Università
degli Studi
di Padova

Master Thesis in Physics of Data

Artificial Spill Generator at COMPASS.

Master Candidate

Lorenzo Borella

External Supervisor

Benjamin Moritz Veit

CERN Summer Student Supervisor

Supervisor

Prof. Andrea Triossi

University of Padova

Academic Year
2022/2023

Ai miei genitori

Abstract

The Artificial Spill Generator firmware for control, monitor and generate accelerator timing signals, has been developed for the DAQ system of CERN SPS M2 beamline experiments COMPASS and AMBER [3], within the frame of the Summer Student Program. In this work, COMPASS experimental context is described, reporting its field of research, the main purposes of its creation and the architecture of its spectrometer setup. A more detailed presentation of its Trigger and DAQ systems is also produced, providing a description of the bigger architecture in which the Artificial Spill Generator was firstly devised and eventually deployed. The structure and behaviour of the M2 beam line of CERN SPS exploited by COMPASS is explained, providing links with the functioning of the FPGA-based continuously running DAQ currently used in the experiment. Moreover, the hardware and software monitoring tools of the DAQ are presented, making comments on how they interact with the Artificial Spill Generator. Eventually, the logic and the behaviour of the firmware are reported in detail, explaining the different tasks and measurements associated with such module. After having passed all the required tests, the Artificial Spill Generator firmware has been programmed into an FPGA board, which is currently still implemented in COMPASS and AMBER DAQ systems, improving their acquisition performances.

Contents

1
Introduction

The following work describes the context and the stages of the development of the Artificial Spill Generator firmware. The project has been brought on in the experimental context of the Common Muon and Proton Apparatus for Structure and Spectroscopy (COMPASS) [23], within the frame of the Summer Student Program at the European Council for Nuclear Research (CERN). The content of this piece is divided into three main chapters, where the implementation of the Artificial Spill Generator firmware is explained with a deductive approach.

First of all, the experimental context of COMPASS is described, summarizing its main research purposes and the achievements gathered during its 25 years of work. The pillar topics of its broad physics plan are reported, touching several subnuclear structure and spectroscopy topics. Together with a brief chronology of the measurements performed, also the evolution of the experimental setup is described in detail. Moreover, the various components of the apparatus are reported, highlighting their main features and how they are related to the theory that justifies the experiment.

Afterwards, the Trigger and DAQ systems are described. In this way, getting more into the specifics of the actual data flow of the experiment, the context and the motivations behind the function of the Artificial Spill Generator become clearer. The functioning of the Trigger and its components are analyzed and the original architecture of the DAQ is explained, in order to provide a general description of how the data are selected, processed and analyzed, from frontend electronics to the final storage on tape. Moreover the continuously running FPGA-based DAQ system currently deployed at COMPASS is delineated, in order to picture the context in which the Artificial Spill Generator has been deployed and the reasons behind its implementation.

Eventually, the logic of the firmware is reported. The circuit is described in an analytic way, treating separately all of its components. The behaviours of the implemented entities are represented by several block and wave diagrams that help the reader to properly understand the timing and logic features of the modules. The firmware can be divided into two main cores, the Generator of the artificial signals and the Manager: these are described separately, paying particular attention to their different purposes and features. The behaviour of the whole circuit is in the end reported by connecting said entities and explaining how they need to interact with each other. Fundamental measurements on the beam parameters, as well as several safety features, are also performed by the module; the necessity of their implementation and their insertion in the logic are explained in detail. In the end, the simulations and tests performed on the Artificial Spill Generator firmware are reported, explaining the tools and techniques applied in order to verify the proper functioning of the module.

The actual implementation, planning and design of the Artificial Spill Generator have been performed by the author of this piece. Once verified its effective working, the module has been inserted into COMPASS DAQ pipeline. It contributed with a increase in efficiency of data taking during the 2022 COMPASS transversity run. It has been deployed for COMPASS collaboration during the last phase of data taking at the M2 beam line, during 2022 run 3, but it is currently performing its work also within the context of AMBER. It hopefully will continue also for the upcoming years of the collaboration.

