Universiteit Gent
Faculteit Wetenschappen
Vakgroep Fysica en Sterrenkunde

No title yet

No sub-title neither, obviously...

Alexis Fagot

Thesis to obtain the degree of
Doctor of Philosophy in Physics
Academic years 2012-2017

Universiteit Gent
Faculteit Wetenschappen
Vakgroep Fysica en Sterrenkunde

Promotoren: Dr. Michael Tytgat
Prof. Dr. Dirk Ryckbosch

Universiteit Gent
Faculteit Wetenschappen

Vakgroep Fysica en Sterrenkunde
Proeftuinstraat 86, B-9000 Gent, België

Tel.: +32 9 264.65.28
Fax.: +32 9 264.66.97

Thesis to obtain the degree of
Doctor of Philosophy in Physics
Academic years 2012-2017

Acknowledgements

Ici on remerciera tous les gens que j'ai pu croiser durant cette aventure et qui m'ont permis de passer un bon moment

Gent, ici la super date de la mort qui tue de la fin d'écriture
Alexis Fagot

Table of Contents

List of Figures

List of Tables

List of Acronyms

List of Acronyms

A

AFL	Almost Full Level

B

BARC	Bhabha Atomic Research Centre
BLT	Block Transfer
BR	Branching Ratio

C

CAEN	Costruzioni Apparecchiature Elettroniche Nucleari S.p.A.
CERN	European Organization for Nuclear Research
CFD	Constant Fraction Discriminator
CMS	Compact Muon Solenoid
CSC	Cathode Strip Chamber

D

DAQ	Data Acquisition
DCS	Detector Control Software
DQM	Data Quality Monitoring
DT	Drift Tube

F

FEE	Front-End Electronics
FEB	Front-End Board

G

GE-/-	Find a good description
GE1/1	Find a good description
GE2/1	Find a good description
GEANT	GEometry ANd Tracking - a series of software toolkit platforms developed by CERN
GEM	Gas Electron Multiplier
GIF	Gamma Irradiation Facility
GIF++	new Gamma Irradiation Facility

H

HL-LHC	High Luminosity LHC
HPL	High-pressure laminate
HV	High Voltage

I

iRPC	improved RPC
IRQ	Interrupt Request

L

LHC	Large Hadron Collider
LS1	First Long Shutdown
LS3	Third Long Shutdown
LV	Low Voltage
LVDS	Low-Voltage Differential Signaling

M

MC	Monte Carlo
MCNP	Monte Carlo N-Particle
ME-/-	Find good description

ME0	Find good description
MRPC	Multigap RPC

N

NIM	Nuclear Instrumentation Module logic signals

P

PMT	PhotoMultiplier Tube

R

RE-/-	Find a good description
RE2/2	Find a good description
RE3/1	Find a good description
RE3/2	Find a good description
RE4/1	Find a good description
RE4/2	Find a good description
RE4/3	Find a good description
RMS	Root Mean Square
ROOT	a framework for data processing born at CERN
RPC	Resistive Plate Chamber

S

SPS	Super Proton Synchrotron

T

TDC	Time-to-Digital Converter
ToF	Time-of-flight

W

webDCS	Web Detector Control System

Nederlandse samenvatting –Summary in Dutch–

Le resume en Neerlandais (j’aurais peut-etre de apprendre la langue juste pour ca...).

English summary

Le meme résume mais en Anglais (on commencera par la hein!).

1 Introduction

1.1 A story of High Energy Physics

1.2 Organisation of this study

2

Investigating the TeV scale

2.1 The Standard Model of Particle Physics

2.2 The Large Hadron Collider and the Compact Muon Solenoid

2.3 Muon Phase-II Upgrade

After the more than two years lasting First Long Shutdown (LS1), the Large Hadron Collider (LHC) delivered its very first Run-II proton-proton collisions early 2015. LS1 gave the opportunity to the LHC and to the its experiments to undergo upgrades. The accelerator is now providing collisions at center-of-mass energy of $13\,\mathrm{TeV}$ and bunch crossing rate of $40\,\mathrm{MHz}$, with a peak luminosity exceeding its design value. During the first and upcoming second LHC Long Shutdown, the Compact Muon Solenoid (CMS) detector is also undergoing a number of upgrades to maintain a high system performance [1].

From the LHC Phase-2 or High Luminosity LHC (HL-LHC) period onwards, i.e. past the Third Long Shutdown (LS3), the performance degradation due to integrated radiation as well as the average number of inelastic collisions per bunch crossing, or pileup, will rise substantially and become a major challenge for the LHC experiments, like CMS that are forced to address an upgrade program for Phase-II [2]. Simulations of the expected distribution of absorbed dose in the CMS detector under HL-LHC conditions, show in figure 5.14 that detectors placed close to the beamline will have to withstand high irradiation, the radiation dose being of the order of a few tens of Gy.

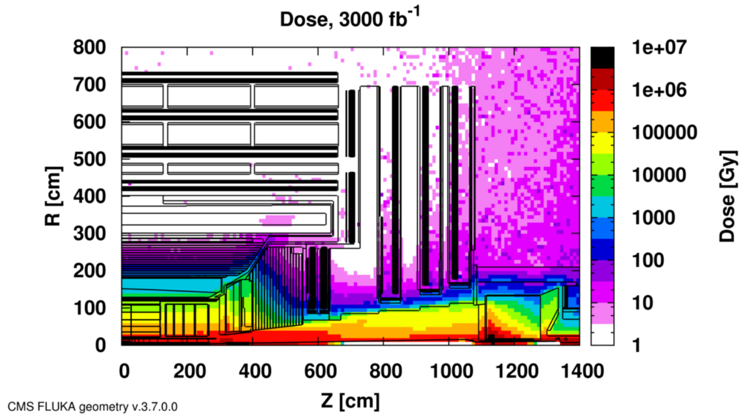


Figure 2.1: Absorbed dose in the CMS cavern after an integrated luminosity of 3000 fb*. R is the transverse distance from the beamline and Z is the distance along the beamline from the Interaction Point at Z=0.*

The measurement of small production cross-section and/or decay branching ratio processes, such as the Higgs boson coupling to charge leptons or the $B_s \longrightarrow \mu^+\mu^-$ decay, is of major interest and specific upgrades in the forward regions of the detector will be required to maximize the physics acceptance on the largest possible solid angle. To ensure proper trigger performance within the present coverage, the muon system will be completed with new chambers. In figure 2.2 one can see that the existing Cathode Strip Chambers (CSCs) will be completed by Gas Electron Multipliers (GEMs) and Resistive Plate Chambers (RPCs) in the pseudo-rapidity region $1.6 < |\eta| < 2.4$ to complete its redundancy as originally scheduled in the CMS Technical Proposal [3].

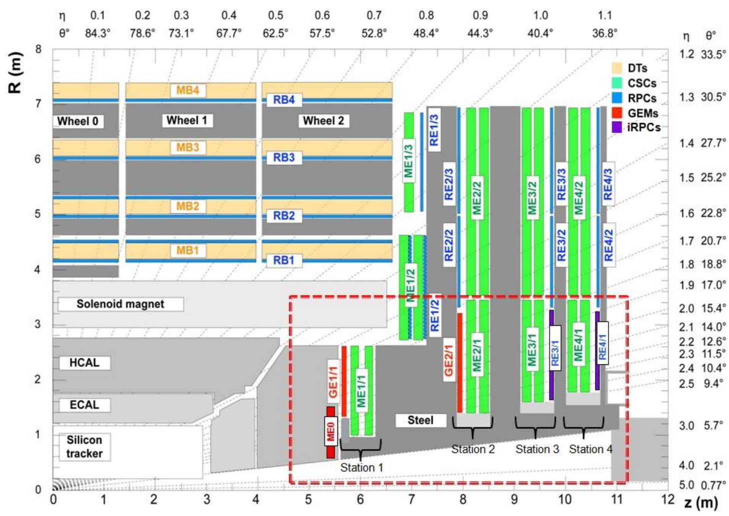


Figure 2.2: A quadrant of the muon system, showing DTs (yellow), RPCs (light blue), and CSCs (green). The locations of new forward muon detectors for Phase-II are contained within the dashed box and indicated in red for GEM stations (ME0, GE1/1, and GE2/1) and dark blue for improved RPC (iRPC) stations (RE3/1 and RE4/1).

RPCs are used by the CMS first level trigger for their good timing performances. Indeed, a very good bunch crossing identification can be obtained with the present CMS RPC system, given their fast response of the order of $1\,\mathrm{ns}$. In order to contribute to the precision of muon momentum measurements, muon chambers should have a spatial resolution less or comparable to the contribution of multiple scattering [1]. Most of the plausible physics is covered only considering muons with $p_T < 100\,\mathrm{GeV}$ thus, in order to match CMS requirements, a spatial resolution of $\mathcal{O}$(few mm) the proposed new RPC stations, as shown by the simulation in figure 2.3. According to preliminary designs, RE3/1 and RE4/1 readout pitch will be comprised between 3 and $6\,\mathrm{mm}$ and 5 η-partitions could be considered.

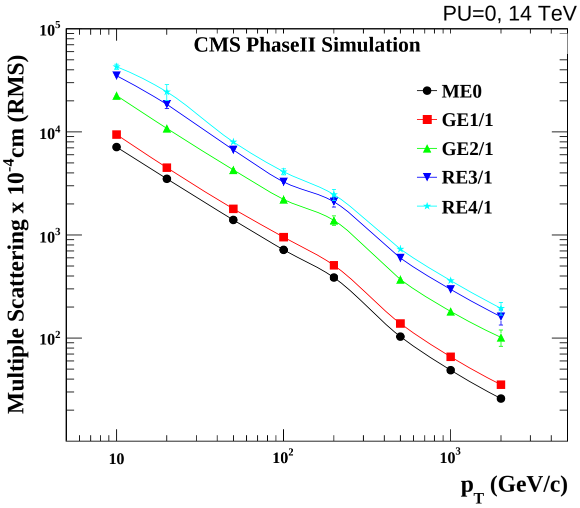


Figure 2.3: RMS of the multiple scattering displacement as a function of muon p_T for the proposed forward muon stations. All of the electromagnetic processes such as bremsstrahlung and magnetic field effect are included in the simulation.

3

Amplification processes in gaseous detectors

3.1 Signal formation

3.2 Gas transport parameters

4

Resistive Plate Chambers

A Resistive Plate Chamber (RPC) is a gaseous detector using the same physical processes described in Chapter 3. It has been developed in 1981 by Santonico and Cardarelli [4], under the name of *Resistive Plate Counter*, as an alternative to the local-discharge spark counters proposed in 1978 by Pestov and Fedotovich [5, 6]. Working with spark chambers implied using high-pressure gas and high mechanical precision which the RPC simplified by formerly using a gas mixture of argon and butane flowed at atmospheric pressure and a constant and uniform electric field propagated in between two parallel electrode plates. Moreover, a significant increase in rate capability was introduced by the use of electrode plate material with high bulk resistivity, preventing the discharge from growing throughout the whole gas gap. Indeed, the effect of using resistive electrodes is that the constant electric field is locally canceled out by the development of the discharge, limiting its growth.

Through its development history, different operating modes [7–9] and new detector designs [10–12] have been discovered, leading to further improvement of the rate capability of such a detector. Moreover, the addition of SF_6 into the gas mix improved the stability of operation of the RPC [13, 14].

The low developing costs and easily achievable large detection areas offered by RPCs, as well as the wide range of possible designs, made them a natural choice to as muon chambers and/or trigger detectors in multipurpose experiments such as CMS [1] or ATLAS [15], time-of-flight detectors in ALICE [16], calorimeter with CALICE [17] or even detectors for volcanic muography with ToMuVol [18].

4.1 Principle

RPCs are ionisation detectors composed of two parallel resistive plate electrodes in between which a constant electric field is set. The space in between the electrodes, referred as *gap*, is filled with a dense gas that is used to generate primary ionization into the gas volume. The free charge carriers (electrons and cations) created by the ionization of the gas molecules are then accelerated towards the electrodes by the electric field, as shown in Figure 4.1 [19].

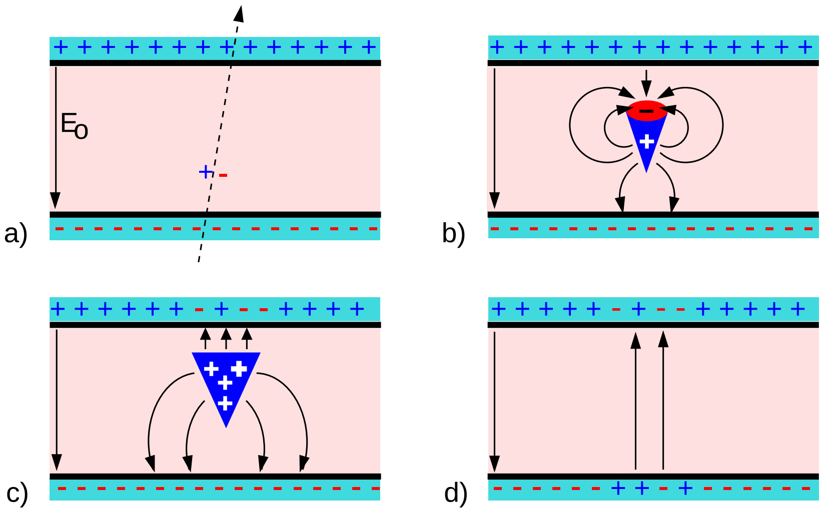


Figure 4.1: Different phases of the avalanche development in the RPC gas volume subjected to a constant electric field E_0. a) An avalanche is initiated by the primary ionisation caused by the passage of a charged particle through the gas volume. b) Due to its growing size, the avalanche starts to locally influence the electric field. c) The electrons, lighter than the cations reach the anode first. d) The ions reach the cathode. While the charges have not recombined, the electric field in the small region around the avalanche stays affected and locally blind the detector.

RPCs being passive detectors, a current on pick-up copper read-out placed outside of the gas volume is induced by the charge accumulation during the growth of the avalanche. As a result, the time resolution of the detector is substantially increased as the output signal is generated while the electrons are still in movement. The advantage of a constant electric field, over multi-wire proportional chambers, is that the electrons are being fully accelerated from the moment charge carriers are freed and feel the full strength of the electric field that doesn't depend on the distance to the readout and that the output signal doesn't need for the electrons to be physically collected.

The typical gas mixture RPCs are operated with is generally composed of 3 gas compounds.

- Tetrafluoroethane ($C_2F_4H_2$), also referred to as *Freon*, is the principal compound of the RPC gas mixtures, with a typical fraction above 90%. It is used for it's high effective Townsend coefficient and the great average fast charge that allows to operate the detector with a high threshold with respect to argon, for example, that has similar effective Townsend coefficient but suffers from a lower fast charge. To operate with similar conditions, argon would require a higher electric field leading to a higher fraction of streamers, thus limiting the rate capability of the detector [20].

- Isobutane (i-C_4H_{10}), only present in a few percent in the gas mixtures, is used for its UV quenching properties [21] helping to prevent streamers due to UV photon emission during the avalanche growth.

- Sulfur hexafluoride, (SF_6), referred to simply as *SF6*, is used in very little quantities for its high electronegativity. Excess of electrons are being absorbed by the compound and streamers are suppressed [14]. Nevertheless, a fraction of SF_6 higher than 1% will not bring any extra benefice in terms of streamer cancelation power but will lead to higher operating voltage.

After an avalanche developed in the gas, a time long compared to the development of a discharge is needed to recombine the charge carriers in the electrode material due to their resistivity. This property has the advantage of affecting the local electric field and avoiding sparks in the detector but, on the other hand, the rate capability is intrinsically limited by the time constant τ_{RPC} of the detector. Using a quasi-static approximation of Maxwell's equations for weakly conducting media, it can be shown that the time constant τ_{RPC} necessary to the charge recombination at the interface in between the electrode and the gas volume is given by the Formula 4.1 [22].

$$\tau_{RPC} = \frac{\epsilon_{electrode} + \epsilon_{gas}}{\sigma_{electrode} + \sigma_{gas}} \tag{4.1}$$

A gas can be assimilated to vacuum, leading to $\epsilon_{gas} = \epsilon_0$ and $\sigma_{gas} = 0$, and the electrodes permittivity and conductivity can be written as $\epsilon_{electrode} = \epsilon_r \epsilon_0$ and $\sigma_{electrode} = 1/\rho_{electrode}$, showing the strong dependence of the time constant to the electrodes resistivity in Formula 4.2.

$$\tau_{RPC} = (\epsilon_r + 1)\epsilon_0 \times \rho_{electrode} \tag{4.2}$$

Very few materials with a low enough resistivity exist in nature. The resistivity targeted to build RPCs ranges from 10^9 to 10^{12} $\Omega \cdot \mathrm{cm}$. The most common RPC electrode materials are displayed in Table 4.1. When the doped glass and ceramics can offer short time constants of the order of $1\,\mathrm{ms}$, the developing cost of such materials is quite high due to the very low demand. Thus, High-pressure laminate (HPL) is often the choice for high rate experiments using very large RPC detection areas. Other experiments working at cosmic muon fluxes can safely operate with ordinary float glass.

Material	$\rho_{electrode}$ ($\Omega \cdot \mathrm{cm}$)	ϵ_r	τ_{RPC} (ms)
Float glass	10^{12}	$\sim$7	$\sim$700
High-pressure laminate	10^{10} to 10^{12}	$\sim$6	$\sim$6 to 600
Doped glass (LR S)	10^9 to 10^{11}	$\sim$10	$\sim$1 to 100
Doped ceramics (SiN/SiC)	10^9	$\sim$8.5	$\sim$1
Doped ceramics (Ferrite)	10^8 to 10^{12}	$\sim$20	$\sim$0.2 to 2000

Table 4.1: Properties of the most used electrode materials for RPCs.

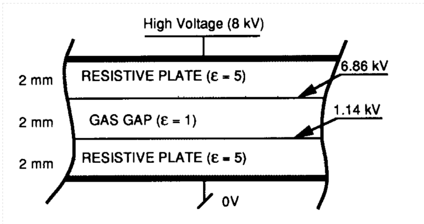
4.1.1 Electron drift velocity

4.2 Rate capability and time resolution of Resistive Plate Chambers

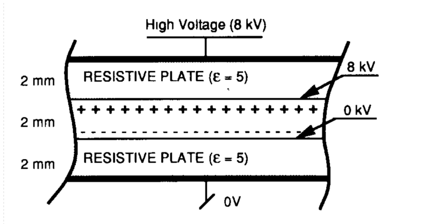
As already previously discussed, the electrode material plays a key role in the max intrinsic rate capability of RPCs. R&D is being done to develop at always cheaper costs material with lower resistivity. Nevertheless, the amount of charge released, i.e. the size of the discharge, if reduced leads to a smaller blind area in the detector, increasing the rate capability of the detector.

4.2.1 Operation modes

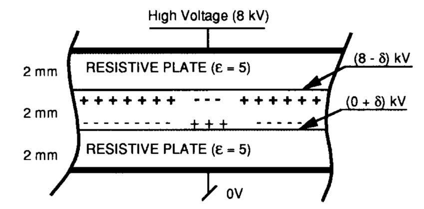
RPCs where developed early 1980s. At that time it was using an operating mode now referred to as *streamer mode*. Streamers are large discharges that develop in between the 2 electrodes enough to locally discharge the electrodes. If the electric field inside of the gas volume is strong enough, with electrons being fast compared to ions, a large and dense cloud of positive ions will develop nearby the anode and extend toward the cathode while the electrons are being collected, eventually leading to a streamer discharge due to the increase of field seen at the cathode. the field is then strong enough so that electrons are pulled out of the cathode. Electrodes, though they are a unique volume of resistive material, can be assimilated to capacitors. At the moment an electric field is applied in between their outer surfaces, the charge carriers inside of the volume will start moving leading to a situation where there is no voltage across the electrodes and a higher density of negative charges, i.e. electrons, on the inner surface of the cathode. Finally, when a streamer discharge occurs, these electrons are partially released in the gas volume contributing to increase the discharge strength until the formation of a conductive plasma, the streamer. This can be understood through Figure 4.2 [7]. Streamer signals are very convenient in terms of read-out as no amplification is required with output pulses amplitudes of the order of a few tens to few hundreds of mV as can be seen on Figure 4.3.



(a)



(b)

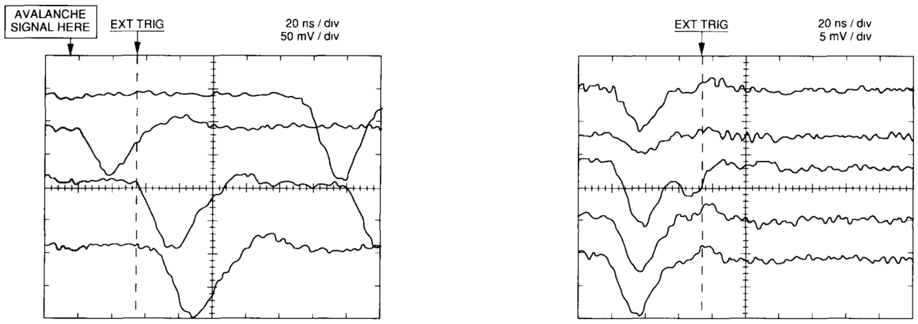


(c)

Figure 4.2: Movement of the charge carriers in an RPC. Figure 4.2a: Voltage across an RPC whose electrode have a relative permittivity of 5 at the moment the tension s applied. Figure 4.2b: After the charge carriers moved, the electrodes are charged and there is no voltage drop over the electrodes anymore. The full potential is applied on the gas gap only. Figure 4.2c: The streamer discharge initiated by a charged particle transports electrons and cations towards the anode and cathode respectively.

When the electric field is reduced though, the electronic gain is small until the electrons get close enough to the anode and the positive ion cloud is much smaller. The electric field cannot rise to the point a field emission of electrons on the cathode is possible. The resulting signal is weak, of the order of a few mv as shown on Figure 4.3, and requires amplification. This is the *avalanche mode* of RPC operation.

This mode offers a higher rate capability by providing smaller discharges that don't affect the electrodes charge and are more locally contained in the gas volume as was demonstrated by Crotty



(a) *(b)*

Figure 4.3: Typical oscilloscope pulses in streamer mode (Figure 4.3a) and avalanche mode(Figure 4.3b). In the case of streamer mode, the very small avalanche signal is visible.

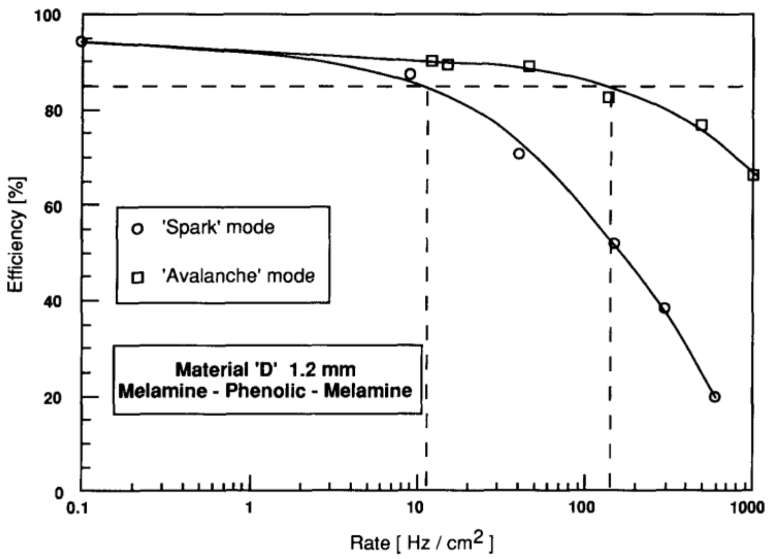


Figure 4.4: Rate capability comparison for the streamer and avalanche mode of operation. An order of magnitude in rate capability for a maximal efficiency drop of 10% is gained by using the avalanche mode over the streamer mode.

with Figure 4.4 [7]. The detector only stays locally blind the time the charge carriers are recombined and there is no need for electrode recharge which is a long process affecting a large portion of the detector. Another advantage of avalanche signals over streamer is the great time consistency. Figure 4.3 shows very clearly that avalanche signals have a very small time jitter. This is a natural choice for high rate experiments.

4.2.2 Detector design, active volume and time jitter

Different RPC design have been used and each of them present its own advantages. Historically, the first type of RPC to have been developed is what is referred now to *noarrow gap* RPC [4, 23]. After the avalanche mode has been discovered [7], it has been proven that increasing the width of the gas gap lead to higher rate capability, due to lower charge deposition per avalanche, and lower power dissipation [23]. Nevertheless, by increasing the gas gap width, the time resolution of the detector decreases. This is a natural result if the increase of active gas volume in the detector is taken into account. Indeed, for a given threshold, only the small fraction of gas closest to the cathode will provide enough gain to have a detectable signal. In the case of a wider gas volume, the active region is then larger and a larger time jitter is introduced with the variation of starting position of the avalanche, as discussed in [10]. To solve improve both the time resolution and the rate capability, different methods were used trying to get advantages of both narrow and wide gap RPCs.

Double gap RPCs are made out of 2 narrow RPC detectors. The 2 RPC gap are placed on top of each other, their anodes facing each other and placing the readout panel with the copper strips in between the 2 RPCs. This detector layout is used as an OR system in which each individual chamber participates in the ouput signal. The gain of such a detector is greatly reduced with respect to single gap RPCs with an efficiency plateau reached at lower voltage as visible on Figure 4.5.

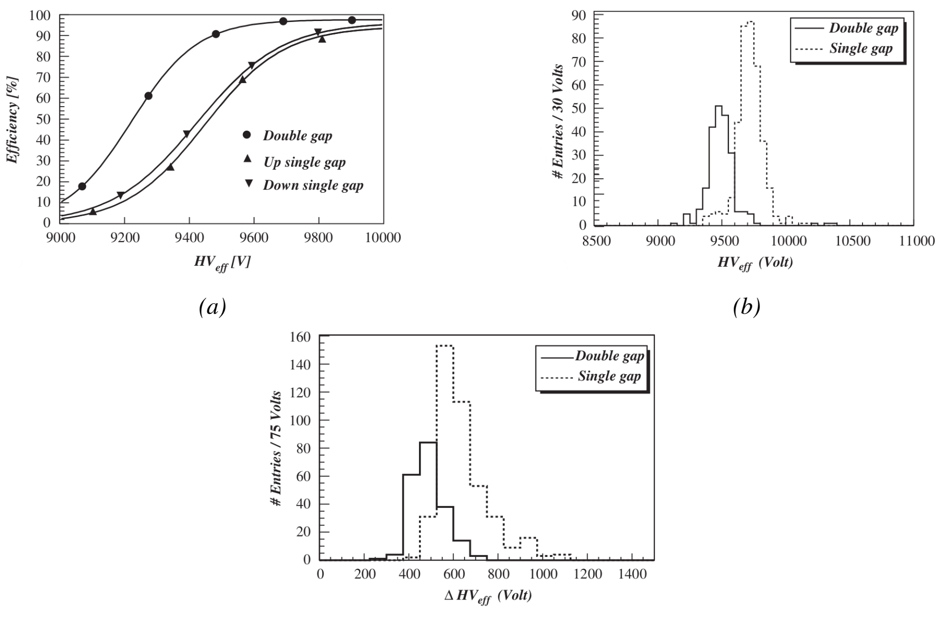


Figure 4.5: Comparison of performance of CMS double and single gap RPCs using cosmic muons [24]. Figure 4.5a: Comparison of efficiency sigmoids. Figure 4.5b: Voltage distribution at 95% of maximum efficiency. Figure 4.5c: $\Delta^{90\%}_{10\%}$ distribution.

Multigap RPC (MRPC) are layouts in which floating sub electrode plates are placed into a wide gap RPC to divide the gas volume and create a sum of narrow gaps [10, 11]. The time resolution of such a detector can reach of few tens of ps, with gas gaps of the order of a a few hundred μm as shown in Figure 4.6 representing ALICE Time-of-flight (ToF) MRPCs.

Sometimes used as a double multigap RPC, taking advantage of the OR of double gap RPCs, the MRPC is mainly used as ToF detector [25–29] due to it's excellent timing properties that allow

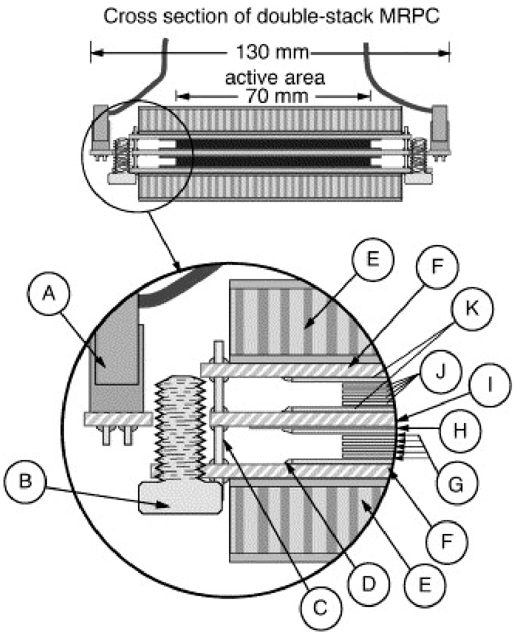


Figure 4.6: Presentation of ALICE MRPC using $250\,\mu\mathrm{m}$ *gas gaps,* $620\,\mu\mathrm{m}$ *outer glass electrodes and* $550\,\mu\mathrm{m}$ *inner floating electrodes. More details on the labels are given in [25].*

to perform particle identification as explained by Williams in [30]. The principle of particle identification using ToF is simply the measurement of the velocity of a particle. Indeed, particles are defined by their mass (for the parameter of interest here, their electric charge being measured using the bending angle of the particles traveling through a magnetic field) and this mass can be calculated by measuring the velocity β and momentum of the particle:

$$\beta = \frac{p}{\sqrt{p^2 + m^2}} \tag{4.3}$$

Intuitively, it is trivial to understand that 2 different particles having the same momentum will have a different velocity due to the mass difference and thus a different flight time through the detector and this is used to separate and identify particles. The better the time resolution of the ToF system used, the stronger will the separation be:

$$T = \frac{L}{v} = \frac{L}{c \cdot \beta}, \quad \Delta T = T_1 - T_2 = \frac{L}{c}\left(\sqrt{1 + m_1^2/p^2} - \sqrt{1 + m_2^2/p^2}\right) \cong (m_1^2 - m_2^2)\frac{L}{2cp^2} \tag{4.4}$$

An example of particle identification is given for the case of STAR experiment in Figure 4.7.

