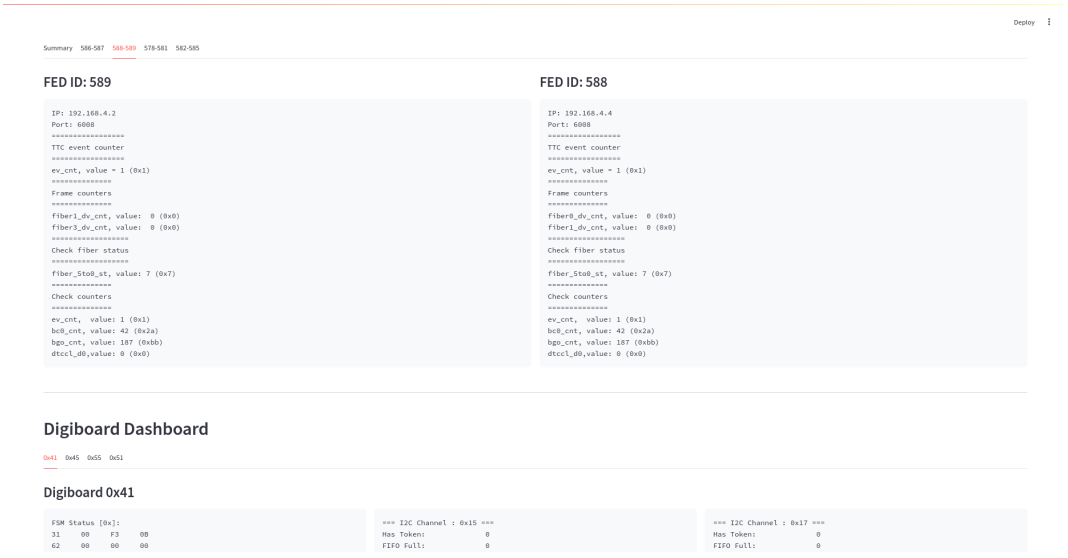

Figure 4: View of the FED ID information

In the FED IDs tabs, figure 4, the SRS status of each FED is presented. The trigger commands (TTC) event counter, the corresponding frame counters, fiber checks, and counter checks are also laid out for each FED in the server—as well as their IP address and port. Each server controls a set of FED hardware. Underneath, tabs of their corresponding CCUs can be found. In each one of those, all the information of the CCU is shown, including all four I2C Channels. Finally, a FSM decoder, disclosing the T1 decoder, the Giga Optical Hybrid (GOH) sender, the HPTDC downloaders and frame builders. As it is shown in figure 5. All of them are FPGA firmware related registers/counters to the digitizers.