2

The Experiment

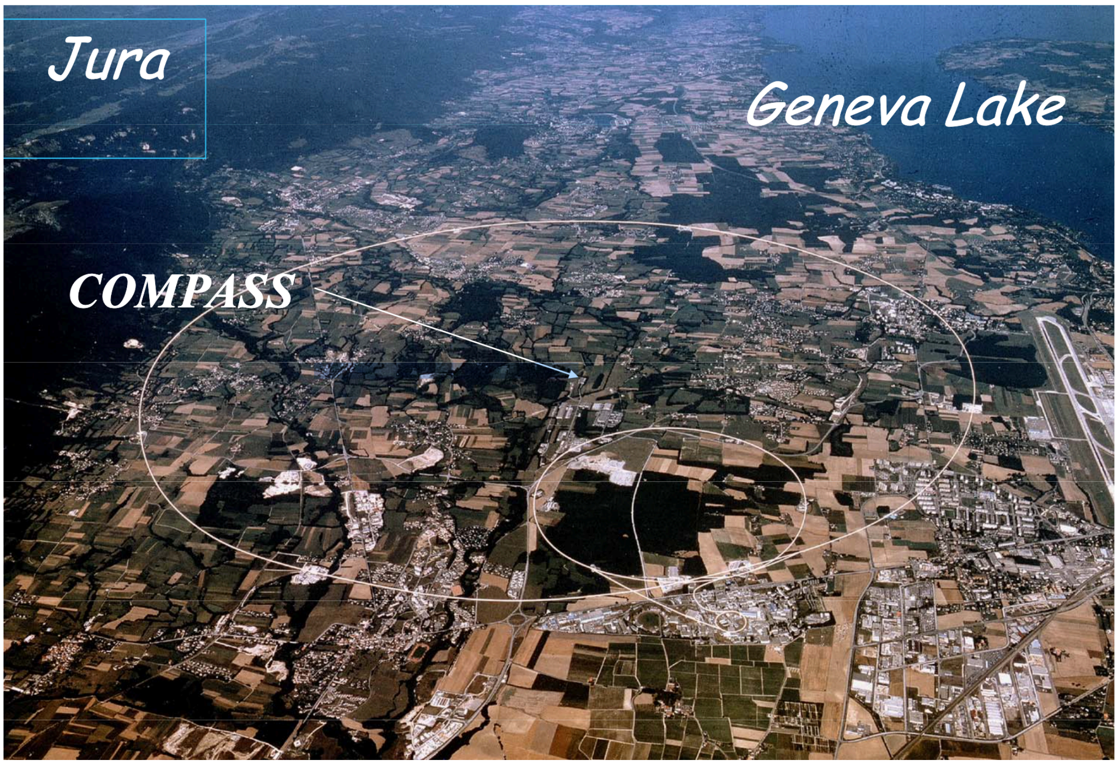


Figure 1: Bird's eye view of CERN's accelerator complex: COMPASS is located in the Prevessin site, around the center of the Large Hadron Collider (LHC) (bigger circle). It receives the particles through the M2 beam line, as they are accelerated by the Super Proton Synchrotrone (SPS) (smaller circle).

COMPASS is a fixed target experiment located in the North Area of CERN (Fig.1). Its last data taking goes back to 2022, with a rich research program for the investigation of subnuclear structure and spectroscopy. It is currently in its last analysis phase, but the new Apparatus for Meson and Baryon Experimental Research (AMBER) [2] collaboration took over the data taking and the beam line, guaranteeing many other years of work to this advantageous collaboration.

Its long and great history effectively begins in 1995 with the Letter of Intent (LoI) of two different experiments, the Hadron Muon Collaboration (HMC) and the CHarm Experiment with Omni-Purpose Setup (CHEOPS). At that time, both collaborations were proposing to build a new fixed target experiment in the Prevessin site, exploiting the M2 beamline with particles accelerated by the SPS. Even though their LoIs were produced independently and justified by different research purposes, they were submitted to the CERN Scientific Committee in the same year. As a consequence, thanks to their compatible physics programs and similar experimental requirements, the two collaborations were merged, resulting in a common environment able to satisfy the demands of both. COMPASS was born from this union, with its first configuration being proposed to CERN's commission in 1996 and approved the following year.

The experimental apparatus was effectively delivered in 2001, after few years of building and installation. In 2002, just after a year-long technical run, data taking began with COMPASS I phase and its first experimental configuration. The measurements exploiting the μ beam scattering on polarized proton and deuteron targets lasted from 2002 to 2011, with some interruptions in between. Apart from a technical shutdown in 2005, the years 2008 and 2009 were instead devoted to the hadron spectroscopy program, using $\pi^{\pm}$ and p beams interacting with liquid hydrogen and nuclear targets [13].