Another benefice of using such small gas gaps is the strong reduction of the average avalanche volume and thus of the blind spot on MRPCs leading to an improved rate capability. Multigaps can sustain backgrounds of several $\mathrm{kHz/cm^2}$ as demonstrated in Figure 4.8.

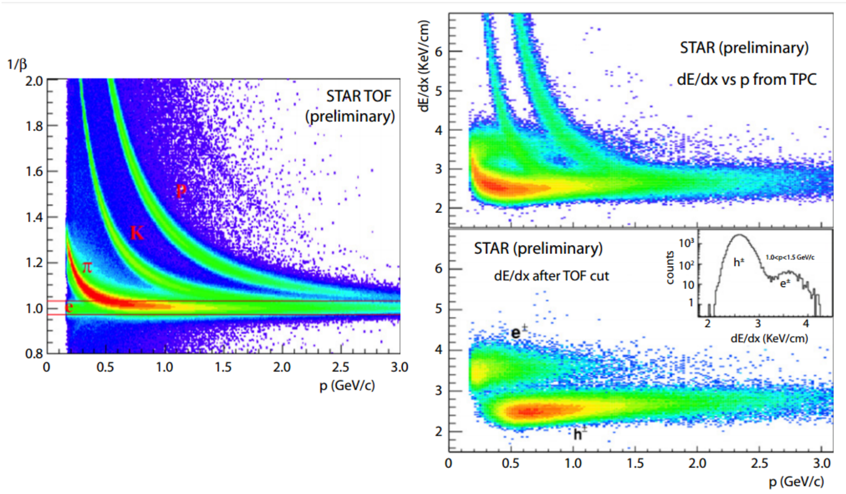


Figure 4.7: Particle identification applied to electrons in the STAR experiment. The identification is performed combining ToF and dE/dx measurements [30].

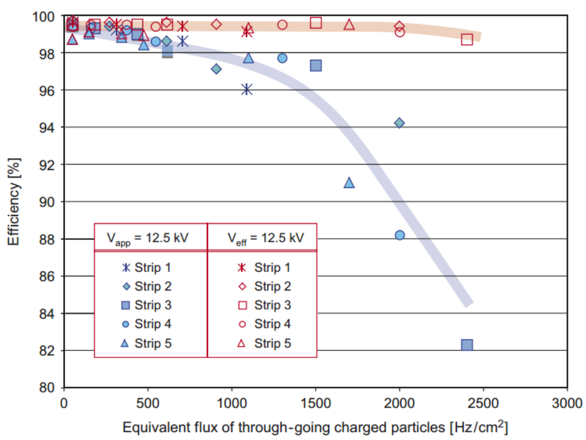


Figure 4.8: Comparison of the detector performance of ALICE ToF MRPC [31] at fixed applied voltage (in blue) and at fixed effective voltage (in red). The effective voltage is kept fixed by increasing the applied voltage accordingly to the current drawn by the detector.

4.3 Resistive Plate Chambers at CMS

4.3.1 Overview

The Resistive Plate Chambers (RPC) system, located in both barrel and endcap regions, provides a fast, independent muon trigger with a looser p_T threshold over a large portion of the pseudorapidity range ($|\eta| < 1.6$) [add reconstruction].

During High-Luminosity LHC (HL-LHC) operations the expected conditions in terms of background and pile-up will make the identification and correct P_T assignment a challenge for the Muon system. The goal of RPC upgrade is to provide additional hits to the Muon system with precise timing. All these informations will be elaborated by the trigger system in a global way enhancing the performance of the trigger in terms of efficiency and rate control. The RPC Upgrade is based on two projects: an improved Link Board System and the extension of the RPC coverage up to $|\eta| = 2.4$. [FIXME 2.4 or 2.5?]

The Link Board system, that will be described in section xxx, is responsible to process, synchronize and zero-suppress the signals coming from the RPC front end boards. The Link Board components have been produced between 2006 and 2007 and will be subjected to aging and failure in the long term. The upgraded Link Board system will overcome the aging problems described in section xxx and will allow for a more precise timing information to the RPC hits from 25 to 1 ns [ref section xxx].

The extension of the RPC system up to $|\eta| = 2.1$ was already planned in the CMS TDR [ref cmstdr] and staged because of budget limitations and expected background rates higher than the rate capability of the present CMS RPCs in that region. An extensive R&D program has been done in order to develop an improved RPC that fulfills the CMS requirements. Two new RPC layers in the innermost ring of stations 3 and 4 will be added with benefits to the neutron-induced background reduction and efficiency improvement for both trigger and offline reconstruction.

4.3.2 The present RPC system

The RPC system is organized in 4 stations called RB1 to RB4 in the barrel region, and RE1 to RE4 in the endcap region. The innermost barrel stations, RB1 and RB2, are instrumented with 2 layers of RPCs facing the innermost (RB1in and RB2in) and outermost (RB1out and RB2out) sides of the DT chambers. Every chamber is then divided from the read-out point of view into 2 or 3 η partitions called "rolls". The RPC system consist of 480 barrel chambers and 576 endcap chambers. Details on the geometry are discussed in the paper [ref to geo paper].

The CMS RPC chamber is a double-gap, operated in avalanche mode to ensure reliable operation at high rates. Each RPC gap consists of two 2-mm-thick resistive High-Pressure Laminate (HPL) plates separated by a 2-mm-thick gas gap. The outer surface of the HPL plates is coated with a thin conductive graphite layer, and a voltage is applied. The RPCs are operated with a 3-component, non-flammable gas mixture consisting of 95.2% freon ($C_2H_2F_4$, known as R134a), 4.5% isobutane (i-C_4H_{10}), and 0.3% sulphur hexafluoride (SF_6) with a relative humidity of 40% - 50%. Readout strips are aligned in η between the 2 gas gaps. [Add a sentence on FEBs.]

The discriminated signals coming from the Front End boards feed via twisted cables (10 to 20 m long) the Link Board System located in UXC on the balconies around the detector. The Link System consist of the 1376 Link Boards (LBs) and the 216 Control Boards (CBs), placed in 108 Link Boxes. The Link Box is a custom crate (6U high) with 20 slots (for two CBs and eighteen LBs). The Link Box contains custom backplane to which the cables from the chambers are connected, as well as the cables providing the LBs and CBs power supply and the cables for the RPC FEBs control with use of the I2C protocol (trough the CB). The backplane itself contains only connectors (and no any other electronic devices).

The Link Board has 96 input channels (one channel corresponds to one RPC strip). The input signals are the $\sim$100 ns binary pulses which are synchronous to the RPC hits, but not to the LHC clock (which drives the entire CMS electronics). Thus the first step of the FEB signals processing is synchronization, i.e. assignment of the signals to the BXes (25 ns periods). Then the data are compressed with a simple zero-suppressing algorithm (the input channels are grouped into 8 bit partitions, only the partitions with at least one nonzero bit are selected for each BX). Next, the non-empty partitions are time-multiplexed i.e. if there are more than one such partition in a given BX, they are sent one-by-one in consecutive BXes. The data from 3 neighbouring LBs are concentrated by the middle LB which contains the optical transmitter for sending them to the USC over a fiber at $1.6\,\mathrm{Gbps}$.

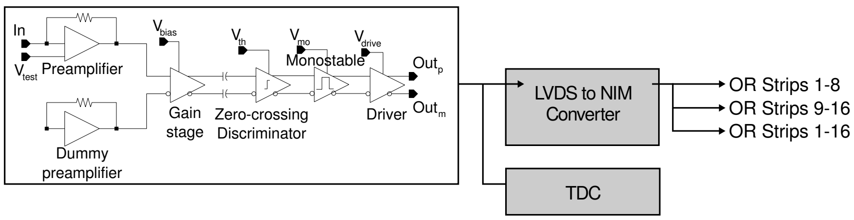
The Control Boards provide the communication of the control software with the LBs via the FEC/CCU system. The CBs are connected into token rings, each ring consists of 12 CBs of one detector tower and a FEC mezzanine board placed on the CCS board located in the VME crate in the USC. In total, there are 18 rings in the entire Link System. The CBs also perform automatic reloading of the LB's firmware which is needed in order to avoid accumulation of the radiation induced SEUs in the LBs firmware.

Both LBs and CB are based on the Xilinx Spartan III FPGAs, the CB additionally contains radiation-tolerant (FLASH based) FPGA Actel ProAsicPlus.

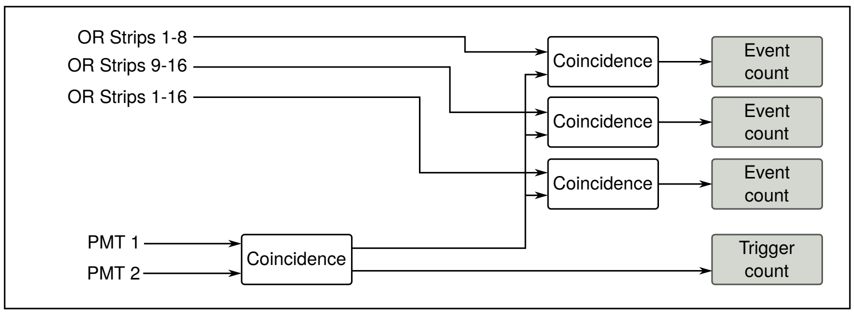
The High Voltage power system is located in USC, not exposed to radiation and easily accessible for any reparation. A single HV channel powers 2 RPC chambers both in the barrel and endcap regions. The Low Voltage boards are located in UXC on the balconies and provide the voltage to the front end electronics.

4.3.3 Pulse processing of CMS RPCs

Signals induced by cosmic particle in the RPC strips are shaped by standard CMS RPC Front-End Electronics (FEE) following the scheme of Figure 4.9. On a first stage, analogic signals are amplified and then sent to the Constant Fraction Discriminator (CFD) described in Figure 4.10. At the end of the chain, $100\,\mathrm{ns}$ long pulses are sent in the LVDS output. These output signal are sent on one side to a V1190A Time-to-Digital Converter (TDC) module from CAEN and on the other to an OR module to count the number of detected signals. Trigger and hit coïncidences are monitored using scalers. The TDC is used to store the data into ROOT files. These files are thus analysed to understand the detectors performance.

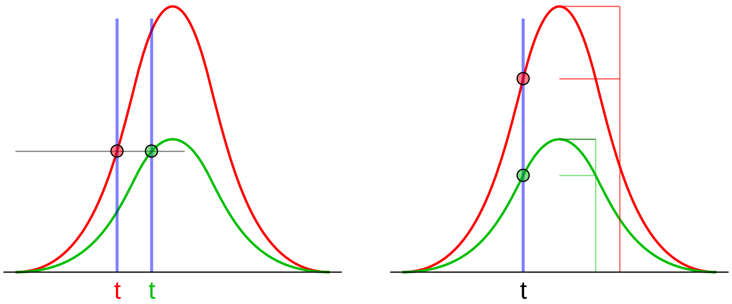


(a)

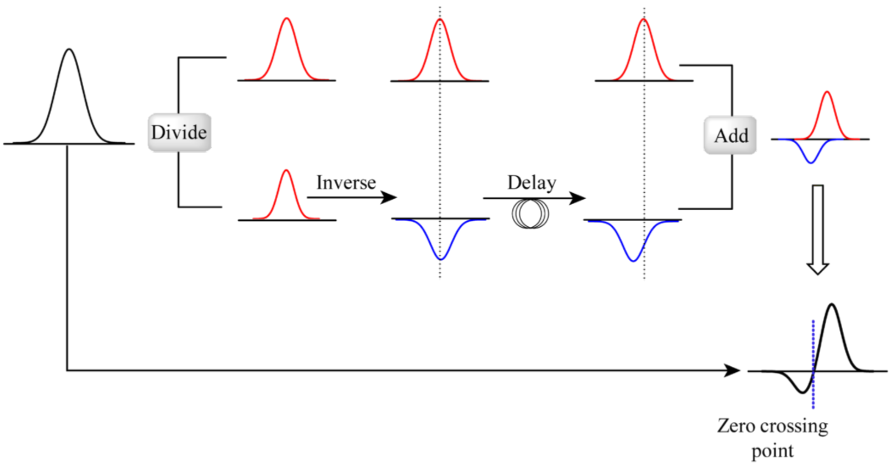


(b)

Figure 4.9: Signals from the RPC strips are shaped by the FEE described on Figure 4.9a. Output LVDS signals are then read-out by a TDC module connected to a computer or converted into NIM and sent to scalers. Figure 4.9b describes how these converted signals are put in coincidence with the trigger.



(a)



(b)

Figure 4.10: Description of the principle of a CFD. A comparison of threshold triggering (left) and constant franction triggering (right) is shown in Figure 4.10a. Constant franction triggering is obtained thanks to zero-crossing technique as explained in Figure 4.10b. The signal arriving at the input of the CFD is split into three components. A first one is delayed and connected to the inverting input of a first comparator. A second component is connected to the noninverting input of this first comparator. A third component is connected to the noninverting input of another comparator along with a threshold value connected to the inverting input. Finally, the output of both comparators is fed through an AND gate.

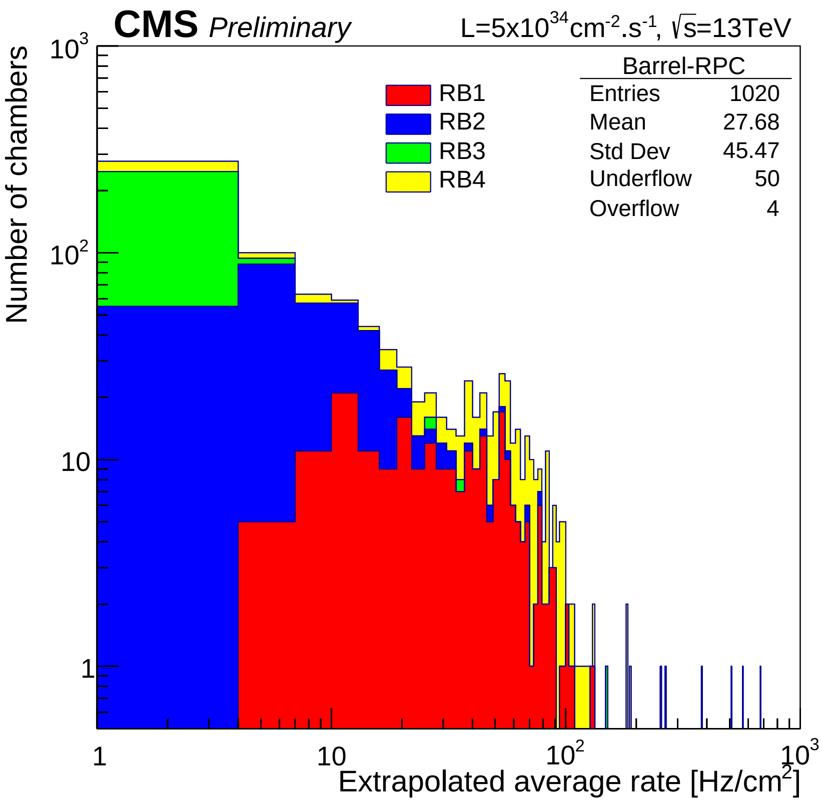
5

Longevity studies and Consolidation of the present CMS RPC subsystem

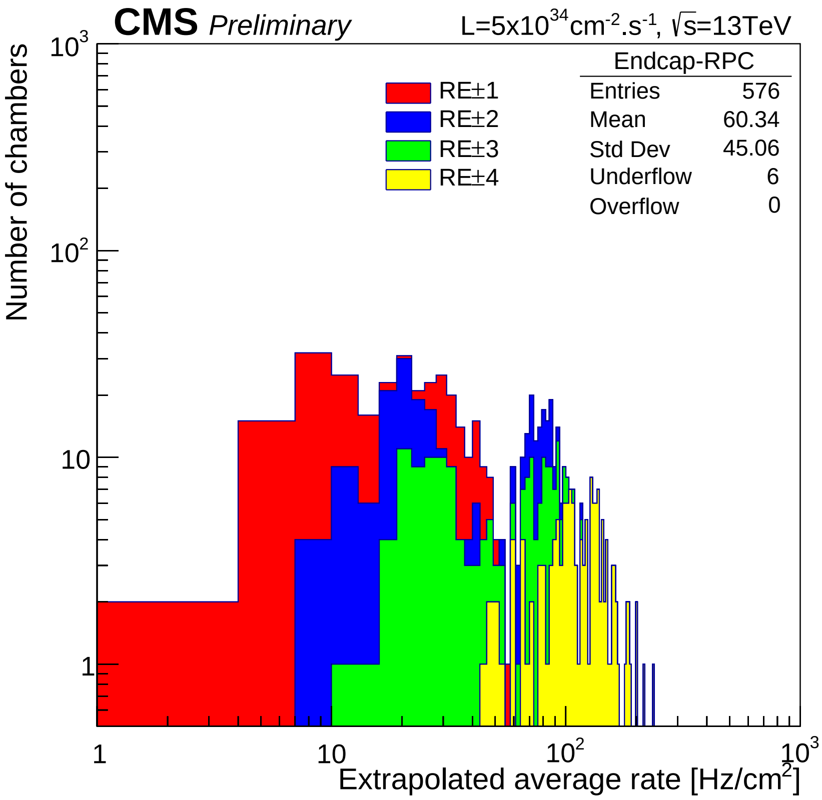
5.1 Testing detectors under extreme conditions

The upgrade from LHC to HL-LHC will increase the peak luminosity from $10^{34}\ \mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ to reach $5{\times}10^{34}\ \mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, increasing in the same way the total expected background to which the RPC system will be subjected to. Composed of low energy gammas and neutrons from p-p collisions, low momentum primary and secondary muons, puch-through hadrons from calorimeters, and particles produced in the interaction of the beams with collimators, the background will mostly affect the regions of CMS that are the closest to the beam line, i.e. the RPC detectors located in the endcaps. [To update.]

The 2016 data allowed to study the values of the background rate in all RPC system. In Figure 5.1, the distribution of the chamber background hit rate per unit area is shown at a luminosity of $5 \times 10^{34} cm^{-2}.s^{-1}$ linearly extrapolating from data collected in 2016 [ref mentioning the linear dependency of rate vs lumi]. The maximum rate per unit area at HL-LHC conditions is expected to be of the order of $600 Hz/cm^2$ (including a safety factor 3). Nevertheless, Fluka simulations have conducted in order to understand the background at HL-LHC conditions. The comparison to the data has shown, in Figure 5.2, a discrepancy of a factor 2 even though the order of magnitude is consistent. [Understand mismatch.]



(a)



(b)

Figure 5.1: (5.1a) Extrapolation from 2016 data of single hit rate per unit area in the barrel region. (5.1b) Extrapolation from 2016 data of single hit rate per unit area in the endcap region.

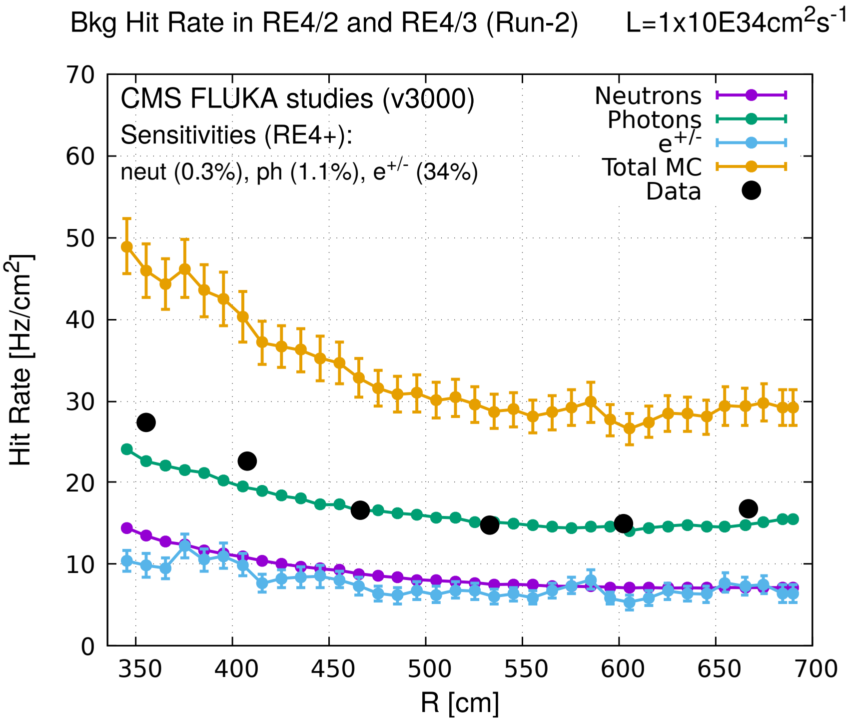


Figure 5.2: Background Fluka simulation compared to 2016 Data at $L = 10^{34} cm^{-2}.s^{-1}$ in the fourth endcap disk region. A mismatch in between simulation and data can be observed. [To be understood.]

In the past, extensive long-term tests were carried out at several gamma and neutron facilities certifying the detector performance. Both full size and small prototype RPCs have been irradiated with photons up to an integrated charge of $\sim 0.05C/cm^2$ and $\sim 0.4C/cm^2$, respectively [32, 33]. During Run-I, the RPC system provided stable operation and excellent performance and did not show any aging effects for integrated charge of the order of $0.01C/cm^2$. Projections on currents from 2016 Data, has allowed to determine that the total integrated charge, by the end of HL-LHC, would be of the order of $1C/cm^2$ (including a safety factor 3). [Corresponding figure needed.]

5.1.1 The Gamma Irradiation Facilities

5.1.1.1 GIF

Located in the SPS West Area at the downstream end of the X5 test beam, the Gamma Irradiation Facility (GIF) was a test area in which particle detectors were exposed to a particle beam in presence of an adjustable gamma background [34]. Its goal was to reproduce background conditions these detectors would suffer in their operating environment at LHC. GIF layout is shown in Figure 5.3. Gamma photons are produced by a strong ^{137}Cs source installed in the upstream part of the zone inside a lead container. The source container includes a collimator, designed to irradiate a $6 \times 6\,\mathrm{m}^2$ area at $5\,\mathrm{m}$ maximum to the source. A thin lens-shaped lead filter helps providing with a uniform outcoming flux in a vertical plane, orthogonal to the beam direction. The principal collimator hole provides a pyramidal aperture of $74° \times 74°$ solid angle and provides a photon flux in a pyramidal volume along the beam axis. The photon rate is controled by further lead filters allowing the maximum rate to be limited and to vary within a range of four orders of magnitude. Particle detectors under test are then placed within the pyramidal volume in front of the source, perpendicularly to the beam line in order to profit from the homogeneous photon flux. Adjusting the background flux of photons can then be done by using the filters and choosing the position of the detectors with respect to the source.

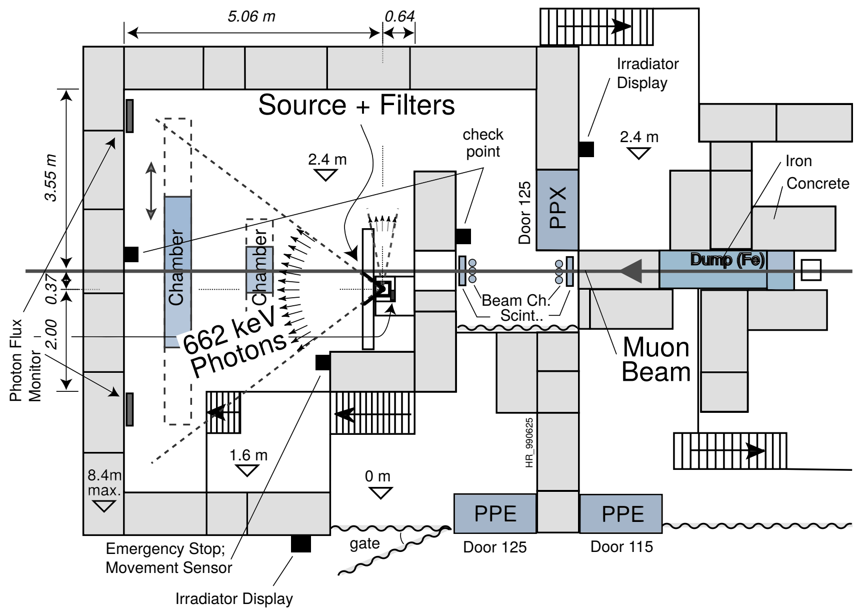


Figure 5.3: Layout of the test beam zone called X5c GIF at CERN. Photons from the radioactive source produce a sustained high rate of random hits over the whole area. The zone is surrounded by $8\,\mathrm{m}$ *high and* $80\,\mathrm{cm}$ *thick concrete walls. Access is possible through three entry points. Two access doors for personnel and one large gate for material. A crane allows installation of heavy equipment in the area.*

As described on Figure 5.4, the ^{137}Cs source emits a $662\,\mathrm{keV}$ photon in 85% of the decays. An activity of $740\,\mathrm{GBq}$ was measured on the 5^{th} March 1997. To estimate the strength of the flux in 2014, it is necessary to consider the nuclear decay through time assiciated to the Cesium source whose half-life is well known ($t_{1/2} = (30.05 \pm 0.08)\,\mathrm{y}$). The GIF tests where done in between the 20^{th} and the 31^{st} of August 2014, i.e. at a time $t = (17.47 \pm 0.02)\,\mathrm{y}$ resulting in an attenuation of the activity from $740\,\mathrm{GBq}$ in 1997 to $494\,\mathrm{GBq}$ in 2014.

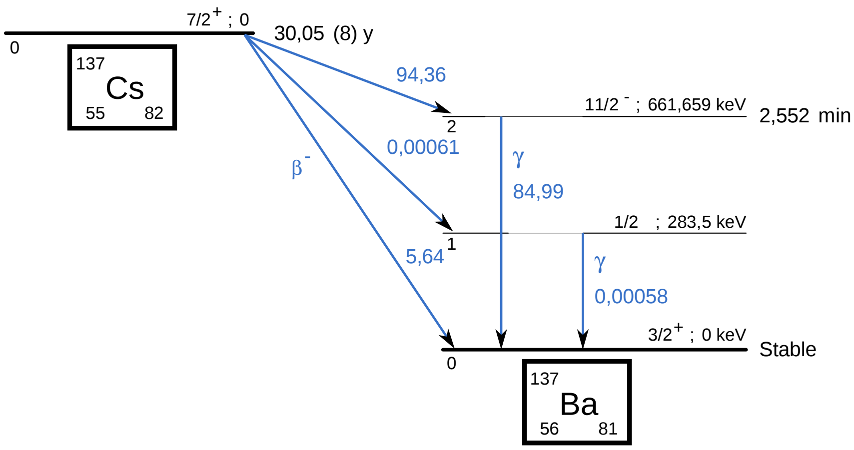


Figure 5.4: ^{137}Cs decays by β^- emission to the ground state of ^{137}Ba (BR = 5.64%) and via the 662 keV isomeric level of ^{137}Ba (BR = 94.36%) whose half-life is 2.55 min.

5.1.1.2 GIF++

The new Gamma Irradiation Facility (GIF++), located in the SPS North Area at the downstream end of the H4 test beam, has replaced its predecessor during LS1 and has been operational since spring 2015 [35]. Like GIF, GIF++ features a ^{137}Cs source of 662 keV gamma photons, their fluence being controlled with a set of filters of various attenuation factors. The source provides two separated large irradiation areas for testing several full-size muon detectors with continuous homogeneous irradiation, as presented in Figure 5.5.