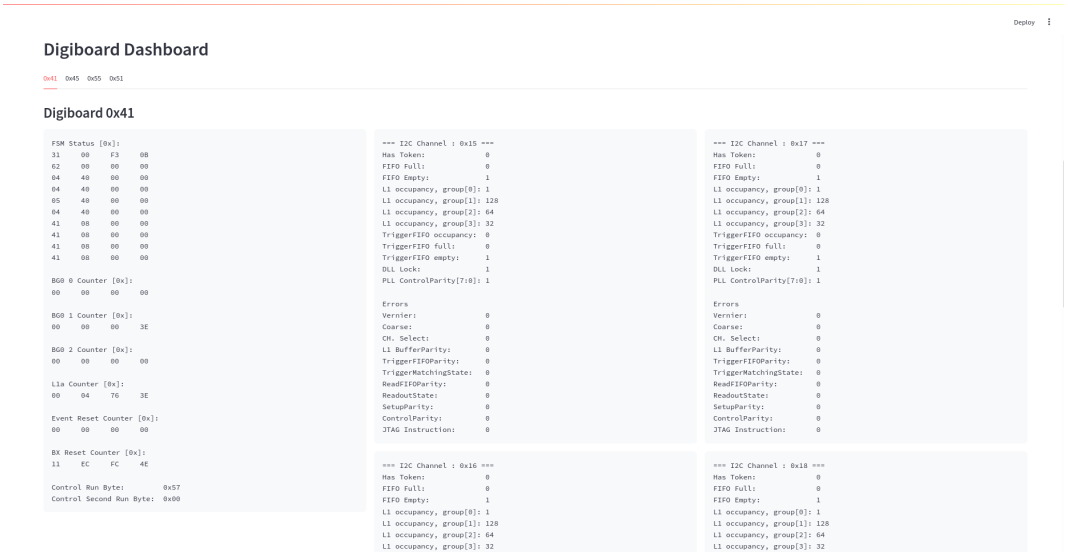

Figure 5: Digitizers information

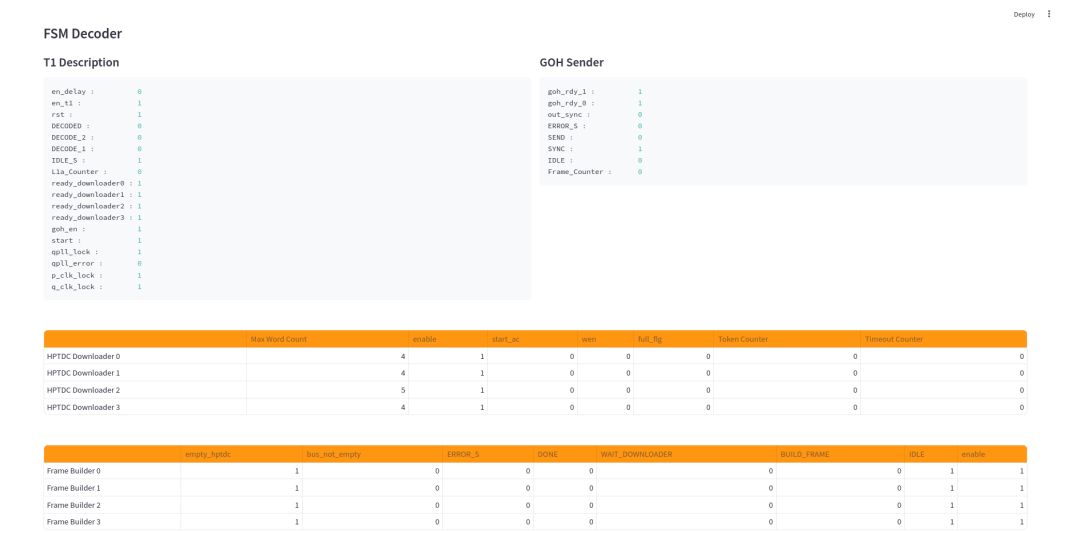

Figure 6: FSM Decoder information

4 Front End Electronics Powercycle

Due to the nature of the electronics, this part of the project was programmed in C++. A finite state machine structure was chosen for the routine because of the front-end electronic chips' exposure to single-event upsets; additionally, this structure provides better modularization and expansion capability. The software was tested on an unused FEC with a single ring and CCU attached. Even with this limitation, all processes executed there can be extrapolated to the rest of the components, as they share the same structure. As a result, two executables were generated: one for a single thread and one for all. The latter has yet to be fully tested.

The powercycle process begins by initializing a specific front-end controller (FEC), ring, and CCU. It then proceeds to reset the virtual machine environment controller (PLX), FEC, CCU, phase-locked loop (PLL), and GOH, followed by configuring the GOH's power, reading the FSMs, and finally scanning the ring. In case of failure of any of these steps, except the FSM reader, three attempts are made. If the failure persists, however, the program is terminated. After an FSM reader failure, the complete process has to be remade, this is, the FEC, ring and CCU has to be reassigned, the PLX reseted, and so on. This process is then applied to every operating FEC, for a total of twelve times.

4.1 Program Structure

As it was mentioned earlier, the program behaves as an FSM. The states being:

1. INIT,
2. PLX_RESET,
3. FEC_RESET,
4. CCU_RESET,
5. PLL_RESET,
6. GOH_RESET,
7. GOH_POWERSET,
8. FSM_READ,
9. SCAN_RING_DEVICE

INIT being the starting state (no action). To each state, a funtion is associated to them, which executes the specific FEC command. They follow the general structure:

```
void function(FecFunctions* FecFunc){
    try {
        FecFunc->fec_function();
    }
    catch (FecExceptionHandler &e) {
```

```
            throw e;
        }
}
```

FecFunctions is a class that is found in the custom library "FecFunctions.h". This object takes as arguments the VME, the chips name ("hptdc"), FEC, slot and CCU addresses. **FecExceptionHandler** is, as the name suggest, a handler class that reports on server side issues.

The main component of the program is the class **Powercycle**, which has a function to add steps to the process, stores—in an orderly manner—the added state and function, track the number of attempts, and, runs the procedure. The run command handles the repetition procedure mentioned previously. Between each procedure a 50 μs delay was added to account for the distance between the servers and the electronics. At the beginning and end of each state a message is returned to the user to aid in error debugging. Finally, the class also keep track of the number of times each state has been retried and the number of times the whole process has been attempted.

5 Conclusion

The objectives of generating an automatic DAQ report system and its display on a web application were successfully met. Tests with status reports (containing errors) were performed, all returning satisfactory results. Due to the implementation of the error logs as structured objects, the system is easy to maintain, upgrade and expand—even from a remote location. Additionally, the powercycle for the front end electronics works as expected. However, final testing has not yet been able to be conducted due to the limitations of the test ring.

6 Acknowledgements

I would like to thank my project supervisor, Diego Figueiredo, for the incredible support he has provided during my stay, for the patience he has shown me, and for all he has taught me over these past few weeks. I would also like to thank my thesis supervisor, Isabel Pedraza, for the help and encouragement she has given me over the past year. Without her assistance, none of this would have been possible. Finally, I would like to thank the CERN Student Programme for offering me the amazing opportunity to take part in it.

References

[1] M. Albrow, M. Arneodo, V. Avati *et al.*, 'CMS-TOTEM Precision Proton Spectrometer,' Tech. Rep., 2014. [Online]. Available: https://cds.cern.ch/record/1753795.

[2] M. M. Obertino, 'The PPS tracking system: performance in LHC Run2 and prospects for LHC Run3,' *JINST*, vol. 15, C05049. 11 p, 2020. DOI: 10.1088/1748-0221/15/05/C05049. [Online]. Available: https://cds.cern.ch/record/2747949.

[3] E. Bossini, 'The CMS Precision Proton Spectrometer timing system: performance in Run 2, future upgrades and sensor radiation hardness studies,' *JINST*, vol. 15, C05054. 12 p, May 2020, Proceedings of IPRD2019 Conference. DOI: 10.1088/1748-0221/15/05/C05054. arXiv: 2004.11068. [Online]. Available: https://cds.cern.ch/record/2715995.

[4] V. Brigljevic *et al.*, 'Using XDAQ in application scenarios of the CMS experiment,' CERN, Tech. Rep., 2003, eConf C0303241, MOGT008. eprint: hep-ex/0305076.

[5] TOTEM Collaboration, 'Upgrade of the TOTEM T2 Telescope, Technical Design Report,' CERN, Geneva, Tech. Rep., Jun. 2019. [Online]. Available: https://cds.cern.ch/record/2677468.

[6] P. Pomares, *PPS Error Report System*, https://gitlab.cern.ch/ppomares/error_dashboard_pps, 2025.