The second phase of the experiment (COMPASS II) was approved in 2010, even though the first Primakoff and Deep Virtual Compton Scattering (DVCS) run began in 2012. From 2015 to 2018 the Drell-Yan (DY) program was instead carried on, with $\pi^{\pm}$ scattering on a p target. During 2016 and 2017 DVCS, Hard Exclusive Meson Production (HEMP) and Semi-Inclusive Deep Inelastic Scattering (SIDIS) measurements were also brought on. Just after long shutdown 2, the data taking continued in 2021 and 2022 with SIDIS measurements off a transversely polarized target [13].

Coming to more recent times, a new LoI and proposal for further measurements on the M2 beam line were submitted 2019. The experiment will in fact continue its work under the name of AMBER, with a wider and renewed physics program, expecting the first measurements to be performed in May 2023 [7]. In 2022 COMPASS collaboration celebrated its 25th anniversary, making it one of the longer lasting experiments of CERN. Its technical versatility and the important physics results that it delivered to the scientific community contributed building its historical resonance. The future AMBER is of course expected to add further value to the collaboration: its research program aims at providing precise measurements of the proton radius using elastic $\mu - p$ scattering, investigating DY processes with conventional hadron beams and eventually measuring the antiproton production cross section for dark matter search.

In the following pages, the main historical steps of COMPASS collaborations are presented. A general description of HMC and CHEOPS research plans and experimental requirements is reported; their differences and similarities are highlighted, picturing the frame in which COMPASS was first devised and built. Afterwards, the different phases of the experiment are presented, paying particular attention to experimental setups used and the physical reasons behind their specific implementations. The first phase of the experiment (COMPASS I) is described in more detail with respect to the recent COMPASS II phase. Due to the huge quantity of available information, a selection of the main topics and measurements performed during COMPASS history had to be made. For this reason, a larger representation has been given to the measurements of the muon beam, instead than those belonging to the hadron program research. Eventually the studies of COMPASS II are just briefly reported, due to the much greater complexity of the physics they treat and in order not to digress too far from the main purpose of this work.

2.1 HMC and CHEOPS

HMC originally wanted to exploit the SPS M2 beamline with polarized $\mu^{\pm}$ scattering on a solid state polarized target, with a physics program which touched several open problems of experimental physics at that time. They wanted to perform a high precision measurement of the gluon polarization $\Delta G/G$ by studying open charm leptoproduction events. Moreover, they wanted to gather data on the quark longitudinal spin distribution functions and their integrals, produce high precision measurements on $\Lambda/\overline{\Lambda}$ polarizations and eventually also investigate the quark transverse spin distribution functions [22]. The fundamental requirements that had to be satisfied in order to obtain the desired precisions were a large detector acceptance of around ±200 mrad, a high beam intensity and a precise hadron identification. On the beam side, this practically corresponded to reaching a flux of $2x10^{8}\mu/\text{spill}$[1], with energies from 90 to 200 GeV. For the hadron classification π, K and p had to be correctly separated in the range $10-150$ GeV/c and π^{0} also had to be detected up to 150 GeV/c. Two Ring Imaging Cherenkov Detector (RICH) and lead glass Electromagnetic Calorimeter (ECAL) were the proposed tools to gather such optimal particle identification (see Fig.2) [22].

On the other hand, CHEOPS was exclusively going to concentrate on charm physics, setting as main goals the observation and study of charmed and doubly charmed baryons semi-leptonic decays, together with an abundant research program for exotic hadrons, glueballs and hybrids. For this kind of research, the usage of a large variety of beam projectiles ($\pi^{\pm}$, p, K, heavy ions etc.) and energies was required, possibly reaching the intensity of $5x10^{7}$ particles/spill [21]. The main expected challenge to perform the cited measurements was to correctly reconstruct charmed baryons. The huge backgrounds due to the missing neutrinos were the main cause of systematic errors, therefore the associated charmed meson tagging was going to be fundamental for a proper event

[1]As it is typical for circular accelerators, the particles are delivered in bunches called spills. In Sec. 3.3.1 a proper explanation of the beam delivery at COMPASS experimental hall is reported.

identification. From the experimental point of view, this meant that CHEOPS as well was in need of a large acceptance detector, possibly also extending to the backward emisphere. The proposed experiment was in need of LHC-type detectors in order to fulfill the requirements on rate and radiation stability and every component had to be mounted on rails, in order to guarantee a sufficient setup's flexibility [21].