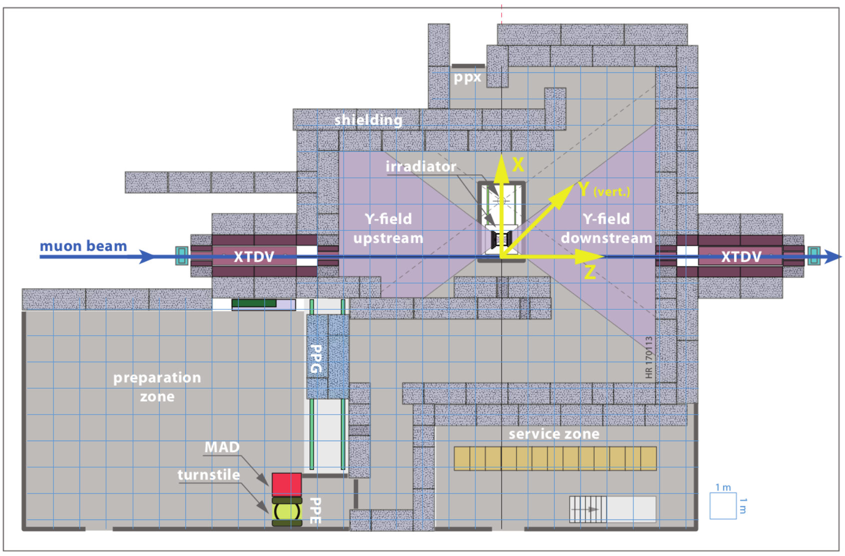
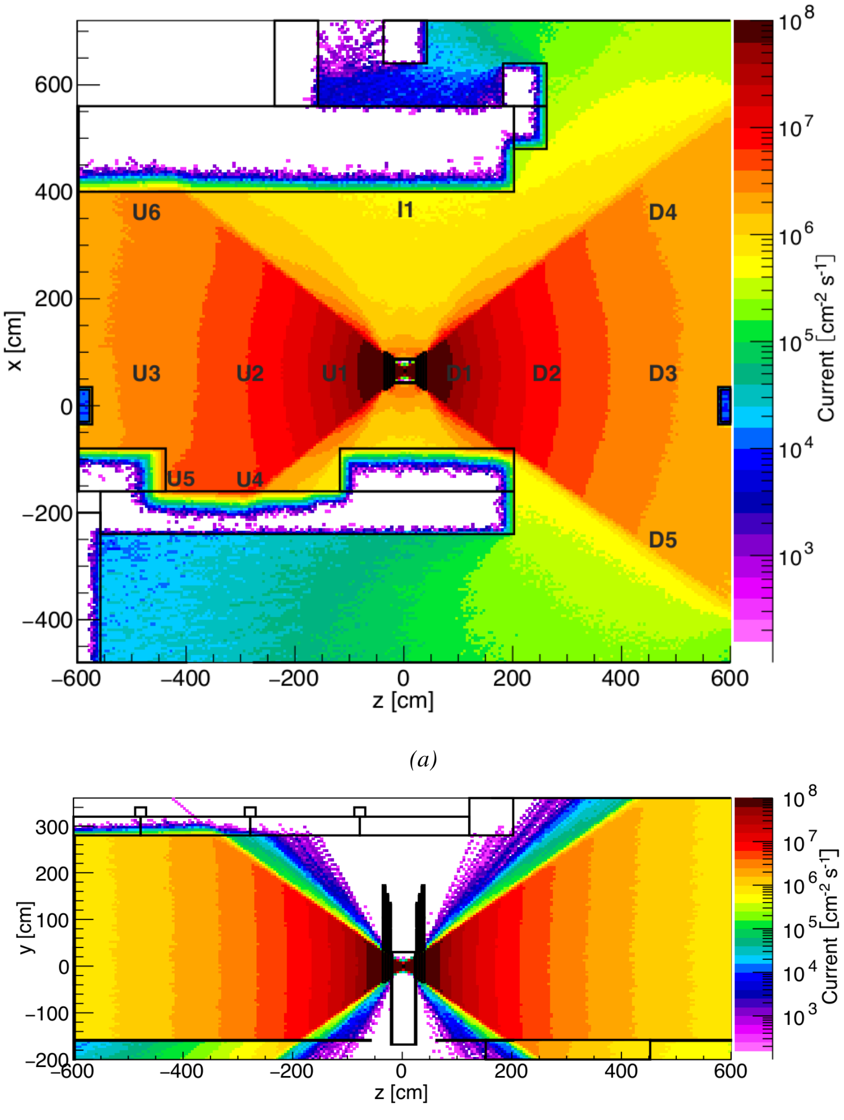


Figure 5.5: Floor plan of the GIF++ facility. When the facility downstream of the GIF++ takes electron beam, a beam pipe is installed along the beam line (z-axis). The irradiator can be displaced laterally (its center moves from x =0.65 m to 2.15 m), to increase the distance to the beam pipe.

The source activity was measured to be about $13.5\,\mathrm{TBq}$ in March 2016. The photon flux being far greater than HL-LHC expectations, GIF++ provides an excellent facility for accelerated aging tests of muon detectors.



(a)

(b)

Figure 5.6: Simulated unattenuated current of photons in the xz plane (Figure 5.6a) and yz plane (Figure 5.6b) through the source at $x = 0.65\,\mathrm{m}$ and $y = 0\,\mathrm{m}$. With angular correction filters, the current of $662\,\mathrm{keV}$ photons is made uniform in xy planes.

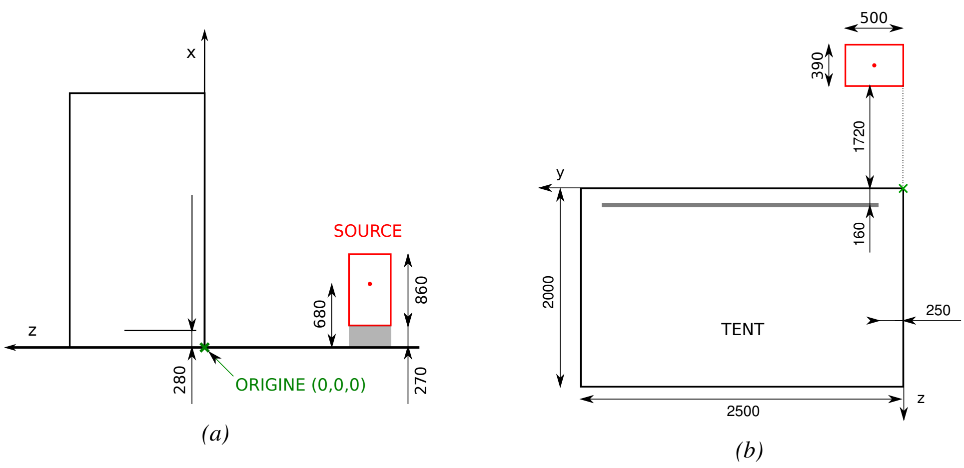
The source is situated in the muon beam line with the muon beam being available a few times a year. The H4 beam, composed of muons with a momentum of about $150\,\mathrm{GeV/c}$, passes through the GIF++ zone and is used to study the performance of the detectors. Its flux is of $104\,\mathrm{particles/s/cm^2}$

focused in an area similar to $10 \times 10\,\mathrm{cm}^2$. Therefore, with properly adjusted filters, one can imitate the HL-LHC background and study the performance of muon detectors with their trigger/readout electronics in HL-LHC environment.

5.2 Preliminary tests at GIF

5.2.1 Resistive Plate Chamber test setup

During summer 2014, preliminary tests have been conducted in the GIF area on a newly produced RE4/2 chamber labelled RE-4-2-BARC-161. This chamber has been placed into a trolley covered with a tent. The position of the RPC inside the tent and of the tent related to the source is described in Figure 5.7. To test this CMS RPC, three different absorber settings were used. First of all, measurements were done with fully opened source. Then, to complete this preliminary study, the gamma flux has been attenuated from a factor 2 and a factor 5. The expected gamma flux at the level of our detector will be discussed in subsection **??**.



(a) *(b)*

Figure 5.7: Description of the RPC setup. Dimensions are given in mm. *A tent containing RPCs is placed at* 1720 mm *from the source container. The source is situated in the center of the container. RE-4-2-BARC-161 chamber is* 160 mm *inside the tent. This way, the distance between the source and the chambers plan is* 2060 mm. *Figure 5.7a provides a side view of the setup in the xz plane while Figure 5.7b shows a top view in the yz plane.*

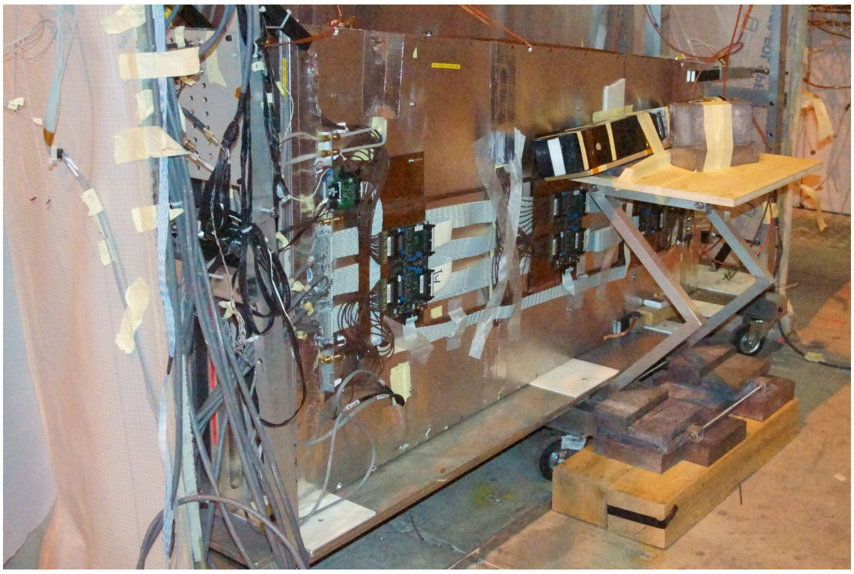


Figure 5.8: RE-4-2-BARC-161 chamber is inside the tent as described in Figure 5.7. In the top right, the two scintillators used as trigger can be seen. This trigger system has an inclination of $10°$ relative to horizontal and is placed above half-partition B2 of the RPCs. PMT electronics are shielded thanks to lead blocks placed in order to protect them without stopping photons from going through the scintillators and the chamber.

At the time of the tests, the beam not being operationnal anymore, a trigger composed of 2 plastic scintillators has been placed in front of the setup with an inclination of $10\,\mathrm{deg}$ with respect to the detector plane in order to look at cosmic muons. Using this particular trigger layout, shown on Figure 5.8, leads to a cosmic muon hit distribution into the chamber similar to the one in Figure 5.9. Measured without gamma irradiation, two peaks can be seen on the profil of partition B, centered on strips 52 and 59. Section **??** will help us undertand that these two peaks are due respectively to forward and backward coming cosmic particles where forward coming particles are first detected by the scintillators and then the RPC while the backward coming muons are first detected in the RPC.

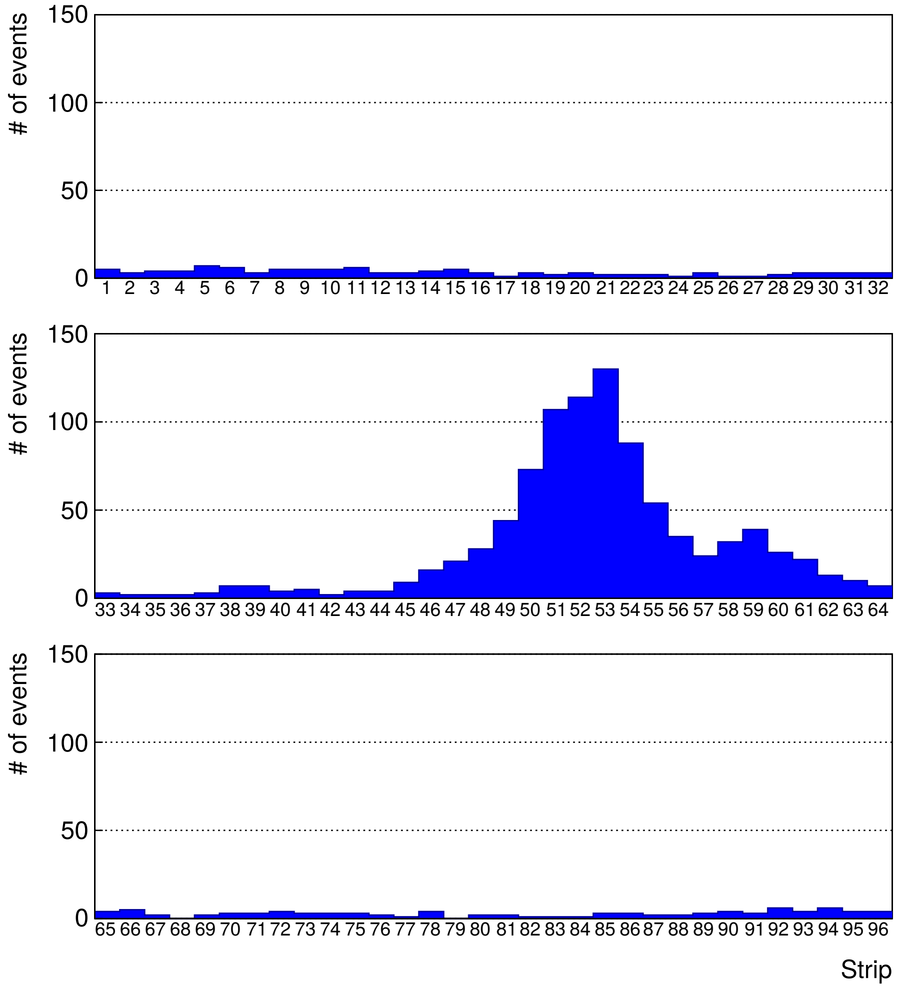


Figure 5.9: Hit distributions over all 3 parttions of RE-4-2-BARC-161 chamber is showed on these plots. Top, middle and bottom figures respectively correspond to partitions A, B, and C. These plots show that some events still occur in other half-partitions than B2, which corresponds to strips 49 to 64, in front of which the trigger is placed, contributing to the inefficiency of detection of cosmic muons. In the case of partitions A and C, the very low amount of data can be interpreted as noise. On the other hand, it is clear that a little portion of muons reach the half-partition B1, corresponding to strips 33 to 48.

5.2.2 Data Acquisition

5.2.3 Geometrical acceptance of the setup layout to cosmic muons

In order to profit from a constant gamma irradiation, the detectors inside of the GIF bunker need to be placed in a plane orthogonal to the beam line. The muon beam that used to be available was meant to test the performance of detectors under test. This beam not being active anymore, another solution to test detector performance had to be used. Thus, it has been decided to use cosmic muons detected through a telescope composed of two scintillators. Lead blocks were used as shielding to protect the photomultipliers from gammas as can be seen from Figure 5.8.

An inclination has been given to the cosmic telescope to maximize the muon flux. A good compromise had to be found between good enough muon flux and narrow enough hit distribution to be sure to contain all the events into only one half partitions as required from the limited available readout hardware. Nevertheless, a consequence of the misplaced trigger, that can be seen as a loss of events in half-partition B1 in Figure 5.9, is an inefficiency. Nevertheless, the innefficiency of approximately 20 % highlighted in Figure 5.10 by comparing the performance of chamber BARC-161 in 904 and at GIF without irradiation seems too important to be explained only by the geometrical acceptance of the setup itself. Simulations have been conducted to show how the setup brings inefficiency.

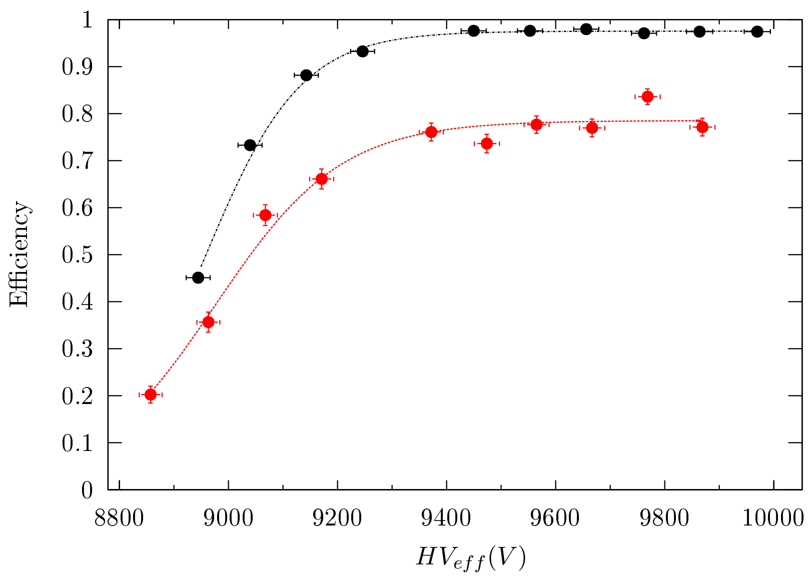
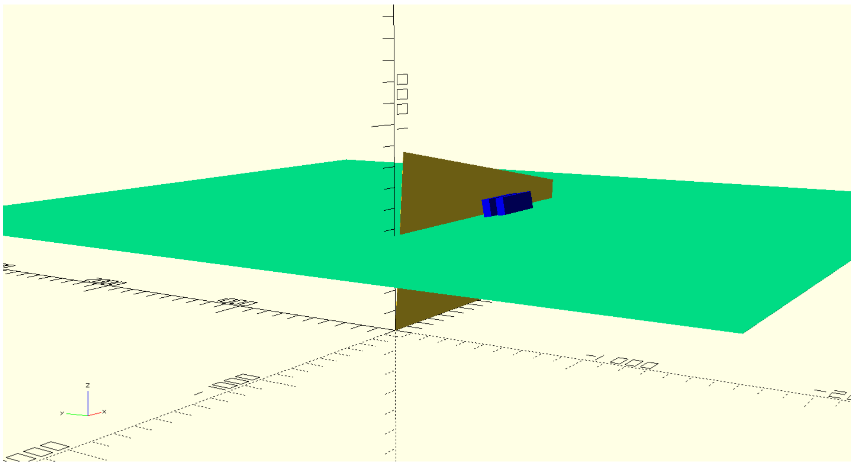


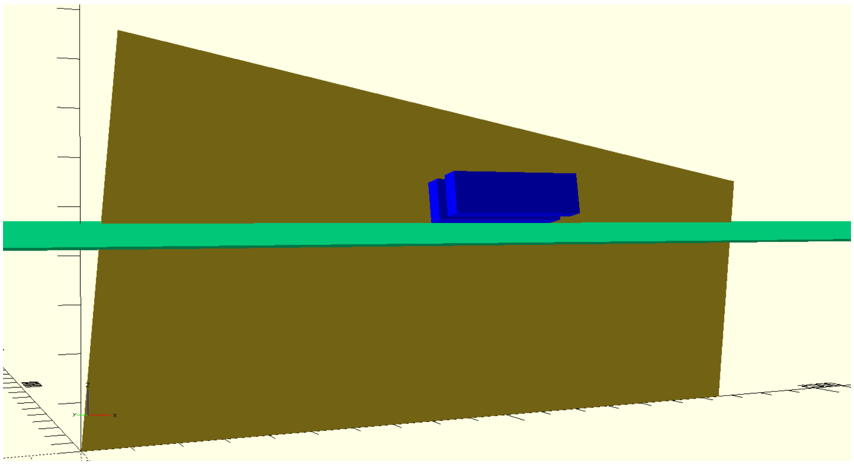
Figure 5.10: Results are derived from data taken on half-partition B2 only. On the 18th of June 2014, data has been taken on chamber RE-2-BARC-161 at building 904 (Prevessin Site) with cosmic muons providing us a reference efficiency plateau of (97.54 ± 0.15)*% represented by a black curve. A similar measurement has been done at GIF on the 21st of July with the same chamber giving a plateau of* (78.52 ± 0.94)*% represented by a red curve.*

5.2.3.1 Description of the simulation layout

The layout of GIF setup has been reproduced and incorporated into a Monte Carlo (MC) simulation to study the influence of the disposition of the telescope on the final distribution measured by the RPC. A 3D view of the simulated layout is given into Figure 5.11. Muons are generated randomly in a horizontal plane located at a height corresponding to the lowest point of the PMTs. This way, the needed size of the plane in order to simulate events happening at very big azimutal angles (i.e. $\theta \approx \pi$) can be kept relatively small. The muon flux is designed to follow the usual $cos^2\theta$ distribution for cosmic particle. The goal of the simulation is to look at muons that pass through the muon telescope composed of the two scintilators and define their distribution onto the RPC plane. During the reconstruction, the RPC plane is then divided into its strips and each muon track is assigned to a strip.



(a)



(b)

Figure 5.11: Representation of the layout used for the simulations of the test setup. The RPC is represented as a yellow trapezoid while the two scintillators as blue cuboids looking at the sky. A green plane corresponds to the muon generation plane within the simulation. Figure 5.7a shows a global view of the simulated setup. Figure 5.7b shows a zommed view that allows to see the 2 scintillators as well as the full RPC plane.

In order to further refine the quality of the simulation and understand deeper the results the dependance of the distribution has been studied for a range of telescope inclinations. Moreover, the threshold applied on the PMT signals has been included into the simulation in the form of a cut. In the approximation of uniform scintillators, it has been considered that the threshold can be understood as the minimum distance particles need to travel through the scintillating material to give a strong enough signal. Particles that travel a distance smaller than the set "threshold" are thus not detected by the telescope and cannot trigger the data taking. Finally, the FEE threshold also has been considered in a similar way. The mean momentum of horizontal cosmic rays is higher than those of vertical ones but the stopping power of matter for momenta ranging from $1\,\mathrm{GeV}$ to $1\,\mathrm{TeV}$ stays comparable. It is then possible to assume that the mean number of primary e^{-}/ion pairs per unit length will stay similar and thus, depending on the applied discriminator threshold, muons with the shortest path through the gas volume will deposit less charge and induce a smaller signal on the pick-up strips that could eventually not be detected. These two thresholds also restrain the overall geometrical acceptance of the system.

5.2.3.2 Simulation procedure

The simulation software has been designed using C++ and the output data is saved into ROOT histograms. Simulations start for a threshold T_{scint} varying in a range from 0 to $45\,\mathrm{mm}$ in steps of $5\,\mathrm{mm}$, where $T_{scint} = 0\,\mathrm{mm}$ corresponds to the case where there isn't any threshold apply on the input signal while $T_{scint} = 45\,\mathrm{mm}$, which is the scintillator thickness, is the case where muons cannot arrive orthogonally onto the scintillator surface. For a given T_{scint}, a set of RPC thresholds are considered. The RPC threshold, T_{RPC} varies from $2\,\mathrm{mm}$, the thickness of the gas volume, to $3\,\mathrm{mm}$ in steps of $0.25\,\mathrm{mm}$. For each (T_{scint};T_{RPC}) pair, $N_\mu = 10^8$ muons are randomly generated inside the muon plane described in the previous paragraph with an azimutal angle θ chosen to follow a $cos^2\theta$ distribution.

Planes are associated to each surface of the scintillators. Knowing muon position into the muon plane and its direction allows us, by assuming that muons travel in a straight line, to compute the intersection of the muon track with these planes. Applying conditions to the limits of the surfaces of the scintillator faces then gives us an answer to weither or not the muon passed through the scintillators. In the case the muon has indeed passed through the telescope, the path through each scintillator is computed and muons whose path was shorter than T_{scint} are rejected and are thus considered as having not interacted with the setup.

On the contrary, if the muon is labeled as good, its position within the RPC plane is computed and the corresponding strip, determined by geometrical tests in the case the distance through the gas volume was enough not to be rejected because of T_{RPC}, gets a hit and several histograms are filled in order to keep track of the generation point on the muon plane, the intersection points of the reconstructed muons within the telescope, or on the RPC plane, the path traveled through each individual scintillator or the gas volume, as well as other histograms. Moreover, muons fill different histograms weither they are forward or backward coming muons. They are discriminated according to their direction components. When a muon is generated, an (x, y, z) position is assigned into the muon plane as well as a (θ;ϕ) pair that gives us the direction it's coming from. This way, muons satisfying the condition $0 \leq \phi < \pi$ are designated as backward coming muons while muons satisfying $\pi \leq \phi < 2\pi$ as forward coming muons.

This simulation is then repeated for different telescope inclinations ranging in between 4 and $20°$ and varying in steps of $2°$. Due to this inclination and to the vertical position of the detector under test, the muon distribution reconstructed in the detector plane is asymmetrical. The choice as been made to chose a skew distribution formula to fit the data built as the multiplication of gaussian and sigmoidal curves together. A typical gaussian formula is given as 5.1 and has three free parameters as A_g, its amplitude, $\bar{x}$, its mean value and σ, its root mean square. Sigmoidal curves as given by formula 5.2 are functions converging to 0 and A_s as x diverges. The inflexion point is given as x_i and λ is proportional to the slope at $x = x_i$. In the limit where $\lambda \longrightarrow \infty$, the sigmoid becomes a step function.

$$g(x) = A_g e^{\frac{-(x-\bar{x})^2}{2\sigma^2}} \tag{5.1}$$

$$s(x) = \frac{A_s}{1 + e^{-\lambda(x - x_i)}} \tag{5.2}$$

Finally, a possible representation of a skew distribution is given by formula 5.3 and is the product of 5.1 and 5.2. Naturally, here $A_{sk} = A_g \times A_s$ and represents the theoretical maximum in the limit where the skew tends to a gaussian function.

$$sk(x) = g(x) \times s(x) = A_{sk} \frac{e^{\frac{-(x-\bar{x})^2}{2\sigma^2}}}{1 + e^{-\lambda(x - x_i)}} \tag{5.3}$$

5.2.3.3 Results

Influence of $\mathbf{T_{scint}}$ on the muon distribution

Influence of $\mathbf{T_{RPC}}$ on the muon distribution

Influence of the telescope inclination on the muon distribution

Comparison to data taken at GIF without irradiation

5.2.4 Photon flux at GIF

5.2.4.1 Expectations from simulations

In order to understand and evaluate the γ flux in the GIF area, simulations had been conducted in 1999 and published by S. Agosteo et al [34]. Table 5.1 presented in this article gives us the γ flux for different distances D to the source. This simulation was done using GEANT and a Monte Carlo N-Particle (MCNP) transport code, and the flux F is given in number of γ per unit area and unit time along with the estimated error from these packages expressed in %.

Nominal	Photon flux F [$\mathrm{s^{-1}cm^{-2}}$]			
ABS	at D = 50 cm	at D = 155 cm	at D = 300 cm	at D = 400 cm
1	$0.12 \cdot 10^8 \pm 0.2\%$	$0.14 \cdot 10^7 \pm 0.5\%$	$0.45 \cdot 10^6 \pm 0.5\%$	$0.28 \cdot 10^6 \pm 0.5\%$
2	$0.68 \cdot 10^7 \pm 0.3\%$	$0.80 \cdot 10^6 \pm 0.8\%$	$0.25 \cdot 10^6 \pm 0.8\%$	$0.16 \cdot 10^6 \pm 0.6\%$
5	$0.31 \cdot 10^7 \pm 0.4\%$	$0.36 \cdot 10^6 \pm 1.2\%$	$0.11 \cdot 10^6 \pm 1.2\%$	$0.70 \cdot 10^5 \pm 0.9\%$

Table 5.1: Total photon flux ($E\gamma \leq 662\,\mathrm{keV}$) with statistical error predicted considering a ^{137}Cs activity of $740\,\mathrm{GBq}$ *at different values of the distance D to the source along the x-axis of irradiation field [34].*

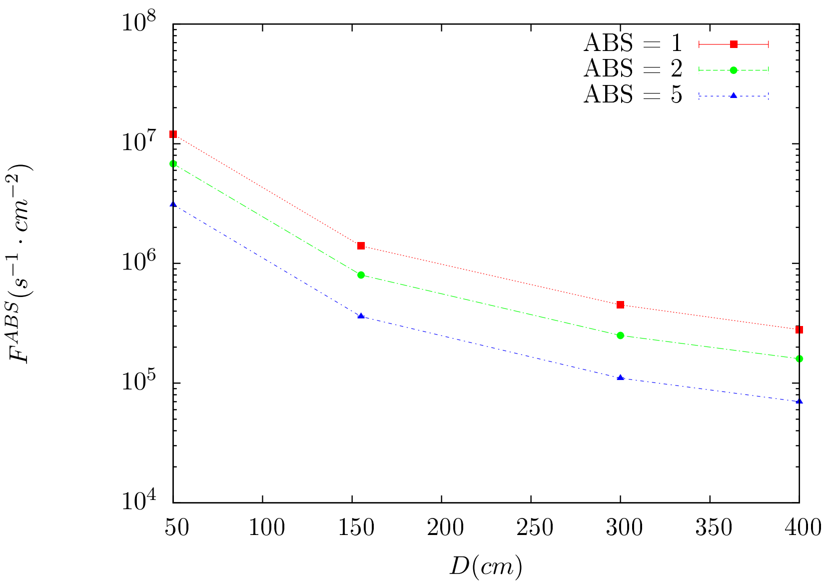


Figure 5.12: γ flux $F(D)$ is plot using values from table 5.1. As expected, the plot shows similar attenuation behaviours with increasing distance for each absorption factors.

The simulation doesn't directly provides us with an estimated flux at the level of our RPC. First of all, it is needed to extract the value of the flux from the available data contained in the original paper and then to estimate the flux in 2014 at the time the experimentation took place. Figure 5.12 that contains the data from Table 5.1. In the case of a pointlike source emiting isotrope and homogeneous gamma radiations, the gamma flux F at a distance D to the source with respect to a reference point situated at D_0 where a known flux F_0 is measured will be expressed like in Equation 5.4, assuming that the flux decreases as $1/D^2$, where c is a fitting factor.

$$F^{ABS} = F_0^{ABS} \times \left(\frac{cD_0}{D}\right)^2 \tag{5.4}$$

By rewriting Equation 5.4, it comes that :

$$c = \frac{D}{D_0}\sqrt{\frac{F^{ABS}}{F_0^{ABS}}} \tag{5.5}$$

$$\Delta c = \frac{c}{2}\left(\frac{\Delta F^{ABS}}{F^{ABS}} + \frac{\Delta F_0^{ABS}}{F_0^{ABS}}\right) \tag{5.6}$$

Finally, using Equation 5.5 and the data in Table 5.1 with $D_0 = 50\,\mathrm{cm}$ as reference point, we can build Table 5.2. It is interesting to note that c for each value of D doesn't depend on the absorption factor.