HMC and CHEOPS had many other common experimental requirements, and also the proposed physics researches were for some aspects compatible and could have been brought on parallely. The decision to merge the two experiments seemed the best thing to do and the creation of a common apparatus that could satisfy all their needs represented a smart and profitable investment. A large geometrical acceptance, together with a good light hadron identification, were the key requirements for both HMC and CHEOPS. Also the possibility to use different projectiles and the need of a precise beam tracking system were other necessary aspects that were requested by the two different physics program. Eventually, very fast frontend electronics and LHC-type detectors were an indispensable part of the proposed experimental setups.

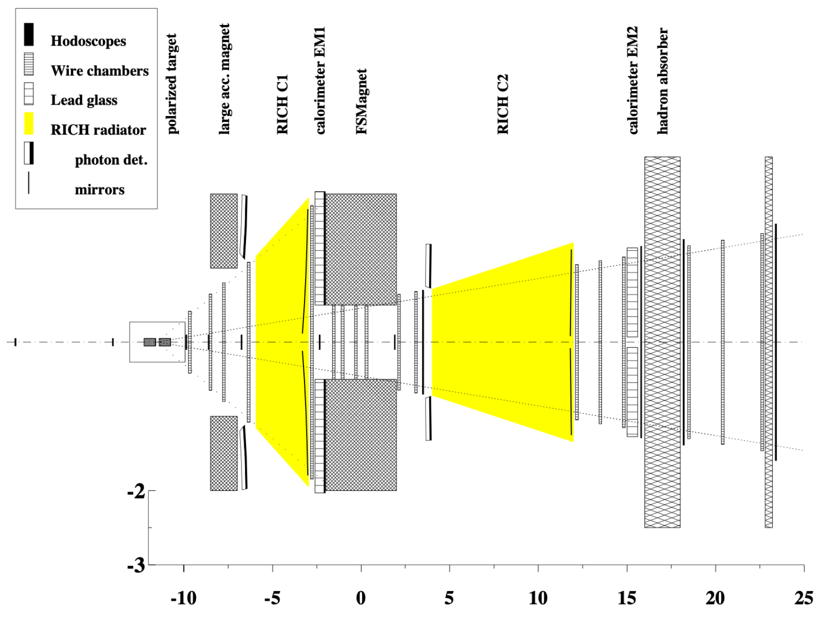


Figure 2: Top view of HMC's detector as first proposed in 1995 [22].

2.2 COMPASS I

The combination of HMC and CHEOPS research plans resulted in the creation of a unique experiment, focused on hadron structure and spectroscopy studies, with the aim of investigating the most hidden aspects of non-perturbative Quantum Chromodynamics (QCD).

Following the intents of CHEOPS collaboration, one of the main purposes of COMPASS was to collect high statistics samples of charmed particles. The hadron beam program was defined with this purpose, hoping to exploit particle momenta up to 300 GeV/c and to explore the nature of charmed/doubly charmed baryons and their semileptonic decays. At the same time, the investigation of gluon polarization $\Delta G/G$ suggested by HMC represented an extremely interesting topic and it was inserted as well into COMPASS research plan, expecting to perform cross-section asymmetries measurements exploiting open charm lepto-production events as workhorse process. In addition to the previous objectives, COMPASS also proposed to measure transverse spin distribution functions and fracture functions, considering an overall 4:1 proportion between longitudinal and transverse target polarization periods.

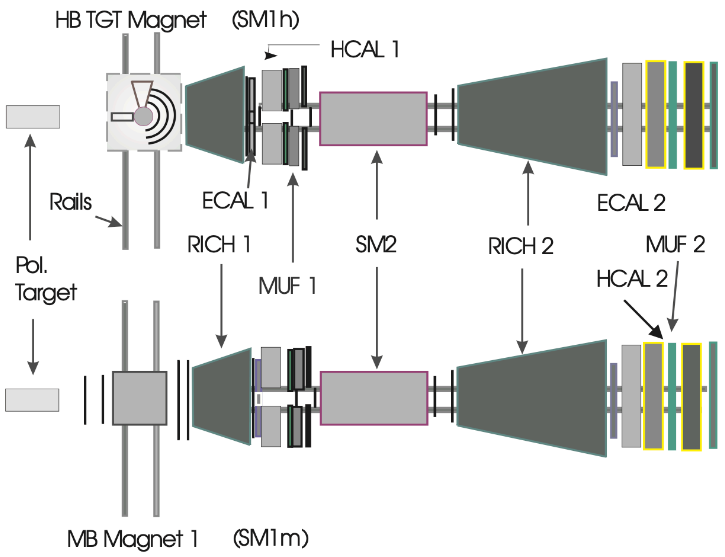


Figure 3: COMPASS proposed apparatus for the hadron (top) and muon (bottom) beam programs [27].