Nominal	Correction factor c		
ABS	at D = $155\,\mathrm{cm}$	at D = $300\,\mathrm{cm}$	at D = $400\,\mathrm{cm}$
1	$1.059 \pm 0.70\%$	$1.162 \pm 0.70\%$	$1.222 \pm 0.70\%$
2	$1.063 \pm 1.10\%$	$1.150 \pm 1.10\%$	$1.227 \pm 0.90\%$
5	$1.056 \pm 1.60\%$	$1.130 \pm 1.60\%$	$1.202 \pm 1.30\%$

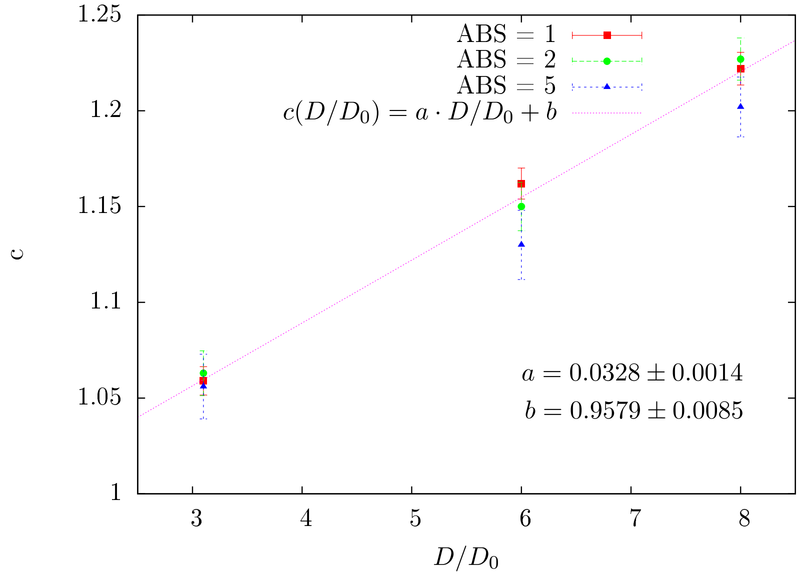
Table 5.2: Correction factor c is computed thanks to formulae 5.5 taking as reference $D_0 = 50\,\mathrm{cm}$ *and the associated flux* F_0^{ABS} *for each absorption factor available in table 5.1.*

For the range of D/D_0 values available, it is possible to use a simple linear fit to get the evolution of c. The linear fit will then use only 2 free parameters, a and b, as written in Equation 5.7. This gives us the results showed in Figure 5.13. Figure 5.13b confirms that using only a linear fit to extract c is enough as the evolution of the rate that can be obtained superimposes well on the simulation points.

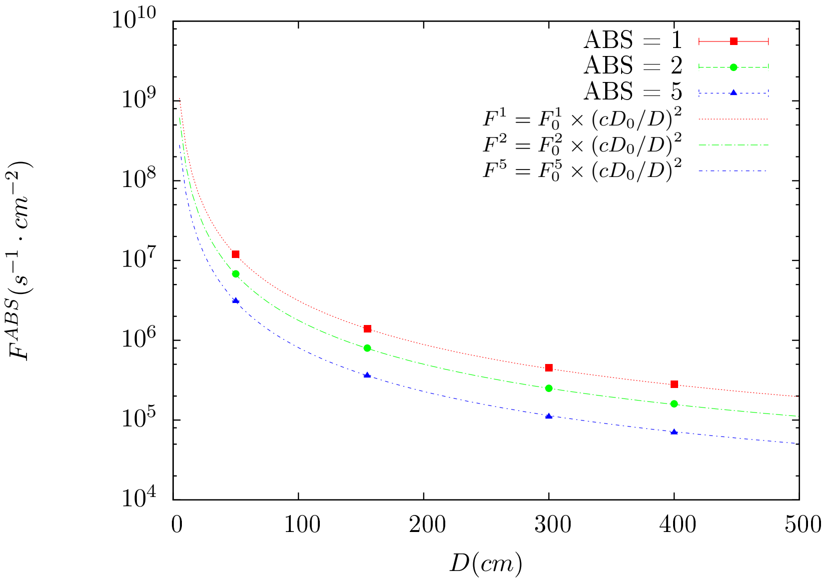
$$c\left(\frac{D}{D_0}\right) = a\frac{D}{D_0} + b \tag{5.7}$$

$$F^{ABS} = F_0^{ABS}\left(a + \frac{bD_0}{D}\right)^2 \tag{5.8}$$

$$\Delta F^{ABS} = F^{ABS}\left[\frac{\Delta F_0^{ABS}}{F_0^{ABS}} + 2\frac{\Delta a + \Delta b\frac{D_0}{D}}{a + \frac{bD_0}{D}}\right] \tag{5.9}$$



(a)



(b)

Figure 5.13: Figure 5.13a shows the linear approximation fit done via formulae 5.7 on data from table 5.2. Figure 5.13b shows a comparison of this model with the simulated flux using a and b given in figure 5.13a in formulae 5.4 and the reference value $D_0 = 50cm$ and the associated flux for each absorption factor F_0^{ABS} from table 5.1

In the case of the 2014 GIF tests, the RPC plane is located at a distance $D = 206\,\mathrm{cm}$ to the source. Moreover, to estimate the strength of the flux in 2014, it is necessary to consider the nuclear decay through time assiciated to the Cesium source whose half-life is well known ($t_{1/2} = (30.05 \pm 0.08)\,\mathrm{y}$). The very first source activity measurement has been done on the 5^{th} of March 1997 while the GIF tests where done in between the 20^{th} and the 31^{st} of August 2014, i.e. at a time $t = (17.47 \pm 0.02)\,\mathrm{y}$ resulting in an attenuation of the activity from $740\,\mathrm{GBq}$ in 1997 to $494\,\mathrm{GBq}$ in 2014. All the needed information to extrapolate the flux through our detector in 2014 has now been assembled, leading to the Table 5.3. It isinteresting to note that for a common RPC sensitivity to γ of $2 \cdot 10^{-3}$, the order of magnitude of the estimated hit rate per unit area is of the order of the kHz for the fully opened source. Moreover, taking profit of the two working absorbers, it will be possible to scan background rates at $0\,\mathrm{Hz}$, $\sim$$300\,\mathrm{Hz}$ as well as $\sim$$600\,\mathrm{Hz}$. Without source, a good estimate of the intrinsic performance will be available. Then at $300\,\mathrm{Hz}$, the goal will be to show that the detectors fulfill the performance certification of CMS RPCs. Then a first idea of the performance of the detectors at higher background will be provided with absorbtion factors 2 ($\sim$$600\,\mathrm{Hz}$) and 1 (no absorbtion). ***[Here I will also put a reference to the plot showing the estimated background rate at the level of RE3/1 in the case of HL-LHC but this one being in another chapter, I will do it later.]***

Nominal ABS	Photon flux F [$\mathrm{s}^{-1}\mathrm{cm}^{-2}$] at D_0^{1997} = $50\,\mathrm{cm}$	at D^{1997} = $206\,\mathrm{cm}$	at D^{2014} = $206\,\mathrm{cm}$	Hit rate/unit area [$\mathrm{Hz\,cm}^{-2}$] at D^{2014} = $206\,\mathrm{cm}$
1	$0.12 \cdot 10^8 \pm 0.2\%$	$0.84 \cdot 10^6 \pm 0.3\%$	$0.56 \cdot 10^6 \pm 0.3\%$	1129 ± 32
2	$0.68 \cdot 10^7 \pm 0.3\%$	$0.48 \cdot 10^6 \pm 0.3\%$	$0.32 \cdot 10^6 \pm 0.3\%$	640 ± 19
5	$0.31 \cdot 10^7 \pm 0.4\%$	$0.22 \cdot 10^6 \pm 0.3\%$	$0.15 \cdot 10^6 \pm 0.3\%$	292 ± 9

Table 5.3: The data at D_0 in 1997 is taken from [34]. In a second step, using Equations 5.8 and 5.9, the flux at D can be estimated in 1997. Then, taking into account the attenuation of the source activity, the flux at D can be estimated at the time of the tests in GIF in 2014. Finally, assuming a sensitivity of the RPC to γ $s = 2 \cdot 10^{-3}$, an estimation of the hit rate per unit area is obtained.

5.2.4.2 Dose measurements

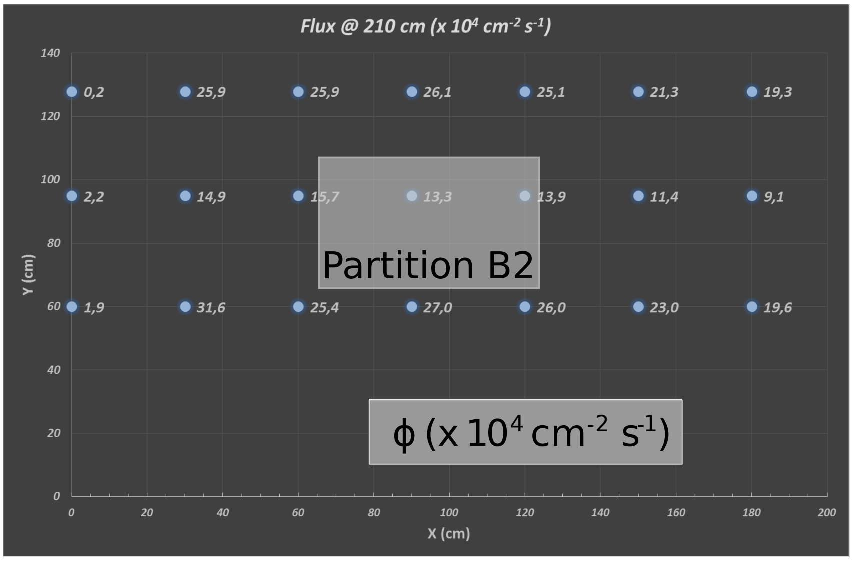


Figure 5.14: Dose measurements has been done in a plane corresponding to the tents front side. This plan is 1900 mm *away from the source. As explained in the first chapter, a lens-shaped lead filter provides a uniform photon flux in the vertical plan orthogonal to the beam direction. If the second line of measured fluxes is not taken into account because of lower values due to experimental equipments in the way between the source and the tent, the uniformity of the flux is well showed by the results.*

5.2.5 Results and discussions

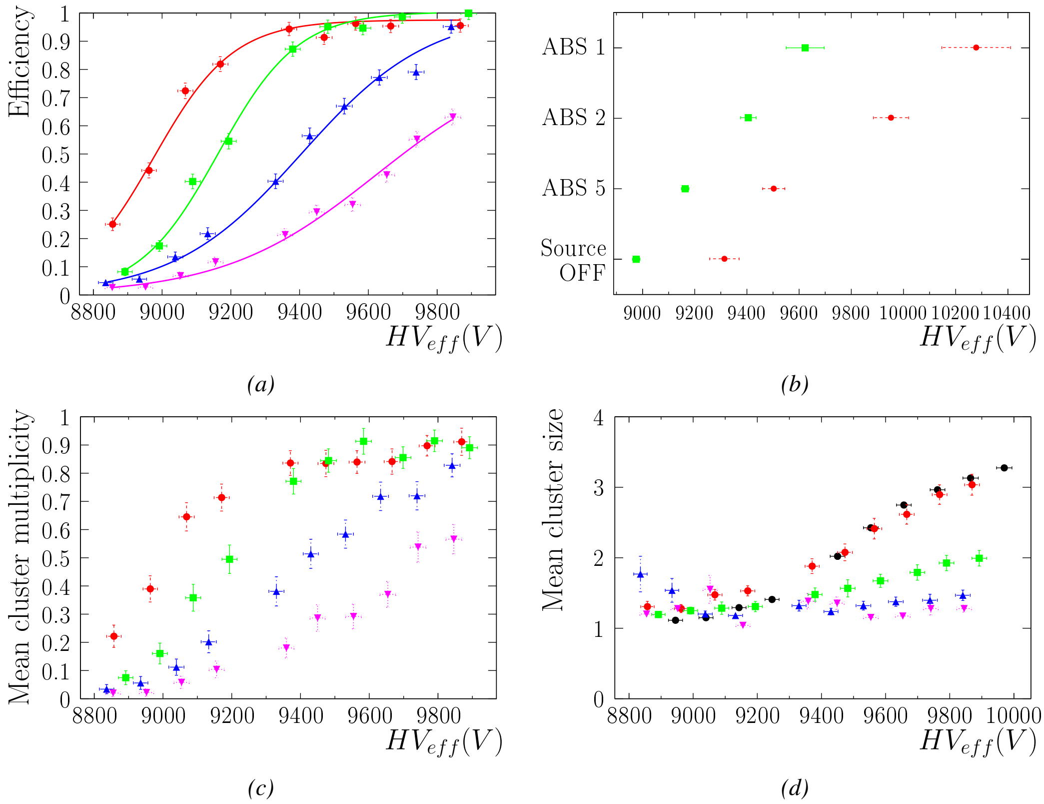
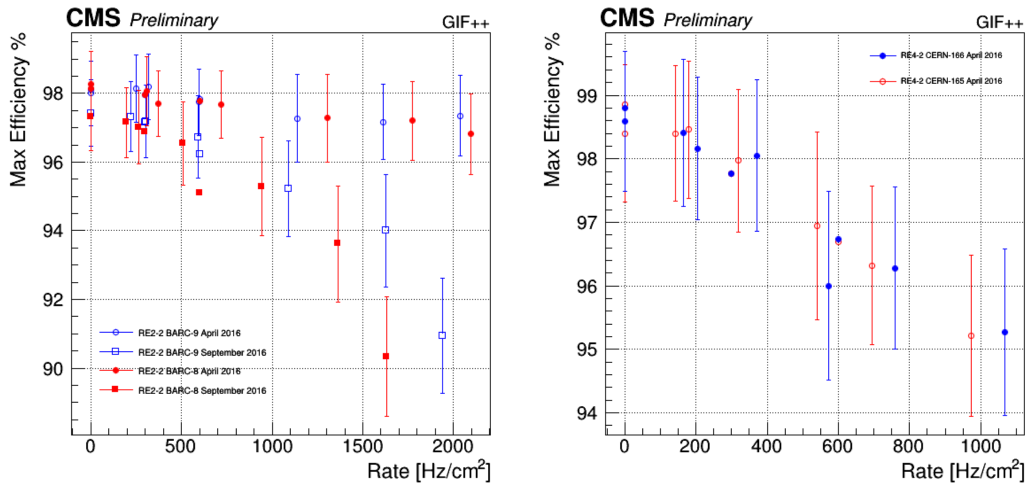


Figure 5.15

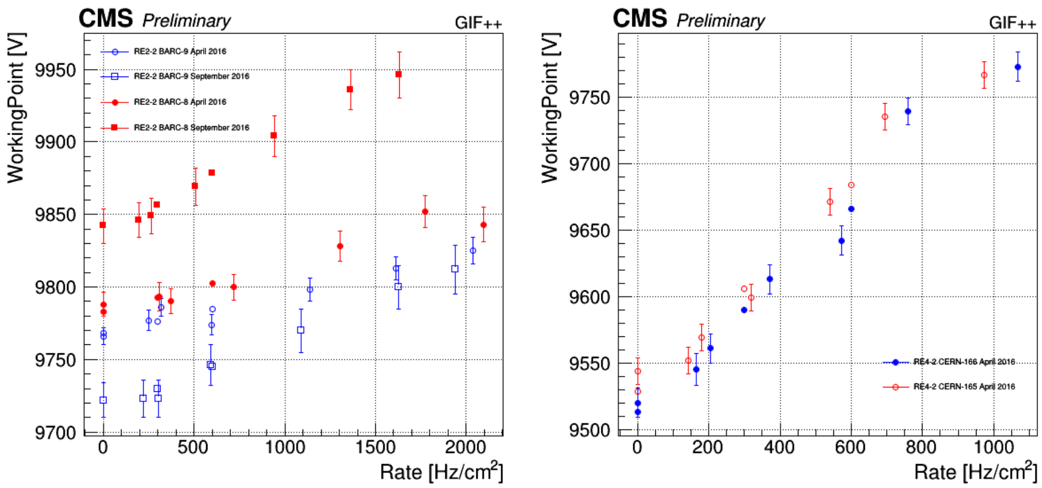
5.3 Longevity tests at GIF++

Longevity studies imply a monitoring of the performance of the detectors probed using a high intensity muon beam in a irradiated environment by periodically measuring their rate capability, the dark current running through them and the bulk resistivity of the Bakelite composing their electrodes. GIF++, with its very intense ^{137}Cs source, provides the perfect environment to perform such kind of tests. Assuming a maximum acceleration factor of 3, it is expected to accumulate the equivalent charge in 1.7 years.
As the maximum background is found in the endcap, the choice naturally was made to focus the GIF++ longevity studies on endcap chambers. Most of the RPC system was installed in 2007. Nevertheless, the large chambers in the fourth endcap (RE4/2 and RE4/3) have been installed during LS1 in 2014. The Bakelite of these two different productions having different properties, four spare chambers of the present system were selected, two RE2,3/2 spares and two RE4/2 spares. Having two chambers of each type allows to always keep one of them non irradiated as reference, the performance evolution of the irradiated chamber being then compared through time to the performance of the non irradiated one.
The performance of the detectors under different level of irradiation is measured periodically during dedicated test beam periods using the H4 muon beam. In between these test beam periods, the two RE2,3/2 and RE4/2 chambers selected for this study are irradiated by the ^{137}Cs source in order to accumulate charge and the gamma background is monitored, as well as the currents. The two remaining chambers are kept non-irradiated as reference detectors. Due to the limited gas flow in GIF++, the RE4 chamber remained non-irradiated until end of November 2016 where a new mass flow controller has been installed allowing for bigger volumes of gas to flow in the system.
Figures 5.16 and 5.17 give us for different test beam periods, and thus for increasing integrated charge through time, a comparison of the maximum efficiency, obtained using a sigmoid-like function, and of the working point of both irradiated and non irradiated chambers [**SIGMOID2005**]. No aging is yet to see from this data, the shifts in γ rate per unit area in between irradiated and non irradiated detectors and RE2 and RE4 types being easily explained by a difference of sensitivity due to the various Bakelite resistivities of the HPL electrodes used for the electrode production.
Collecting performance data at each test beam period allows us to extrapolate the maximum efficiency for a background hit rate of $300\,\mathrm{Hz/cm^2}$ corresponding to the expected HL-LHC conditions. Aging effects could emerge from a loss of efficiency with increasing integrated charge over time, thus Figure 5.18 helps us understand such degradation of the performance of irradiated detectors in comparison with non irradiated ones. The final answer for an eventual loss of efficiency is given in Figure 5.19 by comparing for both irradiated and non irradiated detectors the efficiency sigmoids before and after the longevity study. Moreover, to complete the performance information, the Bakelite resistivity is regularly measured thanks to Ag scans (Figure 5.20) and the noise rate is monitored weekly during irradiation periods (Figure 5.21). At the end of 2016, no signs of aging were observed and further investigation is needed to get closer to the final integrated charge requirements proposed for the longevity study of the present CMS RPC sub-system.



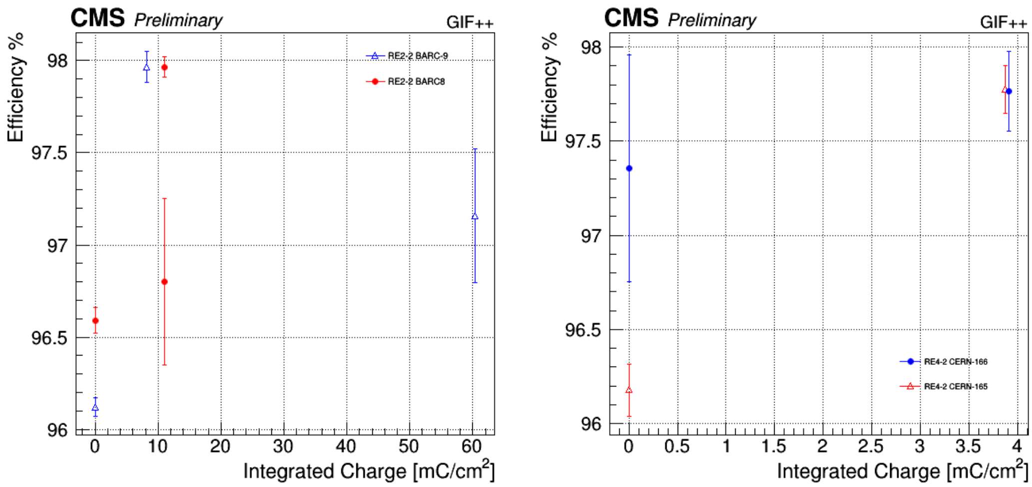
(a) (b)

Figure 5.16: Evolution of the maximum efficiency for RE2 (5.16a) and RE4 (5.16b) chambers with increasing extrapolated γ rate per unit area at working point. Both irradiated (blue) and non irradiated (red) chambers are shown.



(a) (b)

Figure 5.17: Evolution of the working point for RE2 (5.17a) and RE4 (5.17b) with increasing extrapolated γ rate per unit area at working point. Both irradiated (blue) and non irradiated (red) chambers are shown.

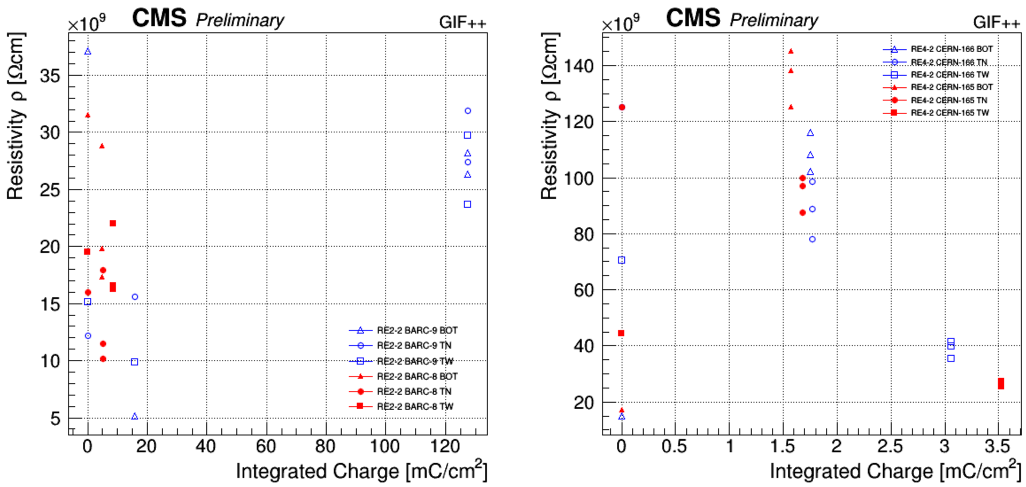


(a) *(b)*

Figure 5.18: Evolution of the maximum efficiency at HL-LHC conditions, i.e. a background hit rate per unit area of $300\,\mathrm{Hz/cm^2}$*, with increasing integrated charge for RE2 (5.18a) and RE4 (5.18b) detectors. Both irradiated (blue) and non irradiated (red) chambers are shown. The integrated charge for non irradiated detectors is recorded during test beam periods and stays small with respect to the charge accumulated in irradiated chambers.*

(a) *(b)*

Figure 5.19: Comparison of the efficiency sigmoid before (triangles) and after (circles) irradiation for RE2 (5.19a) and RE4 (5.19b) detectors. Both irradiated (blue) and non irradiated (red) chambers are shown.



(a) *(b)*

Figure 5.20: Evolution of the Bakelite resistivity for RE2 (5.20a) and RE4 (5.20b) detectors. Both irradiated (blue) and non irradiated (red) chambers are shown.

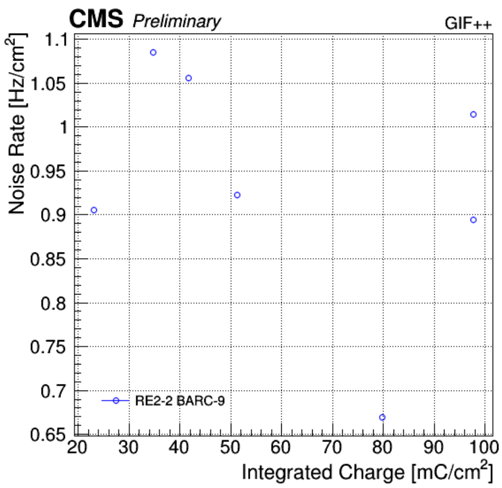


Figure 5.21: Evolution of the noise rate per unit area for the irradiated chamber RE2-2-BARC-9 only.

5.3.1 Description of the Data Acquisition

For the longevity studies, four spare chambers of the present system are used. Two spare RPCs of the RE2,3 stations as well as two spare RPCs from the new RE4 stations have been mounted in a Trolley. Six RE4 gaps are also placed in the trolley. The trolley is placed inside the GIF++ in the upstream region of the bunker, taking the cesium source as a reference. The trolley is oriented for the detection surface of the chambers to be orthogonal to the beam line. The system can be moved along the orthogonal plane in order to have the beam in all η-partitions. For the aging the trolley is moved outside the beam line and is placed in a distance of $5.2\,\mathrm{m}$ to the source, which irradiates the bunker using an attenuation filter of 2.2 which corresponds to a fluence of $10^7\mathrm{gamma/cm^2}$.

During GIF++ operation, the data collected can be divided into different categories as several parameters are monitored in addition to the usual RPC performance data. On one hand, to know the performance of a chamber, it is need to measure its efficiency and to know the background conditions in which it is operated. To do this, the hit signals from the chamber are recorded and stored in a ROOT file via a Data Acquisition (DAQ) software. On the other hand, it is also very important to monitor parameters such as environmental pressure and temperature, gas temperature and humidity, RPC HV, LV, and currents, or even source and beam status. This is done through the GIF++ web Detector Control Software (DCS) that stores this information in a database.

Two different types of tests are conducted on RPCs via the DAQ. Indeed, the performance of the detectors is measured periodically during dedicated test beam periods using the H4 muon beam. In between these test beam periods, when the beam is not available, the chambers are irradiated by the ^{137}Cs in order to accumulate deposited charge and the gamma background is measured.

RPCs under test are connected through LVDS cables to V1190A Time-to-Digital Converter (TDC) modules manufactured by CAEN. These modules, located in the rack area outside of the bunker, get the logic signals sent by the chambers and save them into their buffers. Due to the limited size of the buffers, the collected data is regularly erased and replaced. A trigger signal is needed for the TDC modules to send the useful data to the DAQ computer via a V1718 CAEN USB communication module.

In the case of performance test, the trigger signal used for data acquisition is generated by the coincidence of three scintillators. A first one is placed upstream outside of the bunker, a second one is placed downstream outside of the bunker, while a third one is placed in front of the trolley, close by the chambers. Every time a trigger is sent to the TDCs, i.e. every time a muon is detected, knowing the time delay in between the trigger and the RPC signals, signals located in the right time window are extracted from the buffers and saved for later analysis. Signals are taken in a time window of $400\,\mathrm{ns}$ centered on the muon peak (here we could show a time spectrum). On the other hand, in the case of background rate measurement, the trigger signal needs to be ”random” not to measure muons but to look at gamma background. A trigger pulse is continuously generated at a rate of $300\,\mathrm{Hz}$ using a dual timer. To integrate an as great as possible time, all signals contained within a time window of 10us prior to the random trigger signal are extracted form the buffers and saved for further analysis (here another time spectrum to illustrate could be useful, maybe even place both spectrum together as a single Figure).

The signals sent to the TDCs correspond to hit collections in the RPCs. When a particle hits a RPC, it induce a signal in the pickup strips of the RPC readout. If this signal is higher than the detection threshold, a LVDS signal is sent to the TDCs. The data is then organised into 4 branches keeping track of the event number, the hit multiplicity for the whole setup, and the time and channel profile of the hits in the TDCs.

5.3.2 RPC current, environmental and operation parameter monitoring

In order to take into account the variation of pressure and temperature between different data taking periods the applied voltage is corrected following the relationship :

$$HVeff = HVapp \times \left(0.2 + 0.8 \cdot \frac{P_0}{P} \times \frac{T}{T_0}\right) \tag{5.10}$$

where T_0 (=$293\,\mathrm{K}$) and P_0 (=$990\,\mathrm{mbar}$) are the reference values.

5.3.3 Measurement procedure

Insert a short description of the online tools (DAQ, DCS, DQM).
Insert a short description of the offline tools : tracking and efficiency algorithm.
Identify long term aging effects we are monitoring the rates per strip.

5.3.4 Longevity studies results

6

Investigation on high rate RPCs

6.1 Rate limitations and ageing of RPCs

6.1.1 Low resistivity electrodes

6.1.2 Low noise front-end electronics

6.2 Construction of prototypes

6.3 Results and discussions

7 Conclusions and outlooks

7.1 Conclusions

7.2 Outlooks

References

[1] CERN. Geneva. LHC Experiments Committee. *The CMS muon project : Technical Design Report*. Tech. rep. CERN-LHCC-97-032. CMS Collaboration, 1997.

[2] CERN. Geneva. LHC Experiments Committee. *Technical Proposal for the Phase-II Upgrade of the CMS Detector*. Tech. rep. CERN-LHCC-2015-010. CMS Collaboration, 2015.

[3] CERN. Geneva. LHC Experiments Committee. *CMS, the Compact Muon Solenoid : technical proposal*. Tech. rep. CERN-LHCC-94-38. CMS Collaboration, 1994.

[4] R. Santonico and R. Cardarelli. "Development of resistive plate counters". In: *Nucl. Instr. Meth. Phys. Res.* 187 (1981), pp. 377–380.

[5] Yu.N. Pestov and G.V. Fedotovich. *A picosecond time-of-flight spectrometer fot eh VEPP-2M based on local-discharge spark counter*. Tech. rep. SLAC-TRANS-0184. SLAC, 1978.

[6] W.W. Ash, ed. *Spark Counter With A Localized Discharge*. Vol. SLAC-R-250. 1982, pp. 127–131.

[7] I. Crotty et al. "The non-spark mode and high rate operation of resistive parallel plate chambers". In: *NIMA* 337 (1993), pp. 370–381.

[8] I. Crotty et al. "Further studies of avalanche mode operation of resistive parallel plate chambers". In: *NIMA* 346 (1994), pp. 107–113.

[9] R. Cardarelli et al. "Avalanche and streamer mode operation of resistive plate chambers". In: *NIMA* 382 (1996), pp. 470–474.

[10] E. Cerron Zeballos et al. "A new type of resistive plate chamber: The multigap RPC". In: *NIMA* 374 (1996), pp. 132–135.

[11] M.C.S. Williams. "The development of the multigap resistive plate chamber". In: *Nucl. Phys. B* 61 (1998), pp. 250–257.

[12] H. Czyrkowski et al. "New developments on resistive plate chambers for high rate operation". In: *NIMA* 419 (1998), pp. 490–496.

[13] P. Camarri et al. "Streamer suppression with SF6 in RPCs operated in avalanche mode". In: *NIMA* 414 (1998), pp. 317–324.