Specific measurements require optimal experimental setups and COMPASS was designed to satisfy the whole spectrum of the necessities defined by its wide physics program. The muon beam and hadron beam programs were going to need different and optimized architectures, starting from a shared common structure.

According to the first COMPASS proposal (see Fig.3), the spectrometer was going to be divided into two independent stages downstream from the polarized target, to analyze outgoing particles at different angles: the Large Angle Spectrometer (LAS) and Small Angle Spectrometer (SAS). Two dipole Spectrometer Magnets (SM 1 and 2) were going to be placed at the beginning of both LAS and SAS with a respective angular acceptance of ±180 mrad and ±30 mrad while two RICH1/2 detectors were planned to be placed in the LAS and SAS regions, with the aim of identify light hadrons[2]. Hadronic Calorimeter (HCAL) 1/2 were also going to be present at both stages of the spectrometer with the main purpose of gathering Trigger information, while ECAL1/2 were going to be used to identify fast particles, γ and π^0. Muon Wall (MW) 1/2 were planned to be located at the end of both LAS and SAS, always paired with Muon Filter (MF)1/2 (absorbers with sufficient thickness in order to make sure to stop all particles beside muons) [27]. Moreover, to allow to trigger on the scattered muons, a set of fast plastic scintillator hodosocopes was installed in both sections of the spectrometer, before and after the muon filters.

The structure reported in Fig.3 represents the original proposals for the architectures of the hadron and muon beam measurements. The experiment was eventually built with slightly different characteristics and components, which will be explained in detail in the following pages. The setup, together with the collaboration and the research purposes, has been updated several times during its years of work.

[2]Eventually RICH2 was never inserted in the experimental hall, since the cost benefit ratio was not going to be convenient.

In the next paragraphs, the theoretical aspects of its research are presented and associated to their specific experimental requirements, trying to deliver a general description of the evolution of the collaboration and of the experiment itself.

2.2.1 Muon Beam

In the 1996 COMPASS I proposal, the muon beam program was presented with a detailed description of the theory behind all the measurements that were going to be performed. The research plan was accompanied and justified by a series of Monte Carlo simulations, considering incident muonic beam of 100 GeV, $2x10^8\mu/$spill and 80% polarization. For the target, they considered ^6LiD with a 50% polarization for the D measurements and NH_3 with an 85% polarization for the p measurements. The luminosity during the first years of measurements was expected to be around $1.9fb^{-1}/$year, assuming a total SPS proton operation of 150 days/year, but it eventually turned out to be an underestimated value.

The periods for longitudinal and transverse target polarization were planned to be separated, due to technical times for changing polarization and to the different physics program they belonged to. The running time was planned to be split into 80% longitudinal, dedicating this time to the gluon polarization, longitudinal spin distribution functions and part of the Λ program measurements, and 20% transverse, with the aim of studying the transverse quark distributions and part of the Λ program [27].

The theoretical reasons behind the choice of measuring these quantities are extremely complex and they would need a much wider context for their proper explanation. A few of the performed measurements are reported in detail in this work in order to understand the importance of COMPASS work, but the majority of them are just mentioned, since their treatment is not part of the main purpose of this piece. In particular, the measurement of the Λ polarization and of the $\Lambda - \overline{\Lambda}$ spin correlations in the target fragmentation region can help discriminate between models with polarized strange quarks and gluons, delivering additional information for the description of the internal structure of nucleons.

For the same reason, COMPASS also planned to measure the transverse spin distribution functions $\Delta_T q(x) = q_\uparrow(x) - q_\downarrow(x)$ in semi-inclusive DIS on transversely polarized p and D targets, since the momentum distribution $q(x)$, the helicity distribution $\Delta q(x)$ and the transverse spin distribution $\Delta_T q(x)$ can completely specify the state of the quark inside the nucleon [27].

The $\Delta G/G$ and longitudinal spin distributions measurements represent the widest and most important part of the study that was firstly proposed by COMPASS and are explained more in detail in the following sections.

Gluon Polarization.

The best option for the measurement of gluon polarization at the CERN μ beam, with energies in the range 100 – 200 GeV, was expected to be the study of the longitudinal spin asymmetry of open charm leptoproduction events. The analysis of such events is based on the reconstruction of the D^0 mesons from their hadronic decay products, since heavy quarks are produced in leading order via the Photon Gluon Fusion (PGF) process (see Fig.4) [6].