[14] E. Cerron Zeballos et al. "Effect of adding SF6 to the gas mixture in a multigap resistive plate chamber". In: *NIMA* 419 (1998), pp. 475–478.

[15] CERN. Geneva. LHC Experiments Committee. *ATLAS muon spectrometer: Technical design report*. Tech. rep. CERN-LHCC-97-22. ATLAS Collaboration, 1997.

[16] CERN. Geneva. LHC Experiments Committee. *ALICE Time-Of-Flight system (TOF) : Technical Design Report*. Tech. rep. CERN-LHCC-2000-012. ALICE Collaboration, 2000.

[17] The CALICE collaboration. "First results of the CALICE SDHCAL technological prototype". In: *JINST* 11 (2016).

[18] PoS, ed. *Density Imaging of Volcanoes with Atmospheric Muons using GRPCs*. International Europhysics Conference on High Energy Physics - HEP 2011. 2011.

[19] C. Lippmann. “Detector Physics of Resistive Plate Chambers”. PhD thesis. Johann Wolfgang Goethe-Universität, 2003.

[20] M. Abbrescia et al. “Properties of C2H2F4-based gas mixture for avalanche mode operation of resistive plate chambers”. In: *NIMA* 398 (1997), pp. 173–179.

[21] G.Battistoni et al. “Sensitivity of streamer mode to single ionization electrons”. In: *NIMA* 235 (1985), pp. 91–97.

[22] W. Riegler. “Induced signals in resistive plate chambers”. In: *NIMA* 491 (2002), pp. 258–271.

[23] E. Cerron Zeballos et al. “A comparison of the wide gap and narrow gap resistive plate chamber”. In: *NIMA* 373 (1996), pp. 35–42.

[24] M.Abbrescia et al. “Cosmic ray tests of double-gap resistive plate chambers for the CMS experiment”. In: *NIMA* 550 (2005), pp. 116–126.

[25] ALICE Collaboration. “A study of the multigap RPC at the gamma irradiation facility at CERN”. In: *NIMA* 490 (2002), pp. 58–70.

[26] B. Bonner et al. “A multigap resistive plate chamber prototype for time-of-flight for the STAR experiment at RHIC”. In: *NIMA* 478 (2002), pp. 176–179.

[27] S. Yang et al. “Test of high time resolution MRPC with different readout modes for the BESIII upgrade”. In: *NIMA* 763 (2014), pp. 190–196.

[28] A. Akindinovg et al. “RPC with low-resistive phosphate glass electrodes as a candidate for the CBM TOF”. In: *NIMA* 572 (2007), pp. 676–681.

[29] JINST, ed. *Development of the MRPC for the TOF system of the MultiPurpose Detector*. RPC2016: XII Workshop on Resistive Plate Chambers and Related Detectors. 2016.

[30] M.C.S. Williams. “Particle identification using time of flight”. In: *Journal of Physics G* 39 (2012).

[31] A. Alici et al. “Aging and rate effects of the Multigap RPC studied at the Gamma Irradiation Facility at CERN”. In: *NIMA* 579 (2007), pp. 979–988.

[32] M. Abbrescia et al. “Study of long-term performance of CMS RPC under irradiation at the CERN GIF”. In: *NIMA* 533 (2004), pp. 102–106.

[33] H.C. Kim et al. “Quantitative aging study with intense irradiation tests for the CMS forward RPCs”. In: *NIMA* 602 (2009), pp. 771–774.

[34] S. Agosteo et al. “A facility for the test of large-area muon chambers at high rates”. In: *NIMA* 452 (2000), pp. 94–104.

[35] PoS, ed. *CERN GIF ++ : A new irradiation facility to test large-area particle detectors for the high-luminosity LHC program*. Vol. TIPP2014. 2014, pp. 102–109.

[36] A. Fagot. *GIF++ DAQ v4.0*. 2017. URL: `https://github.com/afagot/GIF_DAQ`.

[37] CAEN. *Mod. V1190-VX1190 A/B, 128/64 Ch Multihit TDC*. 14th ed. 2016.

[38] CAEN. *Mod. V1718 VME USB Bridge*. 9th ed. 2009.

[39] W-Ie-Ne-R. *VME 6021-23 VXI*. 5th ed. 2016.

[40] Wikipedia. *INI file*. 2017. URL: `https://en.wikipedia.org/wiki/INI_file`.

[41] S. Carrillo A. Fagot. *GIF++ Offline Analysis v6*. 2017. URL: `https://github.com/afagot/GIF_OfflineAnalysis`.

A

A data acquisition software for CAEN VME TDCs

Certifying detectors in the perspective of HL-LHC required to develop tools for the GIF++ experiment. Among them was the C++ Data Acquisition (DAQ) software that allows to make the communications in between a computer and TDC modules in order to retrieve the RPC data [36]. In this appendix, details about this software, as of how the software was written, how it functions and how it can be exported to another similar setup, will be given.

A.1 GIF++ DAQ file tree

GIF++ DAQ source code is fully available on github at `https://github.com/afagot/GIF_DAQ`. The software requires 3 non-optional dependencies:

- CAEN USB Driver, to mount the VME hardware,
- CAEN VME Library, to communicate with the VME hardware, and
- ROOT, to organize the collected data into a TTree.

The CAEN VME library will not be packaged by distributions and will need to be installed manually. To compile the GIF++ DAQ project via a terminal, from the DAQ folder use the command:

```
make
```

The source code tree is provided below along with comments to give an overview of the files' content. The different objects created for this project (`v1718`, `v1190a`, `IniFile` & `DataReader`) will be described in details in the following sections.

```
GIF_DAQ
├── bin
│   └── daq ............................................................ EXECUTABLE
├── include .................................................. LIST OF C++ HEADER FILES
│   ├── CAENVMElib.h ................................................... CAEN C++ LIBS
│   ├── CAENVMEoslib.h ............................................. CAEN OS C++ LIBS
│   ├── CAENVMEtypes.h ............................................... CAEN VARIABLES
│   ├── DataReader.h ..................................... DECLARATION OF OBJECT DATAREADER
│   ├── IniFile.h ........................... DECLARATION OF OBJECT INIFILE FOR INI PARSER
│   ├── MsgSvc.h ................................... DECLARATION OF DAQ LOG MESSAGES
│   ├── utils.h ................ DECLARATION OF USEFUL VARIABLES AND COMM. FUNCTIONS
│   ├── v1190a.h ........................................ DECLARATION OF OBJECT V1190A
│   └── v1718.h .......................................... DECLARATION OF OBJECT V1718
├── lib ............................................................ CAEN LIBRARY
│   ├── install
│   └── x86
│       └── libCAENVME.so.2.41
├── obj ........................................... BINARY FILES CREATED BY COMPILER
│   └── ...
├── src ........................................................ LIST OF C++ SOURCE FILES
│   ├── daq.cxx ........................................................ MAIN FILE
│   ├── DataReader.cxx ............................ DEFINITION OF DATAREADER'S METHODS
│   ├── IniFile.cxx .................................. DEFINITION OF INIFILE'S METHODS
│   ├── MsgSvc.cxx ............................ DEFINITION OF LOG MESSAGING FUNCTIONS
│   ├── utils.cxx ..................... DEFINITION OF USEFUL COMMUNICATION FUNCTIONS
│   ├── v1190a.cxx .................................. DECLARATION OF V1190A'S METHODS
│   └── v1718.cxx .................................... DECLARATION OF V1718'S METHODS
├── makefile ............................................... COMPILER INSTRUCTIONS
└── README.md ........................................... REAMDE FILE FOR GITHUB
```

A.2 Usage of the DAQ

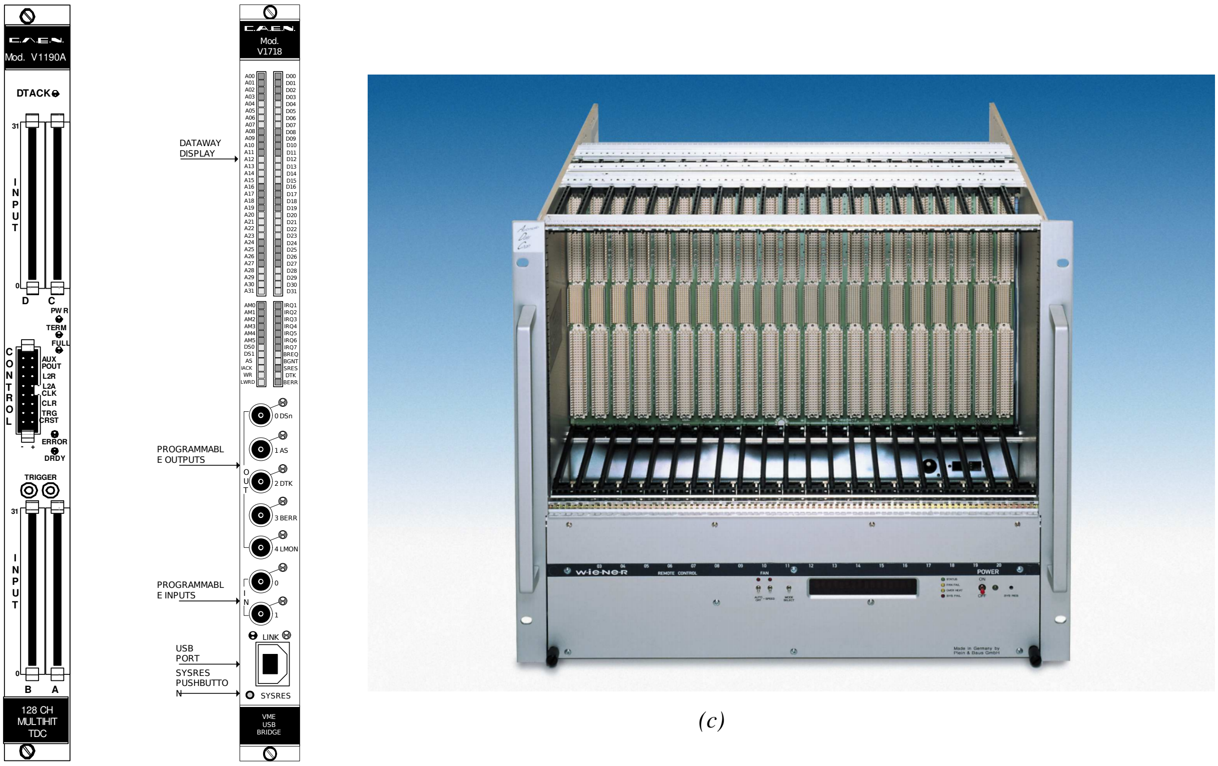
GIF++ DAQ, as used in GIF++, is not a standalone software. Indeed, the system being more complexe, the DAQ only is a sub-layer of the software architecture developped to control and monitor the RPCs that are placed into the bunker for performance study in an irradiated environment. The top layer of GIF++ is a Web Detector Control System (webDCS) application. The DAQ is only called by the webDCS when data needs to be acquired. The webDCS operates the DAQ through command line. To start the DAQ, the webDCS calls:

```
bin/daq /path/to/the/log/file/in/the/output/data/folder
```

where `/path/to/the/log/file/in/the/output/data/folder` is the only argument required. This log file is important for the webDCS as this file contains all the content of the communication of the webDCS and the different systems monitored by the webDCS. Its content is constantly displayed during data taking for the users to be able to follow the operations. The communication messages are normally sent to the webDCS log file via the functions declared in file `MsgSvc.h`, typically `MSG_INFO(string message)`.

A.3 Description of the readout setup

The CMS RPC setup at GIF++ counts 5 V1190A Time-to-Digital Converter (TDC) manufactured by CAEN [37]. V1190A are VME units accepting 128 independent Multi-Hit/Multi-Event TDC channels whose signals are treated by 4 100 ps high performance TDC chips developped by CERN / ECP-MIC Division. The communication between the computer and the TDCs to transfer data is done via a V1718 VME master module also manufactured by CAEN and operated from a USB port [38]. These VME modules are all hosted into a 6U VME 6021 powered crate manufactured by W-Ie-Ne-R than can accomodate up to 21 VME bus cards [39]. These 3 components of the DAQ setup are shown in Figure A.1.



(a) *(b)* *(c)*

Figure A.1: (A.1a) View of the front panel of a V1190A TDC module [37]. (A.1b) View of the front panel of a V1718 Bridge module [38]. (A.1c) View of the front panel of a 6U 6021 VME crate [39].

A.4 Data read-out

To efficiently perform a data readout algorithm, C++ objects to handle the VME modules (TDCs and VME bridge) have been created along with objects to store data and read the configuration file that comes as an input of the DAQ software.

A.4.1 V1190A TDCs

The DAQ used at GIF takes profit of the *Trigger Matching Mode* offered by V1190A modules. This setting is enabled through the method `v1190a::SetTrigMatching (int ntdcs)` where `ntdcs` is the total number of TDCs in the setup this setting needs to be enabled for (Source Code A.1). A trigger matching is performed in between a trigger time tag, a trigger signal sent into the TRIGGER input of the TDC visible on Figure A.1a, and the channel time measurements, signals recorded from the detectors under test in our case. Control over this data acquisition mode, explained through Figure A.2, is offered via 4 programmable parameters:

- **match window:** the matching between a trigger and a hit is done within a programmable time window. This is set via the method
  `void v1190a::SetTrigWindowWidth(Uint windowWidth,int ntdcs)`
- **window offset:** temporal distance between the trigger tag and the start of the trigger matching window. This is set via the method
  `void v1190a::SetTrigWindowWidth(Uint windowWidth,int ntdcs)`
- **extra search margin:** an extended time window is used to ensure that all matching hits are found. This is set via the method
  `void v1190a::SetTrigSearchMargin(Uint searchMargin,int ntdcs)`
- **reject margin:** older hits are automatically rejected to preven buffer overflows and to speed up the search time. This is set via the method
  `void v1190a::SetTrigRejectionMargin(Uint rejectMargin,int ntdcs)`

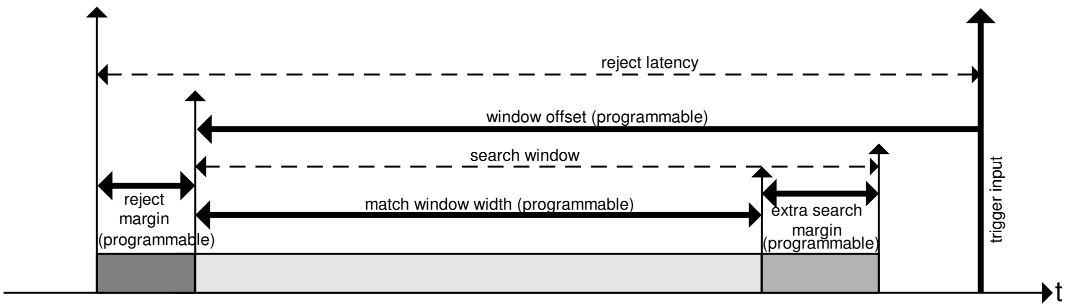


Figure A.2: Module V1190A Trigger Matching Mode timing diagram [37].

Each of these 4 parameters are given in number of clocks, 1 clock being 25 ns long. It is easy to understand at this level that there are 3 possible functionning settings:

- **1:** the match window is entirely contained after the trigger signal,
- **2:** the match window overlaps the trigger signal, or
- **3:** the match window is entirely contained before the trigger signal as displayed on Figure A.2.

In both the first and second cases, the sum of the window width and of the offset can be set to a maximum of 40 clocks, which corresponds to 1 μs. Evidently, the offset can be negative, allowing for a longer match window, with the constraint of having the window ending at most 1 μs after the trigger signal. In the third case, the maximum negative offset allowed is of 2048 clocks (12 bit) corresponding to 51.2 μs, the match window being strictly smaller than the offset. In the case of GIF++, the choice has been made to use this last setting by delaying the trigger signal. During the studies performed in GIF++, both the efficiency of the RPCs, probed using a muon beam, and the noise or gamma background rate are monitored. The extra search and reject margins are left unused.

To probe the efficiency of RPC detectors, the trigger time tag is provided by the coïncidence of scintillators when a bunch of muons passes through GIF++ area is used to trigger the data acquisition. For this measurement, it is useful to reduce the match window width only to contain the muon information. Indeed, the delay in between a trigger signal and the detection of the corresponding muon in the RPC being very contant (typically a few tens of ns due to jitter and cable length), the muon signals are very localised in time. Thus, due to a delay of approximalety 325 ns in between the muons and the trigger, the settings where chosen to have a window width of 24 clocks (600 ns) centered on the muon peak thanks to a negative offset of 29 clocks (725 ns).

On the otherhand, monitoring the rates don't require for the DAQ to look at a specific time window. It is important to integrate enough time to have a robust measurement of the rate as the number of hits per time unit. The triggerring signal is provided by a pulse generator at a frequency of 300 Hz to ensure that the data taking occurs in a random way, with respect to beam physics, to probe only the irradiation spectrum on the detectors. The match window is set to 400 clocks (10 μs) and the negative offset to 401 clocks as it needs to exceed the value of the match window.

```
class v1190a
{
 private :
    long              Handle;
    vector<Data32>    Address;
    CVDataWidth       DataWidth;
    CVAddressModifier AddressModifier;

 public:

    v1190a(long handle, IniFile *inifile, int ntdcs);
    ~v1190a();
    Data16 write_op_reg(Data32 address, int code, string error);
    Data16 read_op_reg(Data32 address, string error);
    void   Reset(int ntdcs);
    void   Clear(int ntdcs);
    void   TestWR(Data16 value,int ntdcs);
    void   CheckTDCStatus(int ntdcs);
    void   CheckCommunication(int ntdcs);
    void   SetTDCTestMode(Data16 mode,int ntdcs);
    void   SetTrigMatching(int ntdcs);
    void   SetTrigTimeSubstraction(Data16 mode,int ntdcs);
    void   SetTrigWindowWidth(Uint windowWidth,int ntdcs);
    void   SetTrigWindowOffset(Uint windowOffset,int ntdcs);
    void   SetTrigSearchMargin(Uint searchMargin,int ntdcs);
    void   SetTrigRejectionMargin(Uint rejectMargin,int ntdcs);
    void   GetTrigConfiguration(int ntdcs);
    void   SetTrigConfiguration(IniFile *inifile,int ntdcs);
    void   SetTDCDetectionMode(Data16 mode,int ntdcs);
    void   SetTDCResolution(Data16 lsb,int ntdcs);
    void   SetTDCDeadTime(Data16 time,int ntdcs);
    void   SetTDCHeadTrailer(Data16 mode,int ntdcs);
    void   SetTDCEventSize(Data16 size,int ntdcs);
    void   SwitchChannels(IniFile *inifile,int ntdcs);
    void   SetIRQ(Data32 level, Data32 count,int ntdcs);
    void   SetBlockTransferMode(Data16 mode,int ntdcs);
    void   Set(IniFile *inifile,int ntdcs);
    void   CheckStatus(CVErrorCodes status) const;
    int    ReadBlockD32(Uint tdc, const Data16 address,
               Data32 *data, const Uint words, bool ignore_berr);
    Uint   Read(RAWData *DataList,int ntdcs);
};
```

Source Code A.1: Description of C++ object `v1190a`*.*

The v1190a object, defined in the DAQ software as in Source Code A.1, offers the possility to concatenate all TDCs in the readout setup into a single object containing a list of hardware addresses (addresses to access the TDCs' buffer through the VME crate) and each constructor and method acts on the list of TDCs.

A.4.2 DataReader

Enabled thanks to `v1190a::SetBlockTransferMode(Data16 mode, int ntdcs)`, the data transfer is done via Block Transfer (BLT). Using BLT allows to tranfer a fixed number of events called a *block*. This is used together with an Almost Full Level (AFL) of the TDCs' output buffers, defined through `v1190a::SetIRQ(Data32 level, Data32 count,int ntdcs)`. This AFL gives the maximum amount of 32735 words (16 bits, corresponding to the depth of a TDC output buffer) that can writen in a buffer before an Interrupt Request (IRQ) is generated and seen by the VME Bridge, stopping the data acquisition to transfer the content of each TDC buffers before resuming. For each trigger, 6 words or more are written into the TDC buffer:

- **a global header** providing information of the event number since the beginning of the data acquisition,
- **a TDC header**,
- **the TDC data** (*if any*), 1 for each hit recorded during the event, providing the channel and the time stamp associated to the hit,
- **a TDC error** providing error flags,
- **a TDC trailer**,
- **a global trigger time tag** that provides the absolute trigger time relatively to the last reset, and
- **a global trailer** providing the total word count in the event.

As previously described in Section 4.3.3, CMS RPC FEEs provide us with $100\,\mathrm{ns}$ long LVDS output signals that are injected into the TDCs' input. Any avalanche signal that gives a signal above the FEEs threshold is thus recorded by the TDCs as a hit within the match window. Each hit is assigned to a specific TDC channel with a time stamp, with a precision of $100\,\mathrm{ps}$. The reference time, $t_0 = 0$, is provided by the beginning of the match window. Thus for each trigger, coming from a scintillator coïncidence or the pulse generator, a list of hits is stored into the TDCs' buffers and will then be transfered into a ROOT Tree.

When the BLT is used, it is easy to understand that the maximum number of words that have been set as ALF will not be a finite number of events or, at least, the number of events that would be recorded into the TDC buffers will not be a multiple of the block size. In the last BLT cycle to tranfer data, the number of events to transfer will most propably be lower than the block size. In that case, the TDC can add fillers at the end of the block but this option requires to send more data to the computer and is thus a little slower. Another solution is to finish the transfer after the last event by sending a bus error that states that the BLT reached the last event in the pile. This method has been chosen in GIF++.

Due to irradiation, an event in GIF++ can count up to 300 words per TDC. A limit of 4096 words (12 bits) has been set to generate IRQ which represent from 14 to almost 700 events depending on the average of hits collected per event. Then the block size has been set to 100 events with enabled bus errors. When an AFL is reached for one of the TDCs, the VME bridge stops the acquisition by sending a BUSY signal.

The data is then transfered one TDC at a time into a structure called `RAWData` (Source Code A.2).

```
struct RAWData{
    vector<int>              *EventList;
    vector<int>              *NHitsList;
    vector<int>              *QFlagList;
    vector<vector<int> >     *ChannelList;
    vector<vector<float> > *TimeStampList;
};
```

Source Code A.2: Description of data holding C++ structure `RAWData`.

In order to organize the data transfer and the data storage, an object called `DataReader` was created (Source Code A.3). On one hand, it has `v1718` and `v1190a` objects as private members for communication purposes, such as VME modules settings via the configuration file `*iniFile` or data read-out through `v1190a::Read()` and on the other hand, it contains the struture `RAWData` that allows to organise the data in vectors reproducing the tree structre of a ROOT file.

```
class DataReader
{
    private:
        bool     StopFlag;
        IniFile *iniFile;
        Data32   MaxTriggers;
        v1718   *VME;
        int      nTDCs;
        v1190a  *TDCs;
        RAWData  TDCData;

    public:
        DataReader();
        virtual ~DataReader();
        void     SetIniFile(string inifilename);
        void     SetMaxTriggers();
        Data32   GetMaxTriggers();
        void     SetVME();
        void     SetTDC();
        int      GetQFlag(Uint it);
        void     Init(string inifilename);
        void     FlushBuffer();
        void     Update();
        string   GetFileName();
        void     WriteRunRegistry(string filename);
        void     Run();
};
```

Source Code A.3: Description of C++ object `DataReader`.

Each event is tranferred from `TDCData` and saved into branches of a ROOT `TTree` as 3 integers that represent the event ID (`EventCount`), the number of hits read from the TDCs (`nHits`), and the quality flag that provides information for any problem in the data transfer (`qflag`), and 2 lists of *nHits* elements containing the fired TDC channels (`TDCCh`) and their respective time stamps (`TDCTS`), as presented in Source Code A.4. The ROOT file file is named using information contained into the configuration file, presented in section A.5.2. The needed information is extracted using method `DataReader::GetFileName()` and allow to build the output filename format `ScanXXXXXX_HVX_DAQ.root` where `ScanXXXXXX` is a 6 digit number representing the scan number into GIF++ database and `HVX` the HV step within the scan that can be more than a single digit. An example of ROOT data file is provided with Figure A.3.

```
RAWData TDCData;
TFile *outputFile = new TFile(outputFileName.c_str(),"recreate");
TTree *RAWDataTree = new TTree("RAWData","RAWData");

int            EventCount = -9;
int            nHits = -8;
int            qflag = -7;
vector<int>    TDCCh;
vector<float>  TDCTS;

RAWDataTree->Branch("EventNumber",&EventCount, "EventNumber/I");
RAWDataTree->Branch("number_of_hits",&nHits,"number_of_hits/I");
RAWDataTree->Branch("Quality_flag",&qflag,"Quality_flag/I");
RAWDataTree->Branch("TDC_channel",&TDCCh);
RAWDataTree->Branch("TDC_TimeStamp",&TDCTS);

//...
//Here read the TDC data using v1190a::Read() and place it into
//TDCData for as long as you didn't collect the requested amount
//of data.
//...

for(Uint i=0; i<TDCData.EventList->size(); i++){
    EventCount  = TDCData.EventList->at(i);
    nHits       = TDCData.NHitsList->at(i);
    qflag       = TDCData.QFlagList->at(i);
    TDCCh       = TDCData.ChannelList->at(i);
    TDCTS       = TDCData.TimeStampList->at(i);
    RAWDataTree->Fill();
}
```

Source Code A.4: Highlight of the data transfer and organisation within `DataReader::Run()` after the data has been collected into `TDCData`.

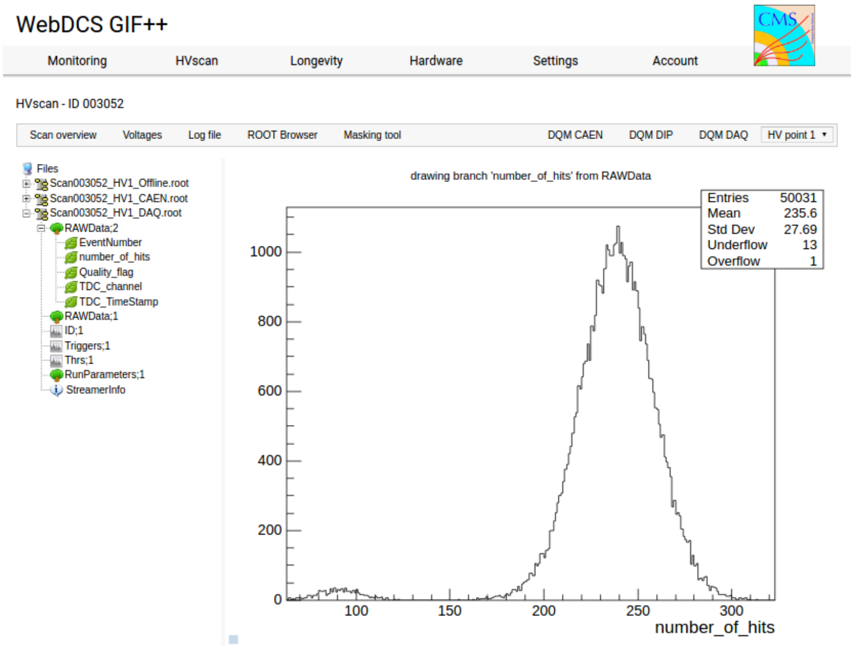


Figure A.3: Structure of the ROOT output file generated by the DAQ. The 5 branches (`EventNumber`, `number_of_hits`, `Quality_flag`, `TDC_channel` and `TDC_TimeStamp`) are visible on the left panel of the ROOT browser. On the right panel is visible the histogram corresponding to the variable `nHits`. In this specific example, there were approximately 50k events recorded to measure the gamma irradiation rate on the detectors. Each event is stored as a single entry in the `TTree`.

A.4.3 Data quality flag

Among the parameters that are recorded for each event, the quality flag, defined in Source Code A.5, is determined on the fly by checking the data recorded by every single TDC. From method `v1190a::Read()`, it can be understood that the content of each TDC buffer is readout one TDC at a time. Entries are created in the data list for the first TDC and then, when the second buffer is readout, events corresponding to entries that have already been created to store data for the previous TDC are added to the existing list element. On the contrary, when an event entry has not been yet created in the data list, a new entry is created.

```
typedef enum _QualityFlag {
    GOOD      = 1,
    CORRUPTED = 0
} QualityFlag;
```

Source Code A.5: Definition of the quality flag **`enum`**.

It is possible that each TDC buffer contains a different number of events. In cases where the first element in the buffer list is an event for corresponds to a new entry, the difference in between the intry from the buffer and the last entry in the data list is recorded and checked. If it is greater than 1, what should never be the case, the quality flag is set to `CORRUPTED` for this TDC and an empty entry is created in the place of the missing ones. Missing entries are believe to be the result of a bad hold on the TDC buffers at the moment of the readout. Indeed, the software hold is effective only on 1 TDC at a time and no solution as been found yet to completely block the writting in the buffers when an IRQ is received.

At the end of each BLT cycle, the ID of the last entry stored for each TDC buffer is not recorded. When starting the next cycle, if the first entry in the pile corresponds to an event already existing in the list, the readout will start from this list element and will not be able to check the difference in between this entry's ID and the one of the last entry that was recorded for this TDC buffer in the previous cycle. In the case events were missing, the flag stays at its initial value of 0, which is similar to `CORRUPTED` and it is assumed that then this TDC will not contribute to `number_of_hits`, `TDC_channel` or `TDC_TimeStamp`.

Finally, since there will be 1 `RAWData` entry per TDC for each event (meaning `nTDCs` entries, referring to `DataReader` private attribute), the individual flags of each TDC will be added together. The final format is an integer composed `nTDCs` digits where each digit is the flag of a specific TDC. This is constructed using powers of 10 like follows:

TDC 0: `QFlag` $= 10^0 \times$ `_QualityFlag`

TDC 1: `QFlag` $= 10^1 \times$ `_QualityFlag`

...

TDC N: `QFlag` $= 10^N \times$ `_QualityFlag`

and the final flag to be with N digits:

`QFlag = n....3210`

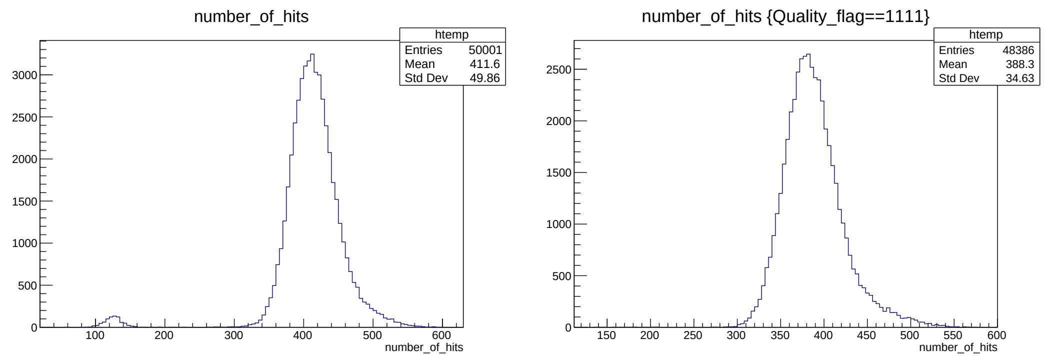
each digit being 1 or 0. Bellow is given an example with a 4 TDCs setup.

If all TDCs were good : `QFlag = 1111`,

but if TDC 2 was corrupted : `QFlag = 1011`.

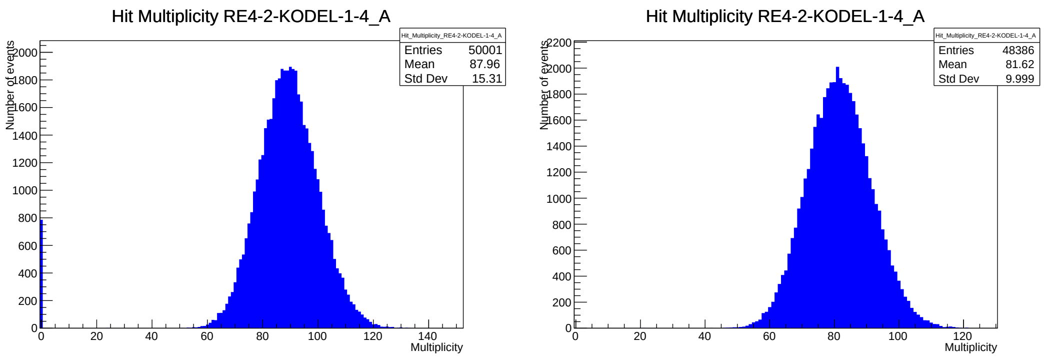
When data taking is over and the data contained in the dynamical `RAWData` structure is transfered to the ROOT file, all the 0s are changed into 2s by calling the method `DataReader::GetQFlag()`. This will help translating the flag without knowing the number of TDCs beforehand. Indeed, a flag `111` could be due to a 3 TDC setup with 3 good individual TDC flags or to a more than 3 TDC setup with TDCs those ID is greater than 2 being `CORRUPTED`, thus giving a 0.

The quality flag has been introduced quite late, in October 2017 only, to the list of GIF++ DAQ parameters to be recorded into the output ROOT file. Before this addition, the missing data, corrupting the quality for the offline analysis, was contributing to artificially fill data with lower multiplicity. Looking at `TBranch number_of_hits` provides an information about the data of the full GIF++ setup. When a TDC is not able to transfer data for a specific event, the effect is a reduction of the total number of hits recorded in the full setup, this is what can be seen from Figure A.4. After offline reconstruction detector by detector, the effect of missing events can be seen in the artificially filled bin at multiplicity 0 shown in Figure A.5. Nontheless, for data wih high irradiation levels, as it is he case for Figure A.5a, discarding the fake multiplicity 0 data can be done easily during the offline analysis. At lower radiation, the missing events contribution becomes more problematic as the multiplicity distribution overlaps the multiplicity 0 and that in the same time the proportion of missing events decreases. Attempts to fit the distribution with a Poisson or skew distribution function were not conclusive and this very problem has been at the origin of the quality flag that allows to give a non ambiguous information about each event quality.



(a) *(b)*

Figure A.4: The effect of the quality flag is explained by presenting the content of `TBranch number_of_hits` *of a data file without* `Quality_flag` *in Figure A.4a and the content of the same* `TBranch` *for data corresponding to a* `Quality_flag` *where all TDCs were labelled as* `GOOD` *in Figure A.4b taken with similar conditions. It can be noted that the number of entries in Figure A.4b is slightly lower then in Figure A.4a due to the excluded events.*



(a) *(b)*

Figure A.5: Using the same data as previously showed in Figure A.4, the effect of the quality flag is explained by presenting the reconstructed hit multiplicity of a data file without `Quality_flag` *in Figure A.5a and the reconstructed content of the same RPC partition for data corresponding to a* `Quality_flag` *where all TDCs were labelled as* `GOOD` *in Figure A.5b taken with similar conditions. The artificial high content of bin 0 is completely suppressed.*

A.5 Communications

To ensure data readout and dialog in between the machine and the TDCs or in between the webDCS and the DAQ, different communication solutions were used. First of all, it is important to have a module to allow the comminication in between the TDCs and the computer from which the DAQ operates. When this communication is effective, shifters using the webDCS to control data taking can thus send instructions to the DAQ.

A.5.1 V1718 USB Bridge

In the previous section, the data transfer as been discussed. The importance of the `v1718` object (Source Code A.6), used as private member of `DataReader`, was not explicited. VME master modules are used for communication purposes as they host the USB port that connects the powered crate buffer to the computer were the DAQ is installed. From the source code point of view, this object is used to control the communication status, by reading the returned error codes with `v1718::CheckStatus()`, or to check for IRQs coming from the TDCs through `v1718::CheckIRQ()`. Finally, to ensure that triggers are blocked at the hardware level, a NIM pulse is sent out of one of the 5 programmable outputs (`v1718::SendBUSY()`) to the VETO of the coïndidence module where the trigger signals originate from. As long as this signal is ON, no trigger can reach the TDCs anymore.

```
class v1718{
    private:
        int                 Handle;
        Data32              Data;           // Data
        CVIRQLevels         Level;          // Interrupt level
        CVAddressModifier AM;               // Addressing Mode
        CVDataWidth         DataSize;       // Data Format
        Data32              BaseAddress;    // Base Address

    public:
        v1718(IniFile *inifile);
        ~v1718();
        long                GetHandle(void) const;
        int                 SetData(Data16 data);
        Data16              GetData(void);
        int                 SetLevel(CVIRQLevels level);
        CVIRQLevels         GetLevel(void);
        int                 SetAM(CVAddressModifier am);
        CVAddressModifier GetAM(void);
        int                 SetDatasize(CVDataWidth datasize);
        CVDataWidth         GetDataSize(void);
        int                 SetBaseAddress(Data16 baseaddress);
        Data16              GetBaseAddress(void);
        void                CheckStatus(CVErrorCodes status) const;
        bool                CheckIRQ();
        void                SetPulsers();
        void                SendBUSY(BusyLevel level);
};
```

Source Code A.6: Description of C++ object `v1718`.

A.5.2 Configuration file

The DAQ software takes as input a configuration file written using INI standard [40]. This file is partly filled with the information provided by the shifters when starting data acquisition using the webDCS, as shown by Figure A.6. This information is written in section `[General]` and will later be stored in the ROOT file that contains the DAQ data as can be seen from Figure A.3. Indeed, another `TTree` called `RunParameters` as well as the 2 histograms `ID`, containing the scan number, start and stop time stamps, and `Triggers`, containing the number of triggers requested by the shifter, are available in the data files. Moreover, `ScanID` and `HV` are then used to construct the file name thanks to the method `DataReader::GetFileName()`.

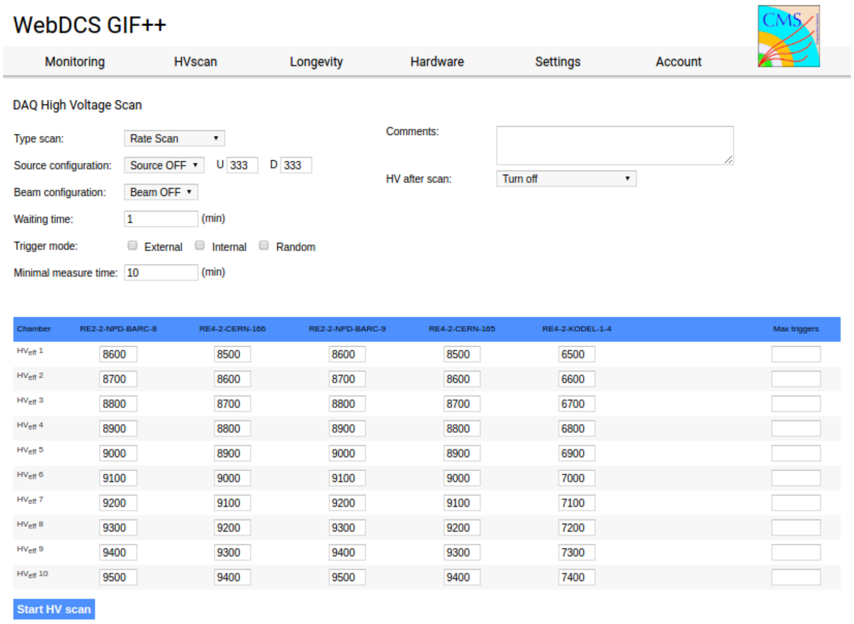


Figure A.6: WebDCS DAQ scan page. On this page, shifters need to choose the type of scan (Rate, Efficiency or Noise Reference scan), the gamma source configuration at the moment of data taking, the beam configuration, and the trigger mode. These information will be stored in the DAQ ROOT output. Are also given the minimal measurement time and waiting time after ramping up of the detectors is over before starting the data acquisition. Then, the list of HV points to scan and the number of triggers for each run of the scan are given in the table underneath.

The rest of the information is written beforehand in the configuration file template, as explicited in Source Code A.7, and contains the hardware addresses to the differents VME modules in the setup as well as settings for the TDCs. As the TDC settings available in the configuration file are not supposed to be modified, an improvement would be to remove them from the configuration file and to hardcode them inside of the DAQ code itself or to place them into a different INI file that would host only the TDC settings to lower the probability for a bad manipulation of the configuration file that can be modified from one of webDCS' menus.

1366

```
[General]
Tdcs=4
ScanID=$scanid
HV=$HV
RunType=$runtype
MaxTriggers=$maxtriggers
Beam=$beam
[VMEInterface]
Type=V1718
BaseAddress=0xFF0000
Name=VmeInterface
[TDC0]
Type=V1190A
BaseAddress=0x00000000
Name=Tdc0
StatusA00-15=1
StatusA16-31=1
StatusB00-15=1
StatusB16-31=1
StatusC00-15=1
StatusC16-31=1
StatusD00-15=1
StatusD16-31=1
[TDC1]
Type=V1190A
BaseAddress=0x11110000
Name=Tdc1
StatusA00-15=1
StatusA16-31=1
StatusB00-15=1
StatusB16-31=1
StatusC00-15=1
StatusC16-31=1
StatusD00-15=1
StatusD16-31=1
```

1367

```
[TDC2]
Type=V1190A
BaseAddress=0x22220000
Name=Tdc2
StatusA00-15=1
StatusA16-31=1
StatusB00-15=1
StatusB16-31=1
StatusC00-15=1
StatusC16-31=1
StatusD00-15=1
StatusD16-31=1
[TDC3]
Type=V1190A
BaseAddress=0x44440000
Name=Tdc3
StatusA00-15=1
StatusA16-31=1
StatusB00-15=1
StatusB16-31=1
StatusC00-15=1
StatusC16-31=1
StatusD00-15=1
StatusD16-31=1
[TDCSettings]
TriggerExtraSearchMargin=0
TriggerRejectMargin=0
TriggerTimeSubstraction=0b1
TdcDetectionMode=0b01
TdcResolution=0b10
TdcDeadTime=0b00
TdcHeadTrailer=0b1
TdcEventSize=0b1001
TdcTestMode=0b0
BLTMode=1
```

Source Code A.7: INI configuration file template for 4 TDCs. In section **`[General]`**, *the number of TDCs is explicited and information about the ongoing run is given. Then, there are sections for each and every VME modules. There buffer addresses are given and for the TDCs, the list of channels to enable is given. Finally, in section* **`[TDCSettings]`**, *a part of the TDC settings are given.*

In order to retreive the information of the configuration file, the object `IniFile` has been developped to provide an INI parser, presented in Source Code A.8. It contains private methods returning a boolean to check the type of line written in the file, whether a comment, a group header or a key line (`IniFile::CheckIfComment()`, `IniFile::CheckIfGroup()` and `IniFile::CheckIfToken()`). The key may sometimes be referred to as *token* in the source code. Moreover, the private element `FileData` is a map of **`const`** `string` to `string` that allows to store the data contained inside the configuration file via the public method `IniFile::GetFileData()` following the formatting (see method `IniFile::Read()`):

```
string group, token, value;
// Get the field values for the 3 strings.
// Then concatenate group and token together as a single string
// with a dot separation.
token = group + "." + token;
FileData[token] = value;
```

More methods have been written to translate the different keys into the right variable format when used by the DAQ. For example, to get a **`float`** value out of the configuration file data, knowing the group and the key needed, the method `IniFile::floatType()` can be used. It takes 3 arguments being the group name and key name (both `string`), and a default **`float`** value used as exception in the case the expected combination of group and key cannot be found in the configuration file. This default value is then used and the DAQ continues on working after sending an alert in the log file for further debugging.

```
typedef map< const string, string > IniFileData;

class IniFile{
    private:
        bool        CheckIfComment(string line);
        bool        CheckIfGroup(string line,string& group);
        bool        CheckIfToken(string line,string& key,string& value);
        string      FileName;
        IniFileData FileData;
        int         Error;

    public:
        IniFile();
        IniFile(string filename);
        virtual     ~IniFile();

        // Basic file operations
        void        SetFileName(string filename);
        int         Read();
        int         Write();
        IniFileData GetFileData();

        // Data readout methods
        Data32      addressType (string groupname, string keyname, Data32 defaultvalue);
        long        intType     (string groupname, string keyname, long defaultvalue);
        long long   longType    (string groupname, string keyname, long long defaultvalue );
        string      stringType  (string groupname, string keyname, string defaultvalue );
        float       floatType   (string groupname, string keyname, float defaultvalue );

        // Error methods
        string      GetErrorMsg();
};
```

Source Code A.8: Description of C++ object `IniFile` used as a parser for INI file format.

A.5.3 WebDCS/DAQ intercommunication

When shifters send instructions to the DAQ via the configuration file, it is the webDCS itself that gives the start command to the DAQ and then the 2 softwares use inter-process communication through file to synchronise themselves. This communication file is represented by the variable `const string __runstatuspath`.

On one side, the webDCS sends commands or status that are readout by the DAQ:

- `INIT`, status sent when launching a scan and read via function `CtrlRunStatus(...)`,
- `START`, command to start data taking and read via function `CheckSTART()`,
- `STOP`, command to stop data taking at the end of the scan and read via function `CheckSTOP()`, and
- `KILL`, command to kill data taking sent by user and read via function `CheckKILL()`

and on the other, the DAQ sends status that are controled by the webDCS:

- `DAQ_RDY`, sent with `SendDAQReady()` to signify that the DAQ is ready to receive commands from the webDCS,
- `RUNNING`, sent with `SendDAQRunning()` to signify that the DAQ is taking data,
- `DAQ_ERR`, sent with `SendDAQError()` to signify that the DAQ didn't receive the expected command from the webDCS or that the launch command didn't have the right number of arguments,
- `RD_ERR`, sent when the DAQ wasn't able to read the communication file, and
- `WR_ERR`, sent when the DAQ wasn't able to write into the communication file.

A.5.4 Example of inter-process communication cycle

Under normal conditions, the webDCS and the DAQ processes exchange commands and status via the file hosted at the address `__runstatuspath`, as explained in subsection A.5.3. An example of cycle is given in Table A.1. In this example, the steps 3 to 5 are repeated as long as the webDCS tells the DAQ to take data. A data taking cycle is the equivalent as what is called a *Scan* in GIF++ jargon, referring to a set a runs with several HV steps. Each repetition of steps 3 to 5 is then equivalent to a single *Run*.

At any moment during the data taking, for any reason, the shifter can decide that the data taking needs to be stopped before it reached the end of the scheduled cycle. Thus at any moment on the cycle, the content of the inter-process communication file will be changed to `KILL` and the DAQ will shut down right away. The DAQ checks for `KILL` signals every 5s after the TDCs configuration is over. So far, the function `CheckKILL()` has been used only inside of the data taking loop of method `DataReader::Run()` and thus, if the shifter decides to KILL the data taking during the TDC configuration phase or the HV ramping in between 2 HV steps, the DAQ will not be stopped smoothly and a *force kill* command will be sent to stop the DAQ process that is still awake on the computer. Improvements can be brought on this part of the software to make sure that the DAQ can safely shutdown at any moment.

A.6 Software export

In section A.2 was discussed the fact that the DAQ as written in its last version is not a standalone software. It is possible to make it a standalone program that could be adapted to any VME setup using V1190A and V1718 modules by creating a GUI for the software or by printing the log messages that are normally printed in the webDCS through the log file, directly into the terminal. This method was used by the DAQ up to version 3.0 moment where the webDCS was completed. Also, it is possible to check branches of DAQ v2.X to have example of communication through a terminal.

DAQ v2.X is nontheless limited in it's possibilities and requires a lot of offline manual interventions from the users. Indeed, there is no communication of the software with the detectors' power supply system that would allow for a user a predefine a list of voltages to operate the detectors at

step	actions of webDCS	status of DAQ	`__runstatuspath`
1	launch DAQ ramp voltages ramping over wait for currents stabilization	readout of IniFfile configuration of TDCs	`INIT`
2		configuration done send DAQ ready wait for `START` signal	`DAQ_RDY`
3	waiting time over send `START`		`START`
4	wait for run to end monitor DAQ run status	data taking ongoing check for `KILL` signal	`RUNNING`
5		run over send `DAQ_RDY` wait for next DCS signal	`DAQ_RDY`
6	ramp voltages ramping over wait for currents stabilization		`DAQ_RDY`
3	waiting time over send `START`		`START`
4	wait for run to end monitor DAQ run status	update IniFile information data taking ongoing check for `KILL` signal	`RUNNING`
5		run over send `DAQ_RDY` wait for next DCS signal	`DAQ_RDY`
7	send command `STOP`	DAQ shuts down	`STOP`

Table A.1: Inter-process communication cycles in between the webDCS and the DAQ through file string signals.

and loop over to take data without any further manual intervention. In v2.X, the data is taken for a single detector setting and at the end of each run, the softwares asks the user if he intends on taking more runs. If so, the software invites the user to set the operating voltages accordingly to what is necessary and to manual update the configuration file in consequence. This working mode can be a very first approach before an evolution and has been successfuly used by colleagues from different collaborations.

For a more robust operation, it is recommanded to develop a GUI or a web application to interface the DAQ. Moreover, to limit the amount of manual interventions, and thus the probability to make mistakes, it is also recommanded to add an extra feature into the DAQ by installing the HV Wrapper library provided by CAEN of which an example of use in a similar DAQ software developped by a master student of UGent, and called TinyDAQ, is provided on UGent's github. Then, this HV Wrapper will help you communicating with and give instructions to a CAEN HV powered crate and can be added into the DAQ at the same level where the communication with the user was made in DAQ v2.X. In case you are using another kind of power system for your detectors, it is stringly adviced to use HV modules or crates that can be remotely controled via a using C++ libraries.

B

Details on the offline analysis package

The data collected in GIF++ thanks to the DAQ described in Appendix A is difficult to interpret by a human user that doesn't have a clear idea of the raw data architecture of the ROOT data files. In order to render the data human readable, a C++ offline analysis tool was designed to provide users with detector by detector histograms that give a clear overview of the parameters monitored during the data acquisition [41]. In this appendix, details about this software in the context of GIF++, as of how the software was written and how it functions will be given.

B.1 GIF++ Offline Analysis file tree

GIF++ Offline Analysis source code is fully available on github at `https://github.com/afagot/GIF_OfflineAnalysis`. The software requires ROOT as non-optionnal dependency as it takes ROOT files in input and write an output ROOT file containing histograms. To compile the GIF++ Offline Analysis project is compiled with cmake. To compile, first a `build/` directory must be created to compile from there:

```
mkdir build
cd build
cmake ..
make
make install
```

To clean the directory and create a new build directory, the bash script `cleandir.sh` can be used:

```
./cleandir.sh
```

The source code tree is provided below along with comments to give an overview of the files' content. The different objects created for this project (`Infrastructure`, `Trolley`, `RPC`, `Mapping`, `RPCHit`, `RPCCluster` and `Inifile`) will be described in details in the following sections.

```
GIF_OfflineAnalysis
├── bin
│   └── offlineanalysis ........................ EXECUTABLE
├── build ...................................... CMAKE COMPILATION DIRECTORY
│   └── ...
├── include .................................... LIST OF C++ HEADER FILES
│   ├── Cluster.h .............................. DECLARATION OF OBJECT RPCCLUSTER
│   ├── Current.h .............................. DECLARATION OF GETCURRENT ANALYSIS MACRO
│   ├── GIFTrolley.h ........................... DECLARATION OF OBJECT TROLLEY
│   ├── Infrastructure.h ....................... DECLARATION OF OBJECT INFRASTRUCTURE
│   ├── IniFile.h .............................. DECLARATION OF OBJECT INIFILE FOR INI PARSER
│   ├── Mapping.h .............................. DECLARATION OF OBJECT MAPPING
│   ├── MsgSvc.h ............................... DECLARATION OF OFFLINE LOG MESSAGES
│   ├── OfflineAnalysis.h ...................... DECLARATION OF DATA ANALYSIS MACRO
│   ├── RPCDetector.h .......................... DECLARATION OF OBJECT RPC
│   ├── RPCHit.h ............................... DECLARARION OF OBJECT RPCHIT
│   ├── types.h ................................ DEFINITION OF USEFUL VARIABLE TYPES
│   └── utils.h ................................ DECLARATION OF USEFUL FUNCTIONS
├── obj ........................................ BINARY FILES CREATED BY COMPILER
│   └── ...
├── src ........................................ LIST OF C++ SOURCE FILES
│   ├── Cluster.cc ............................. DEFINITION OF OBJECT RPCCLUSTER
│   ├── Current.cc ............................. DEFINITION OF GETCURRENT ANALYSIS MACRO
│   ├── GIFTrolley.cc .......................... DEFINITION OF OBJECT TROLLEY
│   ├── Infrastructure.cc ...................... DEFINITION OF OBJECT INFRASTRUCTURE
│   ├── IniFile.cc ............................. DEFINITION OF OBJECT INIFILE FOR INI PARSER
│   ├── main.cc ................................ MAIN FILE
│   ├── Mapping.cc ............................. DEFINITION OF OBJECT MAPPING
│   ├── MsgSvc.cc .............................. DEFINITION OF OFFLINE LOG MESSAGES
│   ├── OfflineAnalysis.cc ..................... DEFINITION OF DATA ANALYSIS MACRO
│   ├── RPCDetector.cc ......................... DEFINITION OF OBJECT RPC
│   ├── RPCHit.cc .............................. DEFINITION OF OBJECT RPCHIT
│   └── utils.cc ............................... DEFINITION OF USEFUL FUNCTIONS
├── cleandir.sh ................................ BASH SCRIPT TO CLEAN BUILD DIRECTORY
├── CMakeLists.txt ............................. SET OF INSTRUCTIONS FOR CMAKE
├── config.h.in ................................ DEFINITION OF VERSION NUMBER
└── README.md .................................. REAMDE FILE FOR GITHUB
```

B.2 Usage of the Offline Analysis

In order to use the Offline Analysis tool, it is necessary to know the Scan number and the HV Step of the run that needs to be analysed. This information needs to be written in the following format:

```
Scan00XXXX_HVY
```

where `XXXX` is the scan ID and `Y` is the high voltage step (in case of a high voltage scan, data will be taken for several HV steps). This format corresponds to the base name of data files in the database of the GIF++ webDCS. Usually, the offline analysis tool is automatically called by the webDCS at the end of data taking or by a user from the webDCS panel if an update of the tool was brought. Nontheless, an expert can locally launch the analysis for tests on the GIF++ computer, or a user can get the code on is local machine from github and download data from the webDCS for is own analysis. To launch the code, the following command can be used from the `GIF_OfflineAnalysis` folder:

```
bin/offlineanalysis /path/to/Scan00XXXX_HVY
```

where, `/path/to/Scan00XXXX_HVY` refers to the local data files. Then, the offline tool will by itself take care of finding all available ROOT data files present in the folder, as listed bellow:

- `Scan00XXXX_HVY_DAQ.root` containing the TDC data as described in Appendix **??** (events, hit and timestamp lists), and
- `Scan00XXXX_HVY_CAEN.root` containing the CAEN mainframe data recorded by the monitoring tool webDCS during data taking (HVs and currents of every HV channels). This file is created independently of the DAQ.

B.2.1 Output of the offline tool

B.2.1.1 ROOT file

The analysis gives in output ROOT datafiles that are saved into the data folder and called using the naming convention `Scan00XXXX_HVY_Offline.root`. Inside those, a list of `TH1` histograms can be found. Its size will vary as a function of the number of detectors in the setup as each set of histograms is produced detector by detector. For each partition of each chamber, can be found:

- `Time_Profile_Tt_Sc_p` shows the time profile of all recorded events (number of events per time bin),
- `Hit_Profile_Tt_Sc_p` shows the hit profile of all recorded events (number of events per channel),
- `Hit_Multiplicity_Tt_Sc_p` shows the hit multiplicity (number of hits per event) of all recorded events (number of occurences per multiplicity bin),
- `Strip_Mean_Noise_Tt_Sc_p` shows noise/gamma rate per unit area for each strip in a selected time range. After filters are applied on `Time_Profile_Tt_Sc_p`, the filtered version of `Hit_Profile_Tt_Sc_p` is normalised to the total integrated time and active detection area of a single channel,
- `Strip_Activity_Tt_Sc_p` shows noise/gamma activity for each strip (normalised version of previous histogram - strip activity = strip rate / average partition rate),
- `Strip_Homogeneity_Tt_Sc_p` shows the *homogeneity* of a given partition (homogeneity = exp(-strip rates standard deviation(strip rates in partition/average partition rate))),
- `mask_Strip_Mean_Noise_Tt_Sc_p` shows noise/gamma rate per unit area for each masked strip in a selected time range. Offline, the user can control the noise/gamma rate and decide to mask the strips that are judged to be noisy or dead. This is done via the *Masking Tool* provided by the webDCS,

- mask_Strip_Activity_Tt_Sc_p shows noise/gamma activity per unit area for each masked strip with repect to the average rate of active strips,
- NoiseCSize_H_Tt_Sc_p shows noise/gamma cluster size, a cluster being constructed out of adjacent strips giving a signal at the *same time* (hits within a time window of $25\,\mathrm{ns}$),
- NoiseCMult_H_Tt_Sc_p shows noise/gamma cluster multiplicity (number of reconstructed clusters per event),
- Chip_Mean_Noise_Tt_Sc_p shows the same information than Strip_Mean_Noise_Tt_Scp using a different binning (1 chip corresponds to 8 strips),
- Chip_Activity_Tt_Sc_p shows the same information than Strip_Activity_Tt_Scp using chip binning,
- Chip_Homogeneity_Tt_Sc_p shows the homogeneity of a given partition using chip binning,
- Beam_Profile_Tt_Sc_p shows the estimated beam profile when taking efficiency scan. This is obtained by filtering Time_Profile_Tt_Sc_p to only consider the muon peak where the noise/gamma background has been subtracted. The resulting hit profile corresponds to the beam profile on the detector channels,
- L0_Efficiency_Tt_Sc_p shows the level 0 efficiency that was estimated **without** muon tracking,
- MuonCSize_H_Tt_Sc_p shows the level 0 muon cluster size that was estimated **without** muon tracking, and
- MuonCMult_H_Tt_Sc_p shows the level 0 muon cluster multiplicity that was estimated **without** muon tracking.

In the histogram labels, t stands for the trolley number (1 or 3), c for the chamber slot label in trolley t and p for the partition label (A, B, C or D depending on the chamber layout) as explained in Chapter 5.3.

In the context of GIF++, an extra script called by the webDCS is called to extract the histograms from the ROOT files. The histograms are then stored in PNG and PDF formats into the corresponding folder (a single folder per HV step, so per ROOT file). the goal is to then display the histograms on the Data Quality Monitoring (DQM) page of the webDCS in order for the users to control the quality of the data taking at the end of data taking. An example of histogram organisation is given bellow:

```
Scan001000
├── Scan001000_HV1_DAQ.root
├── Scan001000_HV1_CAEN.root
├── Scan001000_HV1_Offline.root
├── HV1
│   ├── DAQ
│   │   ├── Beam_Profile_T1_S1_A.png
│   │   ├── Beam_Profile_T1_S1_A.pdf
│   │   ├── Beam_Profile_T1_S1_B.png
│   │   ├── Beam_Profile_T1_S1_B.pdf
│   │   └── ...
│   └── CAEN
│       ├── HVapp_Example_RPC1.pdf
│       ├── HVapp_Example_RPC1.png
│       ├── HVapp_Example_RPC1.pdf
│       ├── HVapp_Example_RPC1.png
│       └── ...
├── Scan001000_HV2_DAQ.root
├── Scan001000_HV2_CAEN.root
├── Scan001000_HV2_Offline.root
├── HV2
│   ├── DAQ
│   │   └── ...
│   └── CAEN
│       └── ...
├── Scan001000_HV3_DAQ.root
├── Scan001000_HV3_CAEN.root
├── Scan001000_HV3_Offline.root
├── HV3
│   └── ...
└── ...
```

Here can put some screens from the webDCS to show the DQM and the plots available to users.