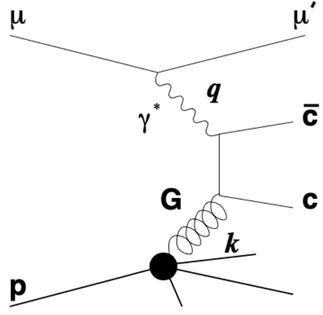


Figure 4: Photon Gluon Fusion Feynman diagram [27].

In particular, the experiment aimed at measuring the spin asymmetry A^{exp} for charm muonproduction, given by the number $N_{c\bar{c}}$ of charm events for anti-parallel and parallel μ and target longitudinal spin orientations. These quantities are related to the virtual photon asymmetry $A^{c\bar{c}}_{\gamma N}$ by means of Eq.1, where the parameters P_B, P_T, f, D represent respectively the beam and target polarization, the fraction of polarizable nucleons in the target and the depolarization of the virtual γ with respect to the μ [27].

$$A^{exp} = \frac{N_{c\bar{c}}^{\uparrow\downarrow} - N_{c\bar{c}}^{\uparrow\uparrow}}{N_{c\bar{c}}^{\uparrow\downarrow} + N_{c\bar{c}}^{\uparrow\uparrow}} = P_B \cdot P_T \cdot f \cdot D \cdot A_{\gamma N}^{c\bar{c}} \tag{1}$$

The asymmetry $A_{\gamma N}^{c\bar{c}}$ is eventually given by the ratio of the helicity dependent and helicity averaged cross sections for charm production $\Delta\sigma^{\gamma N \to c\bar{c}X}$ and $\sigma^{\gamma N \to c\bar{c}X}$, which can be expressed as the convolution of the elementary photon-gluon cross sections with the gluon distributions ΔG and G [27].

Since typically about 60% of charm quarks fragment into a D^0 meson, corresponding in average to $N^{D^0}/N^{c\bar{c}} = 1.23 D^0$ mesons per charm event, the event reconstruction was concentrated on identifying the presence of the D^0 meson. Unfortunately the detection of the decay vertex cannot be performed at COMPASS, due to the multiple scattering in the thick target, therefore the detection strategy fully relied on the combinatorial search for the hadronic decay products. The simplest decay of the D^0 meson is the two-body $D^0 \to K^- + \pi^+$, presenting a branching ratio of $(4.01 \pm 0.14)\%$. The fortunate characteristic of such decay is that in the D^0 c. m. frame, the decay particles emitted at large angles with respect to the D^0 line of flight have large transverse momenta. On the contrary, ordinary fragmentation into K or π prefers lower p_T and reproduce almost collinear decays [27].

As in can be seen in Fig.5, such characteristic resulted optimal for the detection and reconstruction of D^0 decay events. In particular, in the above plots of Fig.5 a simulation of the θ angle (w.r.t. D line of flight) vs momenta, respectively for K and π is reported. The images below instead represent the distribution of events as a function of $cos(\theta_K)$ and $z_D = E_D/\nu$ and the cuts applied to select the proper D^0 meson decay and to reject the background [27].

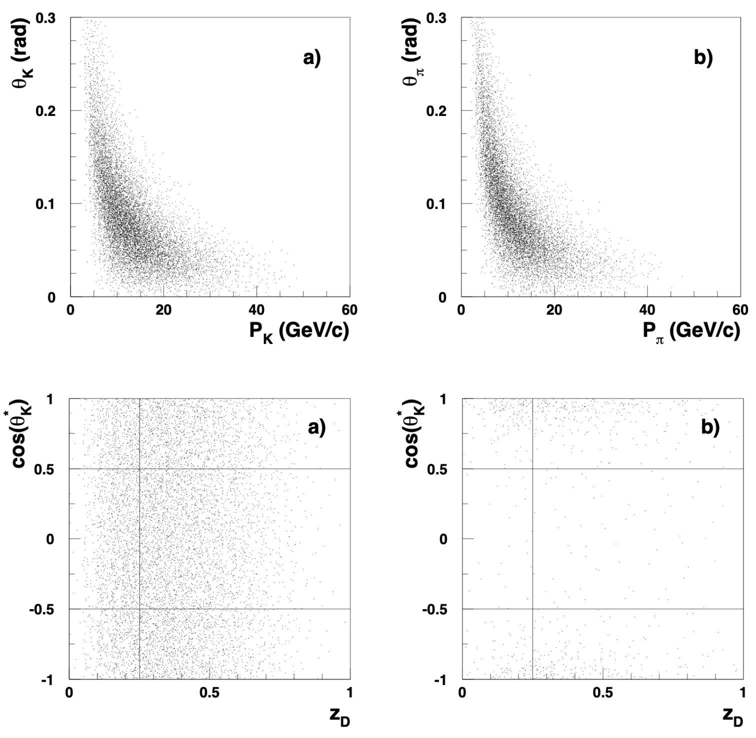


Figure 5: Top: Angle of $K(\pi)$ trajectory w.r.t. D^0 line of flight as a function of $K(\pi)$ momentum. Bottom: $\cos\theta_K$ as a function of the fraction of virtual photon energy carried by the D^0 meson $z_D = E_D/\nu$. The lines represent the cuts applied to reject the background [27].

Such analysis allowed COMPASS to gather high statistics of D^0 meson data, eventually useful for measuring the gluon polarization $\Delta G/G$ and many other quantities, also belonging to the hadron beam program.

Longitudinal Spin Distribution Functions.

With the aim of finding a possible explanation of the violation of the Ellis-Jaffe sum rule, several models were proposed at the time of COMPASS proposal. The most logic approach was to decompose the nucleon spin and try to determine the spin distribution functions of sea and valence quarks, in order to understand in which amount every component was contributing to the total spin of the nucleon. COMPASS proposed to achieve this by measurements of SIDIS events of polarized leptons on polarized proton and deuteron.

Parallel to the ΔG study, this analysis was performed by measuring the asymmetries for the different hadrons and the different targets (Eq.2). In this context the optimal particle identification, necessary for the flavour and valence and sea separations, represented a fundamental requirement of the experimental setup. For the same reason, the large angular acceptance provided by the spectrometer magnet 1 just downstream from the target allowed the additional possibility to perform an analysis of the azimuthal effects in asymmetries and the topology of hadron jets at high p_T.

$$A^h = \frac{1}{P_B P_T f D} \frac{N_h^{\uparrow\downarrow} - N_h^{\uparrow\uparrow}}{N_h^{\uparrow\downarrow} + N_h^{\uparrow\uparrow}} \tag{2}$$

2.2.2 Hadron Beam

The study of charmed hadrons covers a wide spectrum, from investigations of charm production itself, spectroscopy, decay studies up to search for rare processes. It can be divided into three mutually connected categories, semi-leptonic decays, lifetimes and non-leptonic decays, but only few examples of them are reported in the following pages.

Before COMPASS first proposal, the study of semileptonic decays of charmed baryons had been the domain of experiments at e^+e^- colliders, due to the high statistic of hadronic events containing charm quarks, but by exploring the D-meson sector also fixed target experiments had become very competitive. The plan of COMPASS collaboration was to obtain a handle on the inclusive semi-leptonic decays of different c-baryons using the rates of their exclusive decays and a sample of D-tagged events, already reconstructed for the purposes previously cited. Also the measuring of charmed hadrons lifetimes constituted a good testing ground for the understanding of the effects of the hadronic environment, especially in the intermediate region where perturbative and non-perturbative effects overlap. Of course such measures would also have contributed to a more precise knowledge of absolute charm's branching ratios [1].

The technical challenge associated with such measurements was to properly try and reconstruct the charmed baryons, due to the uncertainties carried by the missing neutrino. The consequent enhanced backgrounds could only be reduced with a good tag for the associated charmed meson, hence the necessity of building a spectrometer with large solid angle and acceptance.

One of the other COMPASS aims was to look for a proof of the existence of gluonic systems. In those years, QCD had already been consistently tested and stressed by many experiments, concluding with a confirmation of its viability. Nonetheless, the existence of new classes of non-$q\overline{q}$ mesons still awaited experimental confirmation. Theory predicted the existence of glueballs, objects composed entirely of valence gluons, together with many other hybrids states. Of course the major difficulty in the search for such entities arised from the proliferation of mesonic states with $q\overline{q}$ structure in the mass range in which states with constituent glue are expected. The spectroscopy of scalar and pseudoscalar states is particularly difficult, since higher-spin mesons are often produced more abundantly than states with J=0, therefore scalar contributions remained unrevealed. An additional research in this broad field looked necessary for COMPASS and the whole scientific community [27].