B.2.1.2 CSV files

Moreover, up to 3 CSV files can be created depending on which ones of the 3 input files were in the data folder:

- `Offline-Corrupted.csv` , is used to keep track of the amount of data that was corrupted and removed from old data format files that don't contain any data quality flag.

- `Offline-Current.csv` , contains the summary of the currents and voltages applied on each RPC HV channel.

- `Offline-L0-EffCl.csv` , is used to write the efficiencies, cluster size and cluster multiplicity of efficiency runs. Note that `L0` refers here to *Level 0* and means that the results of effiency and clusterization are a first approximation calculated without performing any muon tracking in between the different detectors. This offline tool provides the user with a preliminar calculation of the efficiency and of the muon event parameters. Another analysis software especially dedicated to muon tracking is called on selected data to retrieve the results of efficiency and muon clusterization using a tracking algorithm to discriminate noise or gamma from muons as muons are the only particles that pass through the full setup, leaving hits than can be used to reconstruct their tracks.

- `Offline-Rate.csv` , is used to write the noise or gamma rates measured in the detector readout partitions.

Note that these 4 CSV files are created along with their *headers* (`Offline-[...]-Header.csv` containing the names of each data columns) and are automatically merged together when the offline analysis tool is called from the webDCS, contrary to the case where the tool is runned locally from the terminal as the merging bash script is then not called. Thus, the resulting files, used to make official plots, are:

- `Corrupted.csv` ,
- `Current.csv` ,
- `L0-EffCl.csv` .
- `Rate.csv` .

B.3 Analysis inputs and information handling

The usage of the Offline Analysis tool as well as its output have been presented in the previous section. It is now important to dig further and start looking at the source code and the inputs necessary for the tool to work. Indeed, other than the raw ROOT data files that are analysed, more information needs to be imported inside of the program to perform the analysis such as the description of the setup inside of GIF++ at the time of data taking (number of trolleys, of RPCs, dimensions of the detectors, etc...) or the mapping that links the TDC channels to the coresponding RPC channels in order to translate the TDC information into human readable data. 2 files are used to transmit all this information:

- `Dimensions.ini`, that provides the necessary setup and RPC information, and
- `ChannelsMapping.csv`, that gives the link between the TDC and RPC channels as well as the *mask* for each channel (masked or not?).

B.3.1 Dimensions file and IniFile parser

This input file, present in every data folder, allows the analysis tool to know of the number of active trolleys, the number of active RPCs in those trolleys, and the details about each RPCs such as the number of RPC gaps, the number of pseudo-rapidity partitions (for CMS-like prototypes), the number of strips per partion or the dimensions. To do so, there are 3 types of groups in the INI file architecture. A first general group, appearing only once at the head of the document, gives information about the number of active trolleys as well as their IDs, as presented in Source Code B.1. For each active trolley, a group similar to Source Code B.2 can be found containing information about the number of active detectors in the trolley and their IDs. Each trolley group as a `Tt` name format, where `t` is the trolley ID. Finally, for each detector stored in slots of an active trolley, there is a group providing information about their names and dimensions, as shown in Source Code B.3. Each slot group as a `TtSs` name format, where `s` is the slot ID of trolley `t` where the active RPC is hosted.

```
[General]
nTrolleys=2
TrolleysID=13
```

Source Code B.1: Example of `[General]` *group as might be found in* `Dimensions.ini`*. In GIF++, only 2 trolleys are available to hold RPCs and place them inside of the bunker for irradiation. The IDs of the trolleys are written in a signle string as "13" and then read character by character by the program.*

```
[T1]
nSlots=4
SlotsID=1234
```

Source Code B.2: Example of trolley group as might be found in `Dimensions.ini`*. In this example, the file tells that there are 4 detectors placed in the holding slots of the trolley* `T1` *and that their IDs, written as a single string variable, are 1, 2, 3 and 4.*

```
[T1S1]
Name=RE2-2-NPD-BARC-8
Partitions=3
Gaps=3
Gap1=BOT
Gap2=TN
Gap3=TW
AreaGap1=11694.25
AreaGap2=6432
AreaGap3=4582.82
Strips=32
ActiveArea-A=157.8
ActiveArea-B=121.69
ActiveArea-C=93.03
```

Source Code B.3: Example of slot group as might be found in `Dimensions.ini`*. In this example, the file provides information about a detector named* `RE2-2-NPD-BARC-8`*, having 3 pseudo-rapidity readout partitions and stored in slot* `S1` *of trolley* `T1`*. This is a CMS RE2-2 type of detector. This information will then be used for example to compute the rate per unit area calculation.*

This information is readout and stored in a C++ object called `IniFile`, that parses the information in the INI input file and stores it into a local buffer for later use. This INI parser is the exact same one that was previously developped for the GIF++ DAQ and described in Appendix A.5.2.

B.3.2 TDC to RPC link file and Mapping

The same way the INI dimension file information is stored using `map`, the channel mapping and mask information is stored and accessed through `map`. First of all, the mapping CSV file is organised into 3 columns separated by tabulations (and not by comas, as expected for CSV files as it is easier using streams to read tab or space separated data using C++):

```
RPC_channel        TDC_channel        mask
```

using as formatting for each field:

```
TSCCC         TCCC         M
```

`TSCCC` is a 5-digit integer where `T` is the trolley ID, `S` the slot ID in which the RPC is held insite the trolley `T` and `CCC` is the RPC channel number, or *strip* number, that can take values up to 3-digits depending on the detector,

`TCCC` is a 4 digit integer where `T` is the TDC ID, `CCC` is the TDC channel number that can take values in between 0 and 127, and

`M` is a 1-digit integer indicating if the channel should be considered (`M` $= 1$) or discarded (`M` $= 0$) during analysis.

This mapping and masking information is readout and stored thanks to the object `Mapping`, presented in Source Code B.4. Similarly to `IniFile` objects, this class has private methods. The first one, `Mapping::CheckIfNewLine()` is used to find the newline character `'\n'` or return character `'\r'` (depending on which kind of operating system interacted with the file). This is used for the simple reason that the masking information has been introduced only during the year 2017 but the channel mapping files exist since 2015 and the very beginning of data taking at GIF++. This means that in the older data folders, before the upgrade, the channel mapping file only had 2 columns, the RPC channel and the TDC channel. For compatibility reasons, this method helps controling the character following the readout of the 2 first fields of a line. In case any end of line character is found, no mask information is present in the file and the default `M` $= 1$ is used. On the contrary, if the next character was a tabulation or a space, the mask information is present.

Once the 3 fields have been readout, the second private method `Mapping::CheckIfTDCCh()` is used to control that the TDC channel is an existing TDC channel. Finally, the information is stored into 3 different maps (`Link`, `ReverseLink` and `Mask`) thanks to the public method `Mapping::Read()`. `Link` allows to get the RPC channel by knowing the TDC channel while `ReverseLink` does the opposite by returning the TDC channel by knowing the RPC channel. Finally, `Mask` returns the mask associated to a given RPC channel.

```
typedef map<Uint,Uint> MappingData;

class Mapping {
    private:
        bool        CheckIfNewLine(char next);
        bool        CheckIfTDCCh(Uint channel);
        string      FileName;
        MappingData Link;
        MappingData ReverseLink;
        MappingData Mask;
        int         Error;

    public:
        Mapping();
        Mapping(string baseName);
        ~Mapping();

        void SetFileName(const string filename);
        int Read();
        Uint GetLink(Uint tdcchannel);
        Uint GetReverse(Uint rpcchannel);
        Uint GetMask(Uint rpcchannel);
};
```

Source Code B.4: Description of C++ object `Mapping` used as a parser for the channel mapping and mask file.

B.4 Description of GIF++ setup within the Offline Analysis tool

In the previous section, the tool input files have been discussed. The dimension file information is stored in a map hosted by the IniFile object. But this information is then used to create a series of new objects that helps defining the GIF++ infrastructure directly into the Offline Analysis. Indeed, from the `RPC`, to the more general `Infrastructure`, every element of the GIF++ infrastrucutre is recreated for each data analysis based on the information provided in input. All this information about the infrastructure will be used to assign each hit signal to a specific strip channel of a specific detector, and having a specific active area. This way, rate per unit area calculation is possible.

B.4.1 RPC objects

`RPC` objects have been developped to represent physical active detectors in GIF++ at the moment of data taking. Thus, there are as many RPC objects created during the analysis than there were active RPCs tested during a run. Each `RPC` hosts the information present in the corresponding INI slot group, as shown in B.3, and organises it using a similar architecture. This can been seen from Source Code B.5.

To make the object more compact, the lists of gap labels, of gap active areas and strip active areas are stored into `vector` dynamical containers. `RPC` objects are always contructed thanks to the dimension file information stored into the `IniFIle` and their ID, using the format `TtSs`. Using the RPC ID, the constructor calls the methods of `IniFile` to initialise the `RPC`. The other constructors are not used but exist in case of need. Finally, some getters have been written to access the different private parameters storing the detector information.

```cpp
class RPC{
    private:
        string          name;        //RPC name as in webDCS database
        Uint            nGaps;       //Number of gaps in the RPC
        Uint            nPartitions; //Number of partitions in the RPC
        Uint            nStrips;     //Number of strips per partition
        vector<string>  gaps;        //List of gap labels (BOT, TOP, etc...)
        vector<float>   gapGeo;      //List of gap active areas
        vector<float>   stripGeo;    //List of strip active areas

    public:
        RPC();
        RPC(string ID, IniFile* geofile);
        RPC(const RPC& other);
        ~RPC();
        RPC& operator=(const RPC& other);

        string GetName();
        Uint   GetNGaps();
        Uint   GetNPartitions();
        Uint   GetNStrips();
        string GetGap(Uint g);
        float  GetGapGeo(Uint g);
        float  GetStripGeo(Uint p);
};
```

Source Code B.5: Description of C++ objects `RPC` that describe each active detectors used during data taking.

B.4.2 Trolley objects

`Trolley` objects have been developped to represent physical active trolleys in GIF++ at the moment of data taking. Thus, there are as many trolley objects created during the analysis than there were active trolleys hosting tested RPCs during a run. Each `Trolley` hosts the information present in the corresponding INI trolley group, as shown in B.2, and organises it using a similar architecture. In addition to the information hosted in the INI file, these object have a dynamical container of `RPC` objects, representing the active detectors the active trolley was hosting at the time of data taking. This can been seen from Source Code B.6.

`Trolley` objects are always contructed thanks to the dimension file information stored into the `IniFIle` and their ID, using the format `Tt`. Using the Trolley ID, the constructor calls the methods of `IniFile` to initialise the `Trolley`. Retrieving the information of the RPC IDs via `SlotsID`, a new `RPC` is constructed and added to the container `RPCs` for each character in the ID `string`. The other constructors are not used but exist in case of need. Finally, some getters have been written to access the different private parameters storing the trolley and detectors information.

```
class Trolley{
    private:
        Uint         nSlots;   //Number of active RPCs in the considered trolley
        string       SlotsID;  //Active RPC IDs written into a string
        vector<RPC*> RPCs;     //List of active RPCs

    public:
        //Constructors, destructor and operator =
        Trolley();
        Trolley(string ID, IniFile* geofile);
        Trolley(const Trolley& other);
        ~Trolley();
        Trolley& operator=(const Trolley& other);

        //Get GIFTrolley members
        Uint   GetNSlots();
        string GetSlotsID();
        Uint   GetSlotID(Uint s);

        //Manage RPC list
        RPC*   GetRPC(Uint r);
        void   DeleteRPC(Uint r);

        //Methods to get members of RPC objects stored in RPCs
        string GetName(Uint r);
        Uint   GetNGaps(Uint r);
        Uint   GetNPartitions(Uint r);
        Uint   GetNStrips(Uint r);
        string GetGap(Uint r, Uint g);
        float  GetGapGeo(Uint r, Uint g);
        float  GetStripGeo(Uint r, Uint p);
};
```

Source Code B.6: Description of C++ objects `Trolley` *that describe each active trolley used during data taking.*

B.4.3 Infrastructure object

The `Infrastructure` object has been developped to represent the GIF++ bunker area dedicated to CMS RPC experiments. With this very specific object, all the information about the CMS RPC setup within GIF++ at the moment of data taking is stored. It hosts the information present in the corresponding INI general group, as shown in B.1, and organises it using a similar architecture. In addition to the information hosted in the INI file, this object have a dynamical container of `Trolley` objects, representing the active tolleys in GIF++ area. This can been seen from Source Code B.7.

The `Infrastructure` object is always contructed thanks to the dimension file information stored into the `IniFIle`. Retrieving the information of the trolley IDs via `TrolleysID`, a new `Trolley` is constructed and added to the container `Trolleys` for each character in the ID `string`. By extension, it is easy to understand that the process described in Section B.4.2 for the construction of RPCs takes place when a trolley is constructed. The other constructors are not used but exist in case of need. Finally, some getters have been written to access the different private parameters storing the infrastructure, trolleys and detectors information.

```cpp
class Infrastructure {
    private:
        Uint             nTrolleys;  //Number of active Trolleys in the run
        string           TrolleysID; //Active trolley IDs written into a string
        vector<Trolley*> Trolleys;   //List of active Trolleys (struct)

    public:
        //Constructors and destructor
        Infrastructure();
        Infrastructure(IniFile* geofile);
        Infrastructure(const Infrastructure& other);
        ~Infrastructure();
        Infrastructure& operator=(const Infrastructure& other);

        //Get Infrastructure members
        Uint   GetNTrolleys();
        string GetTrolleysID();
        Uint   GetTrolleyID(Uint t);

        //Manage Trolleys
        Trolley* GetTrolley(Uint t);
        void     DeleteTrolley(Uint t);

        //Methods to get members of GIFTrolley objects stored in Trolleys
        Uint   GetNSlots(Uint t);
        string GetSlotsID(Uint t);
        Uint   GetSlotID(Uint t, Uint s);
        RPC*   GetRPC(Uint t, Uint r);

        //Methods to get members of RPC objects stored in RPCs
        string GetName(Uint t, Uint r);
        Uint   GetNGaps(Uint t, Uint r);
        Uint   GetNPartitions(Uint t, Uint r);
        Uint   GetNStrips(Uint t, Uint r);
        string GetGap(Uint t, Uint r, Uint g);
        float  GetGapGeo(Uint t, Uint r, Uint g);
        float  GetStripGeo(Uint t, Uint r, Uint p);
};
```

Source Code B.7: Description of C++ object `Infrastructure` that contains the full information about CMS RPC experiment in GIF++.

B.5 Handeling of data

As discussed in Appendix A.4.2, the raw data as a `TTree` architecture where every entry is related to a trigger signal provided by a muon or a random pulse, whether the goal of the data taking was to measure the performance of the detector or the noise/gamma background respectively. Each of these entries, referred also as events, contain a more or less full list of hits in the TDC channels to which the detectors are connected. To this list of hits corresponds a list of time stamps, marking the arrival of the hits within the TDC channel.

The infrastructure of the CMS RPC experiment within GIF++ being defined, combining the information about the raw data with the information provided by both the mapping/mask file and the dimension file allows to build new physical objects that will help in computing efficiency or rates.

B.5.1 RPC hits

The raw data stored in the ROOT file as output of the GIF++ DAQ, is readout by the analysis tool using the structure `RAWData` presented in Source Code B.9 that differs from the structure presented in Appendix A.4.2 as it is not meant to hold all of the data contained in the ROOT file. In this sense, this structure is in the case of the offline analysis tool not a dynamical object and will only be storing a single event contained in a single entry of the `TTree`.

```
class RPCHit {
    private:
        Uint  Channel;    //RPC channel according to mapping (5 digits)
        Uint  Trolley;    //0, 1 or 3 (1st digit of the RPC channel)
        Uint  Station;    //Slot where is held the RPC in Trolley (2nd digit)
        Uint  Strip;      //Physical RPC strip where the hit occured (last 3
 ↪  digits)
        Uint  Partition;  //Readout partition along eta segmentation
        float TimeStamp;  //Time stamp of the arrival in TDC

    public:
        //Constructors, destructor & operator =
        RPCHit();
        RPCHit(Uint channel, float time, Infrastructure* Infra);
        RPCHit(const RPCHit& other);
        ~RPCHit();
        RPCHit& operator=(const RPCHit& other);

        //Get RPCHit members
        Uint  GetChannel();
        Uint  GetTrolley();
        Uint  GetStation();
        Uint  GetStrip();
        Uint  GetPartition();
        float GetTime();
};

typedef vector<RPCHit> HitList;
typedef struct GIFHitList { HitList rpc[NTROLLEYS][NSLOTS][NPARTITIONS]; }
 ↪  GIFHitList;

bool SortHitbyStrip(RPCHit h1, RPCHit h2);
bool SortHitbyTime(RPCHit h1, RPCHit h2);
```

Source Code B.8: Description of C++ object `RPCHit`.

```
struct RAWData{
    int             iEvent;    //Event i
    int             TDCNHits;  //Number of hits in event i
    int             QFlag;     //Quality flag list (1 flag digit per TDC)
    vector<Uint>   *TDCCh;     //List of channels giving hits per event
    vector<float>  *TDCTS;     //List of the corresponding time stamps
};
```

Source Code B.9: Description of C++ structure `RAWData`.

Each member of the structure in then linked to the corresponding branch of the ROOT data tree, as shown in the example of Source Code B.10, and using the method `GetEntry(int i)` of the ROOT class `TTree` will update the state of the members of `RAWData`.

```
TTree*  dataTree = (TTree*)dataFile.Get("RAWData");
RAWData data;

dataTree->SetBranchAddress("EventNumber",    &data.iEvent);
dataTree->SetBranchAddress("number_of_hits", &data.TDCNHits);
dataTree->SetBranchAddress("Quality_flag",   &data.QFlag);
dataTree->SetBranchAddress("TDC_channel",    &data.TDCCh);
dataTree->SetBranchAddress("TDC_TimeStamp",  &data.TDCTS);
```

Source Code B.10: Example of link in between RAWData and TTree.

The data is then analysed entry by entry and to each element of the TDC channel list, a `RPCHit` is constructed by linking each TDC channel to the corresponding RPC channel thanks to the `Mapping` object. The information carried by the RPC channel format allows to easily retrieve the trolley and slot from which the hit was recorded (see section B.3.2). Using these 2 values, the readout partition can be found by knowing the strip channel and comparing it with the number of partitions and strips per partition stored into the `Infrastructure` object.

Thus `RPCHit` objects are then stored into 3D dynamical list called `GIFHitList` (Source Code B.9) where the 3 dimensions refer to the 3 layers of the readout in GIF++ : in the bunker there are *trolleys* (`T`) holding detectors in *slots* (`S`) and each detector readout is divided into 1 or more pseudo-rapidity *partitions* (`p`). Using these 3 information allows to assign an address to each readout partition and this address will point to a specific hit list.

B.5.2 Clusters of hits

All the hits contained in the ROOT file have been sorted into the different hit lists through the `GIFHitList`. At this point, it is possible to start looking for clusters. A cluster is a group of adjacent strips getting hits within a time window of 25 ns. These strips are then assumed to be part of the same physical avalanch signal generated by a muon passing through the chamber or by the interaction of a gamma stopping into the electrodes of the RPCs.

To keep the cluster information, `RPCCluster` objects have been defined as shown in Source Code B.11. Using the information of each individual `RPCHit` taken out of the hit list, it stores the cluster size (number of adjacent strips composing the cluster), the first and last hit, the center for spatial reconstruction and finally the start and stop time stamps as well as te time spread in between the first and last hit.

```
class RPCCluster{
    private:
        Uint  ClusterSize;  //Size of cluster #ID
        Uint  FirstStrip;   //First strip of cluster #ID
        Uint  LastStrip;    //Last strip of cluster #ID
        float Center;       //Center of cluster #ID ((first+last)/2)
        float StartStamp;   //Time stamp of the earliest hit of cluster #ID
        float StopStamp;    //Time stamp of the latest hit of cluster #ID
        float TimeSpread;   //Time difference between earliest and latest hits
                            //of cluster #ID
    public:
        //Constructors, destructor & operator =
        RPCCluster();
        RPCCluster(HitList List, Uint cID, Uint cSize, Uint first, Uint firstID);
        RPCCluster(const RPCCluster& other);
        ~RPCCluster();
        RPCCluster& operator=(const RPCCluster& other);

        //Get Cluster members
        Uint GetID();
        Uint GetSize();
        Uint GetFirstStrip();
        Uint GetLastStrip();
        float GetCenter();
        float GetStart();
        float GetStop();
        float GetSpread();
};

typedef vector<RPCCluster> ClusterList;

//Other functions to build cluster lists out of hit lists
void BuildClusters(HitList &cluster, ClusterList &clusterList);
void Clusterization(HitList &hits, TH1 *hcSize, TH1 *hcMult);
```

Source Code B.11: Description of C++ object `Cluster`.

To investigate the hit list of a given detector partition, the function `Clusterization()` definied in `include/Cluster.h` needs the hits in the list to be time sorted. This is achieved by calling function `sort()` of library `<algorithm>` using the comparator `SortHitbyTime(RPCHit h1, RPCHit h2)` defined in `include/RPCHit.h` that returns `true` if the time stamp of hit `h1` is lower than that of `h2`. A first isolation of strips is made only based on time information. All the hits within the 25 ns window are taken separately from the rest. Then, this sub-list of hits is sorted this time by ascending strip number, using this time the comparator `SortHitbyStrip(RPCHit h1, RPCHit h2)`. Finally, the groups of adjacent strips are used to construct `RPCCluster` objects that are then stored in a temporary list of clusters that is at the end of the process used to know how many clusters were reconstructed and to fill their sizes into an histogram that will allows to know the mean size of muon or gamma clusters.

B.6 DAQ data Analysis

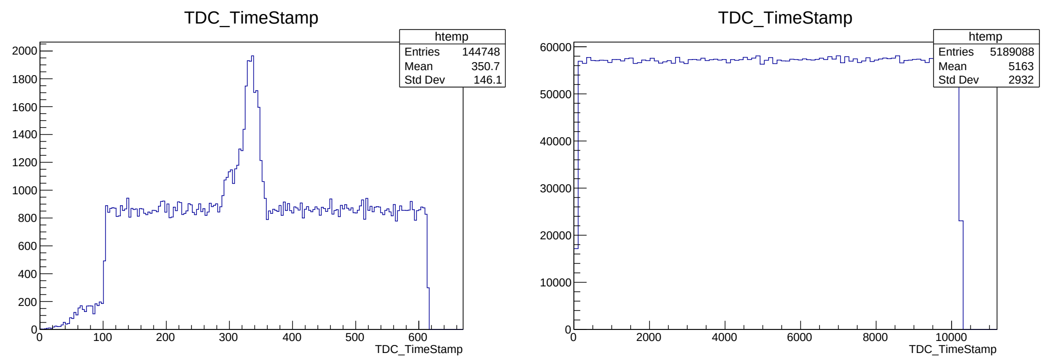
All the ingredients to analyse GIF++ data have been defined. This section will focus on the different part of the analysis performed on the data, from determining the type of data the tool is dealing with to calculating the rate in each detector or reconstructing muon or gamma clusters.

B.6.1 Determination of the run type

In GIF++, both the performance of the detectors in detecting muons in an irradiated environment and the gamma background can be independantly measured. These corresponds to different run types and thus, to different TDC settings giving different data to look at.

In the case of performance measurements, the trigger for data taking is provided by the coïncidence of several scintillators when muons from the beam passing through the area are detected. Data is collected in a 600 ns wide window around the arrival of muons in the RPCs. The expected time distribution of hits is shown in Figure B.1a. The muon peak is clearly visible in the center of the distribution and is to be extracted from the gamma background that composes the flat part of the distribution.

On the other hand, gamma background or noise measurements are focussed on the non muon related physics and the trigger needs to be independant from the muons to give a good measurement of the gamma/noise distribution as seen by the detectors. The trigger is then provided by a pulse generator at a frequency of 300 Hz whose pulse is not likely to be on time with a muon. In order to increase the integrated time without increasing the acquisition time too much, the width of the acquisition windows are increased to 10 µs. The time distribution of the hits is expected to be flat, as shown by Figure B.1b.



(a) *(b)*

Figure B.1: Example of expected hit time distributions in the cases of efficiency (Figure B.1a) and noise/gamma rate per unit area (Figure B.1b) measurements as extracted from the raw ROOT files. The unit along the x-axis corresponds to ns. *The fact that "the" muon peak is not well defined in Figure B.1a is due to the contribution of all the RPCs being tested at the same time that don't necessarily have the same signal arrival time. Each individual peak can have an offset with the ones of other detectors. The inconsistancy in the first* 100 ns *of both time distributions is an artefact of the TDCs and are systematically rejected during the analysis.*

The ROOT files include a `TTree` called `RunParameters` containing, among other things, the information related to the type of run. The run type can then be accessed as described by Source Code B.12 and the function `IsEfficiencyRun()` is then used to determine if the run file is an efficiency run or, on the contrary, another type of run (noise or gamma measurement).

```
TTree* RunParameters = (TTree*)dataFile.Get("RunParameters");
TString* RunType = new TString();
RunParameters->SetBranchAddress("RunType",&RunType);
RunParameters->GetEntry(0);
```

Source Code B.12: Access to the run type contained in `TTree* RunParameters`.

Finally, the data files will have a slightly different content whether it was collected before or after October 2017 and the upgrade of the DAQ software that brought a new information into the ROOT output. This is discussed in Appendix A.4.3 and implies that the analysis will differ a little depending on the data format. Indeed, as no information on the data quality is stored, in older data files, the corrections for missing events has to be done at the end of the analysis. The information about the type of data format is stored in the variable `bool isNewFormat` by checking the list of branches contained in the data tree via the methods `TTree::GetListOfBranches()` and `TCollection::Contains()`.

B.6.2 Beam time window calculation for efficiency runs

Knowing the run type is important first of all to know the width of the acquisition window to be used for the rate calculation and finally to be able to seek for muons. Indeed, the peak that appears in the time distribution for each detectors is then fitted to extract the most probable time window in which the tool should look for muon hits. The data outside of this time window in then used to evaluate the noise or gamma background the detector was subbjected to during the data taking. Computing the position of the peak is done calling the function `SetBeamWindow()` defined in file `src/RPCHit.cc` that loops a first time on the data. The data is first sorted in a 3D array of 1D histograms (`GIFH1Array`, see `include/types.h`). Then the location of the highest bin is determined using `TH1::GetMaximumBin()` and is used to define a window in which a gaussian fit will be applied to compute the peak width. This window is a 80 ns defined by Formula B.1 around the central bin.

$$t_{center}(ns) = bin \times width_{bin}(ns) \tag{B.1a}$$

$$[t_{low};t_{high}] = [t_{center} - 40; t_{center} + 40] \tag{B.1b}$$

Before the fit is performed, the average number of noise/gamma hits per bin is evaluated using the data outside of the fit window. Excluding the first 100 ns, the average number of hits per bin due to the noise or gamma is defined by Formula B.2 after extracting the amount of hits in the time windows $[100; t_{low}]$ and $[t_{high}; 600]$ thanks to the method `TH1::Integral()`. This average number of hits is then subtracted to every bin of the 1D histogram, in order to *clean* it from the noise or gamma contribution as much as possible to improve the fit quality. Bins where $\langle n_{hits} \rangle$ is greater than the actual bin content are set to 0.

$$\Delta t_{noise}(ns) = 600 \overbrace{- t_{high} + t_{low}}^{-80ns} - 100 = 420ns \tag{B.2a}$$

$$\langle n_{hits} \rangle = width_{bin}(ns) \times \frac{\sum_{t=100}^{t_{low}} + \sum_{t=t_{high}}^{600}}{\Delta t_{noise}(ns)} \tag{B.2b}$$

Finally, the fit parameters are extracted and saved for each detector in 3D arrays of `float` (`muonPeak`, see `include/types.h`), a first one for the mean arrival time of the muons, `PeakTime`, and a second one for the width of the peak, `PeakWidth`. The width is defined as 6σ of the gaussian fit. The same settings are applied to every partitions of the same detector. To determine which one of the detector's partitions is directly illuminated by the beam, the peak height of each partition is compared and the highest one is then used to define the peak settings.

B.6.3 Data loop and histogram filling

3D arrays of histogram are created to store the data and display it on the DQM of GIF++ webDCS for the use of shifters. These histograms, presented in section B.2.1.1, are filled while looping on the data. Before starting the analysis loop, it is necessary to control the entry quality for the new file formats featuring `QFlag`. If the `QFlag` value for this entry shows that 1 TDC or more have a `CORRUPTED` flag, then this event is discarded. The loss of statistics is low enough to be neglected. `QFlag` is controled using the function `IsCorruptedEvent()` defined in `src/utils.cc`. As explained in Appendix A.4.3, each digit of this integer represent a TDC flag that can be 1 or 2. Each 2 is the sign of a `CORRUPTED` state. Then, the data is accessed entry by entry in the ROOT `TTree` using `RAWData` and each hit in the hit list is assigned to a detector channel and saved in the corresponding histograms. In the first part of the analysis, in which the loop over the ROOT file's content is performed, the different steps are:

1- RPC channel assignment and control: a check is done on the RPC channel extracted thanks to the mapping via the method `Mapping::GetLink()`. If the channel is not initialised and is 0, or if the TDC channel was greater than 5127, the hit is discarded. This means there was a problem in the mapping. Often a mapping problem leads to the crash of the offline tool.

2- Creation of a `RPCHit` object: to easily get the trolley, slot and partition in which the hit has been assigned, this object is particularly helpful.

3- General histograms are filled: the hit is filled into the time distribution and the general hit distribution histograms, and if the arrival time is within the first 100 ns, it is discarded and nothing else happens and the loop proceeds with the next hit in the list.

4- Multiplicity counter: the hit multiplicity counter of the corresponding detectors incremented.

5-a- *Effiency runs* - Is the hit within the peak window? : if the peak is contained in the peak window previously defined in section B.6.2, the hit is filled into the beam hit profile histogram of the corresponding chamber, added into the list of muon hits and increments the counter of *in time* hits. The term *in time* here refers to the hits that are likely to be muons by arriving in the expected time window. If the hit is outside of the peak window, it is filled into the noise profile histogram of the corresponding detector, added into the list of noise/gamma hits and increments the counter of noise/gamma hits.

5-b- *Noise/gamma rate runs* - Noise histograms are filled: the hit is filled into the noise profile histogram of the corresponding detector, added into the list of noise/gamma hits and increments the counter of noise/gamma hits.

After the loop on the hit list of the entry is over, the next step is too clusterize the 3D lists filled in the previous steps. A 3D loop is then started over the active trolley, slot and RPC partitions to access these objects. Each `NoiseHitList` and `MuonHitList`, in case of efficiency run, are clusterized as decribed in section B.5.2. There corresponding cluster size and multiplicity histograms are filled at the end of the clustering process. Then, the effiency histogram is filled in case of effieincy run. The selection is simply made by checking whether the RPC detected signals in the peak window during this event. Nevertheless, it is useful to highlight that at this level, it is not possible yet to discriminate in between a muon hit and noise or gamma hit. Thus, `MuonCSize_H`, `MuonCMult_H` and `Efficiency0_H` are subjected to noise and gamma contamination. This contamination will be estimated and corrected at the moment the results will be written into output CSV files. Finally, the loop ends on the filling of the general hit multiplicity histogram.

B.6.4 Results calculation

As mentioned in section B.2.1, the analysis of DAQ data provides the user with 3 CSV files and a ROOT file associated to each and every ROOT data file. The fourth CSV file is provided by the extraction of the CEAN main frame data monitored during data tacking and will be discussed later. After looping on the data in the previous part of the analysis macro, the output files are created and a 3D loop on each RPC readout partitions is started to extract the histograms parameters and compute the final results.

B.6.4.1 Rate normalisation

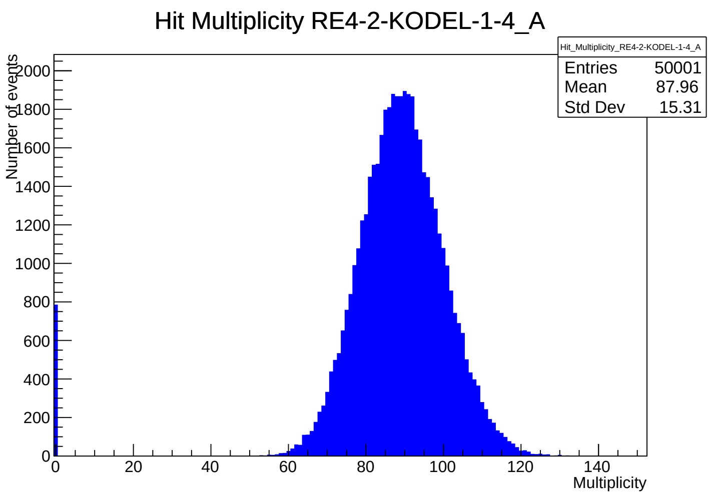


Figure B.2: The effect of the quality flag is explained by presenting the reconstructed hit multiplicity of a data file without `Quality_flag`. The artificial high content of bin 0 is the effect of corrupted data.

To analyse old data format files, not containing any quality flag, it is needed to estimate the amount of corrupted data via a fit as the corrupted data will always fill events with a fake ”0 multiplicity”. Indeed, as no hits were stored in the DAQ ROOT files, these events artificially contribute to fill the bin corresponding to a null multiplicity, as shown in Figure B.2. In the case the mean of the hit multiplicity distribution is high, the contribution of the corrupted data can easily be evaluated for later correction by comparing the level of the bin at multiplicity 0 and of a skew fit curve that should indicate a value consistent with 0. A skew fit has been chosen over a Poisson fit as it was giving better results for lower mean multiplicity values. Nevertheless, for low irradiation cases, as explained in Appendix A.4.3, the hit multiplicity distribution mean is, on the contrary, rather small and the probability to record events without hits can't be considered small anymore, leading to a difficult and non-reliable estimation of the corruption. As can be seen in Source Code B.13, conditions have been applied to prevent bad fits and wrong corruption estimation in cases where :

- The difference in between the data for multiplicity 1 and the corresponding fit value should be lower than 1% of the total amount of data : $\frac{|n_{m=1}-sk(1)|}{N_{tot}} < 0.01$ where $n_{m=1}$ is the number of entries with multiplicity 1, $sk(1)$ the value of the skew fit, as defined by Formula 5.3, for multiplicity 1 and N_{tot} the total number of entries.

- The amount of data contained in the multiplicity 0 bin should not exceed 40% : $\frac{n_{m=0}}{N_{tot}} \leq 0.4$ where $n_{m=0}$ is the number of entries with multiplicity 0. This number has been determined to be the maximum to be able to separate the excess of data due to corruption from the hit multiplicity distribution.

Those 2 conditions need to be fulfilled to estimate the corruption of old data format files. If the fit was successful, the level of corruption is written in `Offline-Corrupted.csv` and the number of corrupted entries, refered as the integer `nEmptyEvent`, is subtracted from the total number of entries when the rate normalisation factor is computed as explicited in Source Code B.13. Note that for new data format files, the number of corrupted entries being set to 0, the definition of `rate_norm` stays valid.

```
if(!isNewFormat){
    TF1* GaussFit = new TF1("gaussfit","[0]*exp(-0.5*((x-[1])/[2])**2)",0,Xmax);
    GaussFit->SetParameter(0,100);
    GaussFit->SetParameter(1,10);
    GaussFit->SetParameter(2,1);
    HitMultiplicity_H.rpc[T][S][p]->Fit(GaussFit,"LIQR","",0.5,Xmax);

    TF1* SkewFit = new TF1("skewfit","[0]*exp(-0.5*((x-[1])/[2])**2) / (1 +
  ↪  exp(-[3]*(x-[4])))",0,Xmax);
    SkewFit->SetParameter(0,GaussFit->GetParameter(0));
    SkewFit->SetParameter(1,GaussFit->GetParameter(1));
    SkewFit->SetParameter(2,GaussFit->GetParameter(2));
    SkewFit->SetParameter(3,1);
    SkewFit->SetParameter(4,1);
    HitMultiplicity_H.rpc[T][S][p]->Fit(SkewFit,"LIQR","",0.5,Xmax);

    double fitValue = SkewFit->Eval(1,0,0,0);
    double dataValue = (double)HitMultiplicity_H.rpc[T][S][p]->GetBinContent(2);
    double difference = TMath::Abs(dataValue - fitValue);
    double fitTOdataVSentries_ratio = difference / (double)nEntries;
    bool isFitGOOD = fitTOdataVSentries_ratio < 0.01;

    double nSinglehit = (double)HitMultiplicity_H.rpc[T][S][p]->GetBinContent(1);
    double lowMultRatio = nSinglehit / (double)nEntries;
    bool isMultLOW = lowMultRatio > 0.4;

    if(isFitGOOD && !isMultLOW){
        nEmptyEvent = HitMultiplicity_H.rpc[T][S][p]->GetBinContent(1);
        nPhysics = (int)SkewFit->Eval(0,0,0,0);
        if(nPhysics < nEmptyEvent)
        nEmptyEvent = nEmptyEvent-nPhysics;
    }
}

double corrupt_ratio = 100.*(double)nEmptyEvent / (double)nEntries;
outputCorrCSV << corrupt_ratio << '\t';

float rate_norm = 0.;
float stripArea = GIFInfra->GetStripGeo(tr,sl,p);

if(IsEfficiencyRun(RunType)){
    float noiseWindow = BMTDCWINDOW - TIMEREJECT - 2*PeakWidth.rpc[T][S][p];
    rate_norm = (nEntries-nEmptyEvent)*noiseWindow*1e-9*stripArea;
} else
    rate_norm = (nEntries-nEmptyEvent)*RDMNOISEWDW*1e-9*stripArea;
```

Source Code B.13: Definition of the rate normalisation variable. It takes into account the number of non corrupted entries and the time window used for noise calculation, to estimate the total integrated time, and the strip active area to express the result as rate per unit area.

B.6.4.2 Rate and activity

At this point, the strip rate histograms, `StripNoiseProfile_H.rpc[T][S][p]`, only contain an information about the total number of noise or rate hits each channel received during the data taking. As described in Source Code B.14, a loop on the strip channels will be used to normalise the content of the rate distribution histogram for each detector partitions. The initial number of hits recorded for a given bin will be extracted and 2 values will be computed:

- the strip rate, defined as the number of hits recorded in the bin normalised like described in the previous section, using the variable rate_norm, and

- the strip activity, defined as the number of hits recorded in the bin normalised to the average number of hits per bin contained in the partition histogram, using the variable averageNhit. This value provides an information on the homogeneity of the detector response to the gamma background or of the detector noise. An activity of 1 corresponds to an average response. Above 1, the channel is more active than the average and bellow 1, the channel is less active.

```
int nNoise = StripNoiseProfile_H.rpc[T][S][p]->GetEntries();
float averageNhit = (nNoise>0) ? (float)(nNoise/nStripsPart) : 1.;

for(Uint st = 1; st <= nStripsPart; st++){
    float stripRate =
        StripNoiseProfile_H.rpc[T][S][p]->GetBinContent(st)/rate_norm;
    float stripAct =
        StripNoiseProfile_H.rpc[T][S][p]->GetBinContent(st)/averageNhit;

    StripNoiseProfile_H.rpc[T][S][p]->SetBinContent(st,stripRate);
    StripActivity_H.rpc[T][S][p]->SetBinContent(st,stripAct);
}
```

Source Code B.14: Description of the loop that allows to set the content of each strip rate and strip activity channel for each detector partition.

On each detector partitions, which are readout by a single FEE, all the channels are not processed by the same chip. Each chip can give a different noise response and thus, histograms using a chip binning are used to investigate chip related noise behaviours. The average values of the strip rate or activity grouped into a given chip are extracted using the using the function GetChipBin() and stored in dedicated histograms as described in Source Codes B.15 and B.16 respectively.

```
float GetChipBin(TH1* H, Uint chip){
    Uint start = 1 + chip*NSTRIPSCHIP;
    int nActive = NSTRIPSCHIP;
    float mean = 0.;

    for(Uint b = start; b <= (chip+1)*NSTRIPSCHIP; b++){
        float value = H->GetBinContent(b);
        mean += value;
        if(value == 0.) nActive--;
    }

    if(nActive != 0) mean /= (float)nActive;
    else mean = 0.;

    return mean;
}
```

Source Code B.15: Function used to compute the content of a bin for an histogram using chip binning.

```
for(Uint ch = 0; ch < (nStripsPart/NSTRIPSCHIP); ch++){
    ChipMeanNoiseProf_H.rpc[T][S][p]->
        SetBinContent(ch+1,GetChipBin(StripNoiseProfile_H.rpc[T][S][p],ch));
    ChipActivity_H.rpc[T][S][p]->
        SetBinContent(ch+1,GetChipBin(StripActivity_H.rpc[T][S][p],ch));
}
```

Source Code B.16: Description of the loop that allows to set the content of each chip rate and chip activity bins for each detector partition knowing the information contained in the corresponding strip distribution histograms.

The activity variable is used to evaluate the homogeneity of the detector response to background or of the detector noise. The homogeneity h_p of each detector partition can be evaluated using the formula $h_p = exp(-\sigma_p^R/\langle R\rangle_p)$, where $\langle R\rangle_p$ is the partition mean rate and σ_p^R is the rate standard deviation calculated over the partition channels. The more homogeneously the rates are distributed and the smaller will σ_p^R be, and the closer to 1 will h_p get. On the contrary, if the standard deviation of the channel's rates is large, h_p will rapidly get to 0. This value is saved into histograms as shown in Source Code B.17 and could in the future be used to monitor through time, once extracted, the evolution of every partition homogeneity. This could be of great help to understand the apparition of eventual hot spots due to ageing of the chambers subjected to high radiation levels. The monitored homogeneity information could then be combined with a monitoring of the activity of each individual channel in order to have a finer information. Monitoring tools have been suggested and need to be developped for this purpose.

```
float MeanPartSDev = GetTH1StdDev(StripNoiseProfile_H.rpc[T][S][p]);
float strip_homog = (MeanPartRate==0)
    ? 0.
    : exp(-MeanPartSDev/MeanPartRate);
StripHomogeneity_H.rpc[T][S][p]->Fill("exp -#left(#frac{#sigma_{Strip
 ↪  Rate}}{#mu_{Strip Rate}}#right)",strip_homog);
StripHomogeneity_H.rpc[T][S][p]->GetYaxis()->SetRangeUser(0.,1.);

float ChipStDevMean = GetTH1StdDev(ChipMeanNoiseProf_H.rpc[T][S][p]);
float chip_homog = (MeanPartRate==0)
    ? 0.
    : exp(-ChipStDevMean/MeanPartRate);
ChipHomogeneity_H.rpc[T][S][p]->Fill("exp -#left(#frac{#sigma_{Chip
 ↪  Rate}}{#mu_{Chip Rate}}#right)",chip_homog);
ChipHomogeneity_H.rpc[T][S][p]->GetYaxis()->SetRangeUser(0.,1.);
```

Source Code B.17: Storage of the homogeneity into dedicated histograms.

B.6.4.3 Strip masking tool

The offline tool is automatically called at the end of each data taking to analyse the data and offer the shifter DQM histograms to control the data quality. After the histograms have been published online in the DQM page, the shifter can decide to mask noisy or dead channels that will contribute to bias the final rate calculation by editing the mask column of `ChannelsMapping.csv` as can be seen in Figure B.3.

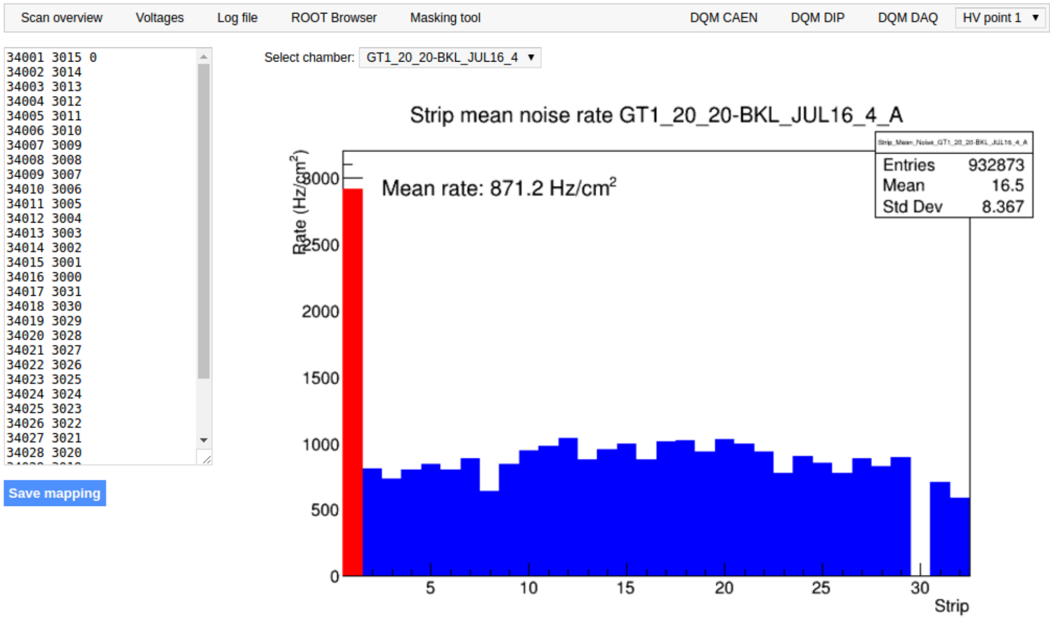


Figure B.3: Display of the maskting tool page on the webDCS. The window on the left allows the shifter to edit `ChannelsMapping.csv`. To mask a channel, it only is needed to set the 3rd field corresponding to the strip to mask to 0. It is not necessary for older mapping file formats to add a 1 for each strip that is not masked as the code is versatile and the default behaviour is to consider missing mask fields as active strips. The effect of the mask is directly visible for noisy channels as the corresponding bin turns red. The global effect of masking strips will be an update of the rate value showed on the histogram that will take into consideration the rejected channels.

From the code point of view, the function `GetTH1Mean()` is used to retrieve the mean rate partition by partition after the rates have been calculated strip by strip and filled into the histograms `StripNoiseProfile_H.rpc[T][S][p]`, as described through Source Code B.18.

Once the mask for each rejected channel has been updated, the shifter can manually run the offline tool again to update the DQM plots, now including the masked strips, as well the rate results written in the output CSV file `Offline-Rate.csv`. If not done during the shifts, the strip masking procedure needs to be carefully done by the person in charge of data analysis on the scans that were selected to produce the final results.

```
float GetTH1Mean(TH1* H){
    int nBins = H->GetNbinsX();
    int nActive = nBins;
    float mean = 0.;

    for(int b = 1; b <= nBins; b++){
        float value = H->GetBinContent(b);
        mean += value;
        if(value == 0.) nActive--;
    }

    if(nActive != 0) mean /= (float)nActive;
    else mean = 0.;

    return mean;
}
```

Source Code B.18: The function `GetTH1Mean()` is used to return the mean along the y-axis of `TH1` histograms containing rate information. In order to take into account masked strips whose rate is set to 0, the function looks for masked channels and decrement the number of active channels for each null value found.

B.6.4.4 Output CSV files filling

All the histograms have been filled. Parameters will then be extracted from them to compute the final results that will later be used to produce plots. Once the results have been computed, the very last step of the offline macro is to write these values into the corresponding CSV outputs. Aside of the file `Offline-Corrupted.csv`, 2 CSV files are being written by the macro `OfflineAnalysis()`, `Offline-Rates.csv` and `Offline-L0-EffCl.csv` that respectively contain information about noise or gamma rates, cluster size and multiplicity, and about level 0 reconstruction of the detector efficiency, muon cluster size and multiplicity. Details on the computation and file writing are respectively given in Sources Codes B.19 and B.20.

Noise/gamma background variables are computed and written in the output file for each detector partitions. A detector average of the hit and cluster rate is also provided, as shown through Sources Code B.19. The variables that are written for each partition are:

- The mean partition hit rate per unit area, `MeanPartRate`, that is extracted from the histogram `StripNoiseProfile_H` as the mean value along the y-axis, as described in section B.6.4.3. No error is recorded for the hit rate as this is considered a single measurement. No statistical error can be associated to it and the systematics are unknown.

- The mean cluster size, `cSizePart`, is extracted from the histogram `NoiseCSize_H` and it's statistical error, `cSizePartErr`, is taken to be 2σ of the total distribution.

- The mean cluster multiplicity per trigger, `cMultPart`, is extracted from the histogram `NoiseCMult_H` and it's statistical error, `cMultPartErr`, is taken to be 2σ of the total distribution. It is important to point to the fact that this variable gives an information that is dependent on the buffer window width used for each trigger for the calculation.

- The mean cluster rate per unit area, `ClustPartRate`, is defined as the mean hit rate normalised to the mean cluster size and it's statistical error, `ClustPartRateErr`, is then obtained using the relative statistical error on the mean cluster size.

```
for (Uint tr = 0; tr < GIFInfra->GetNTrolleys(); tr++){
    Uint T = GIFInfra->GetTrolleyID(tr);

    for (Uint sl = 0; sl < GIFInfra->GetNSlots(tr); sl++){
        Uint S = GIFInfra->GetSlotID(tr,sl) - 1;

        float MeanNoiseRate = 0.;
        float ClusterRate   = 0.;
        float ClusterSDev   = 0.;

        for (Uint p = 0; p < GIFInfra->GetNPartitions(tr,sl); p++){
            float MeanPartRate = GetTH1Mean(StripNoiseProfile_H.rpc[T][S][p]);
            float cSizePart = NoiseCSize_H.rpc[T][S][p]->GetMean();
            float cSizePartErr = (NoiseCSize_H.rpc[T][S][p]->GetEntries()==0)
                ? 0.
                : 2*NoiseCSize_H.rpc[T][S][p]->GetStdDev() /
                            sqrt(NoiseCSize_H.rpc[T][S][p]->GetEntries());
            float cMultPart = NoiseCMult_H.rpc[T][S][p]->GetMean();
            float cMultPartErr = (NoiseCMult_H.rpc[T][S][p]->GetEntries()==0)
                ? 0.
                : 2*NoiseCMult_H.rpc[T][S][p]->GetStdDev() /
                            sqrt(NoiseCMult_H.rpc[T][S][p]->GetEntries());
            float ClustPartRate = (cSizePart==0) ? 0.
                : MeanPartRate/cSizePart;
            float ClustPartRateErr = (cSizePart==0) ? 0.
                : ClustPartRate * cSizePartErr/cSizePart;

            outputRateCSV << MeanPartRate << '\t'
                << cSizePart << '\t' << cSizePartErr << '\t'
                << cMultPart << '\t' << cMultPartErr << '\t'
                << ClustPartRate << '\t' << ClustPartRateErr << '\t';

            RPCarea       += stripArea * nStripsPart;
            MeanNoiseRate += MeanPartRate * stripArea * nStripsPart;
            ClusterRate   += ClustPartRate * stripArea * nStripsPart;
            ClusterSDev   += (cSizePart==0)
                ? 0.
                : ClusterRate*cSizePartErr/cSizePart;
        }

        MeanNoiseRate /= RPCarea;
        ClusterRate   /= RPCarea;
        ClusterSDev   /= RPCarea;

        outputRateCSV << MeanNoiseRate << '\t'
            << ClusterRate << '\t' << ClusterSDev << '\t';
    }
}
```

Source Code B.19: Description of rate result calculation and writing into the CSV output `Offline-Rate.csv`. Are saved into the file for each detector, the mean partition rate, cluster size and cluster mutiplicity, along with their errors, for each partition and as well as a detector average.

Muon performance variables are computed and written in the output file for each detector partitions as shown through Sources Code B.20. The variables that are written for each partition are:

- The muon efficiency, `eff`, extracted from the histogram `Efficiency0_H`. It is reminded that this offline tool doesn't include any tracking algorithm to identify muons from the beam and only relies on the hits arriving in the time window corresponding to the beam time. The content of the efficiency histogram is thus biased by the noise/gamma background contribution into this window and is thus corrected by estimating the muon data content in the peak region knowing the noise/gamma content in the rate calculation region. Both time windows being different, the choice was made to normalise the noise/gamma background calculation window to it's equivalent beam window in order to have comparable values using the variable `windowRatio`. Finally, to estimate the data ratio in the peak region, the variable `DataRatio` is defined as the ratio in between the estimated mean cluster multiplicity of the muons in the peak region, `MuonCM`, and of the total mean cluster multiplicity in the peak region, `PeakCM`. `MuonCM` is itself defined as the difference in between the total mean cluster multiplicity in the peak region and the normalised mean noise/gamma cluster multiplicity calculated outside of the peak region. The statistical error related to the efficiency, `eff_err`, is computed using a binomial distribution, as the efficiency measure the probability of "success" and "failure" to detect muons.

- The mean muon cluster size, `MuonCS`, is calculated using the total mean cluster size and multiplicity in the peak region, respectively extracted from histograms `MuonCSize_H` and `MuonCMult_H`, the noise/gamma background mean cluster size and normalised multiplicity, extracted from `NoiseCSize_H` and `NoiseCMult_H`, and of the estimated muon cluster multiplicity `MuonCM` previously explicited. The associated statistical error, `MuonCM_err`, is calculated using the propagation of errors of the mentioned variables.

- The mean muon cluster multiplicity in the peak region, `MuonCM`, explicited above whose statistical error, `MuonCM_err`, is the sum of statistical error associated to the total mean cluster multiplicity in the peak reagion, `PeakCM_err`, and of the mean noise/gamma cluster size, `NoiseCM_err`.

In addition to these 2 CSV files, the histograms are saved in ROOT file `Scan00XXXX_HVY_Offline.root` as explained in section B.2.1.1.

```
for (Uint tr = 0; tr < GIFInfra->GetNTrolleys(); tr++){
    Uint T = GIFInfra->GetTrolleyID(tr);
    for (Uint sl = 0; sl < GIFInfra->GetNSlots(tr); sl++){
        Uint S = GIFInfra->GetSlotID(tr,sl) - 1;
        for (Uint p = 0; p < GIFInfra->GetNPartitions(tr,sl); p++){
            float noiseWindow =
                BMTDCWINDOW - TIMEREJECT - 2*PeakWidth.rpc[T][S][p];
            float peakWindow = 2*PeakWidth.rpc[T][S][p];
            float windowRatio = peakWindow/noiseWindow;

            float PeakCM = MuonCMult_H.rpc[T][S][p]->GetMean();
            float PeakCS = MuonCSize_H.rpc[T][S][p]->GetMean();
            float NoiseCM = NoiseCMult_H.rpc[T][S][p]->GetMean()*windowRatio;
            float NoiseCS = NoiseCSize_H.rpc[T][S][p]->GetMean();
            float MuonCM = (PeakCM<NoiseCM) ? 0. : PeakCM-NoiseCM;
            float MuonCS = (MuonCM==0 || PeakCM*PeakCS<NoiseCM*NoiseCS)
                ? 0.
                : (PeakCM*PeakCS-NoiseCM*NoiseCS)/MuonCM;
            float PeakCM_err = (MuonCMult_H.rpc[T][S][p]->GetEntries()==0.)
                ? 0.
                : 2*MuonCMult_H.rpc[T][S][p]->GetStdDev() /
                            sqrt(MuonCMult_H.rpc[T][S][p]->GetEntries());
            float PeakCS_err = (MuonCSize_H.rpc[T][S][p]->GetEntries()==0.)
                ? 0.
                : 2*MuonCSize_H.rpc[T][S][p]->GetStdDev() /
                            sqrt(MuonCSize_H.rpc[T][S][p]->GetEntries());
            float NoiseCM_err = (NoiseCMult_H.rpc[T][S][p]->GetEntries()==0.)
                ? 0.
                : windowRatio*2*NoiseCMult_H.rpc[T][S][p]->GetStdDev() /
                            sqrt(NoiseCMult_H.rpc[T][S][p]->GetEntries());
            float NoiseCS_err = (NoiseCSize_H.rpc[T][S][p]->GetEntries()==0.)
                ? 0.
                : 2*NoiseCSize_H.rpc[T][S][p]->GetStdDev() /
                            sqrt(NoiseCSize_H.rpc[T][S][p]->GetEntries());
            float MuonCM_err = (MuonCM==0) ? 0. : PeakCM_err+NoiseCM_err;
            float MuonCS_err = (MuonCS==0 || MuonCM==0) ? 0.
                : (PeakCS*PeakCM_err + PeakCM*PeakCS_err +
                  NoiseCS*NoiseCM_err + NoiseCM*NoiseCS_err +
                  MuonCS*MuonCM_err)/MuonCM;

            float DataRatio = MuonCM/PeakCM;
            float DataRatio_err = (MuonCM==0) ? 0.
                : DataRatio*(MuonCM_err/MuonCM + PeakCM_err/PeakCM);
            float eff = DataRatio*Efficiency0_H.rpc[T][S][p]->GetMean();
            float eff_err =  DataRatio*2*Efficiency0_H.rpc[T][S][p]->GetStdDev()/
                        sqrt(Efficiency0_H.rpc[T][S][p]->GetEntries()) +
                        Efficiency0_H.rpc[T][S][p]->GetMean()*DataRatio_err;

            outputEffCSV << eff << '\t' << eff_err << '\t'
                << MuonCS << '\t' << MuonCS_err << '\t'
                << MuonCM << '\t' << MuonCM_err << '\t';
        }
    }
}
```

Source Code B.20: Description of efficiency result calculation and writing into the CSV output `Offline-L0-EffCl.csv`*. Are saved into the file for each detector, the efficiency, corrected taking into account the background in the peak window of the time profile, muon cluster size and muon cluster mutiplicity, along with their errors, for each partition and as well as a detector average.*

B.7 Current data Analysis

Detectors under test at GIF++ are connected both to a CAEN HV power supply and to a CAEN ADC that reads the currents inside of the RPC gaps bypassing the supply cable. During data taking, the webDCS records into a ROOT file called `Scan00XXXX_HVY_CAEN.root` histograms with the monitored parameters of both CAEN devices. Are recorded for each RPC channels (in most cases, a channel corresponds to an RPC gap):

- the effective voltage, HV_{eff}, set by the webDCS using the PT correction on the CAEN power supply,
- the applied voltage, HV_{app}, monitored by the CAEN power supply, and the statistical error related to the variations of this value through time to follow the variation of the environmental parameters defined as the RMS of the histogram divided by the square root of the number of recorded points,
- the monitored current, I_{mon}, monitored by the CAEN power supply, and the statistical error related to the variations of this value through time to follow the variation of the environmental parameters defined as the RMS of the histogram divided by the square root of the number of recorded points,
- the corresponding current density, J_{mon}, defined as the monitored current per unit area, $J_{mon} = I_{mon}/A$, where A is the active area of the corresponding gap,
- the ADC current, I_{ADC}, recorded through the CAEN ADC module that monitors the dark current in the gap itself. First of all, the resolution of such a module is better than that of CAEN power supplies and moreover, the current is not read-out through the HV supply line but directly at the chamber level giving the real current inside of the detector. The statistical error is defined as the RMS of the histogram distribution divided by the square root of the number of recorded points.

Once extracted through a loop over the element of GIF++ infrastructure via the C++ macro `GetCurrent()`, these parameters, organised in 9 columns per detector HV supply line, are written in the output CSV file `Offline-Current.csv`. The macro can be found in the file `Current.cc`.