2.3 Experimental Apparatus

As already stated before, fixed-target experiments for the study of QCD in its non-perturbative regime require large luminosity and thus high data rate capability, an excellent particle identification and a wide angular acceptance. COMPASS was hence built to achieve these main design goals.

The basic layout of COMPASS experiment, as it was built at the beginning of the collaboration, can be divided into three main parts. The detectors upstream from the target represent the first section of the experiment, called "beam telescope", where measurements on the incoming beam particles are performed. The other two parts are placed downstream from the target (respectively LAS and SAS) and they extend for a total length of 50m. Each of the two spectrometer stations is built around an analyzing magnet, preceded and followed by trackers and completed by a hadron calorimeter and a muon filter station for high energy μ identification. A RICH detector for light hadron identification has also been inserted in the LAS. The majority of the above cited components were mounted on rails since the very beginning of the experiment, in order to be able to move them accordingly to the necessities of the different physical measurements.

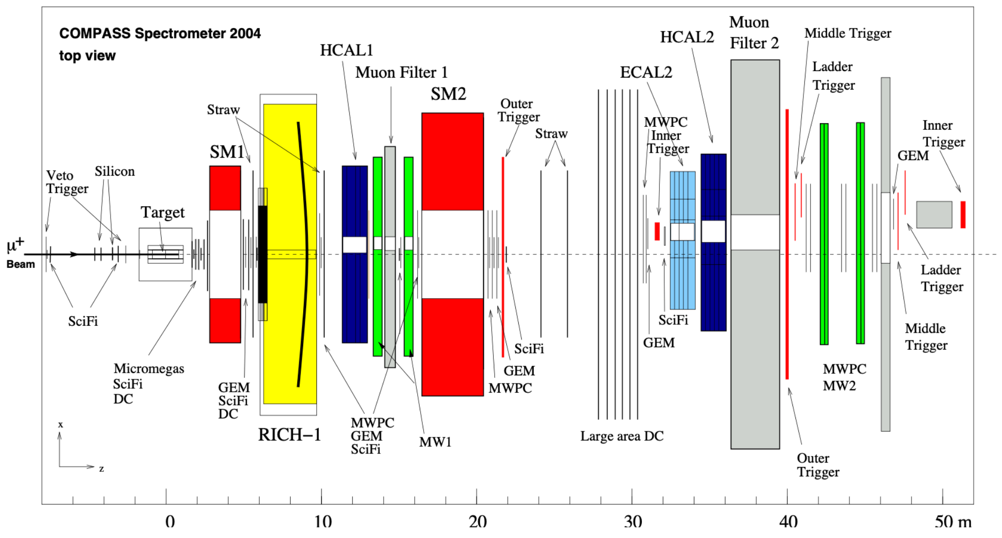


Figure 6: Top view of COMPASS muon setup according to 2004 configuration [11].

The first section of the experiment, the Beam Momentum Station, is located about 100 m upstream from the target and it is responsible for measuring the momentum of the incoming muon. It is provided with an analyzing magnet and six tracking stations Scintillating Fibers (SciFi) detectors. The precise track reconstruction of incoming muons is instead performed just upstram from the target (see Fig.6), where SciFi and silicon microstrip detectors are placed [11].

Located 4 m downstream from the target center, the LAS was built around the Spectrometer Magnet 1, which has been designed to deliver a deflection of 300 mrad for particles with momentum of 1 GeV/c. In order to cope with such bending power, the RICH detector was build with a large transverse dimension, making it capable of detecting and identifying charged hadrons with momenta in the range 3 – 50 GeV/c. The LAS is eventually completed by the HCAL1 and the Muon Filter 1, the former being fundamental for gathering the trigger information and the latter being responsible for muon identification (see Fig.6).

The last section of COMPASS spectrometer (SAS) detects particles at small angles and larger momenta, precisely ±30 mrad and $p > 5$ GeV/c. Together with the second spectrometer magnet, it includes electromagnetic and hadronic calorimeters and it is concluded by a thick muon filter, responsible for stopping outgoing hadrons. Just like HCAL1, HCAL2 is used in the trigger information, while ECAL2 mainly detects γ and π^0 [11].

Regarding the tracking detectors, different techniques and tools need to be used depending on the position in which they are placed. The particle flux per unit transverse surface varies by more than five orders of magnitude in different regions within the overall spectrometer acceptance. Along the beam and close to the target, detectors must combine a high particle rate capability with an excellent space resolution (up to a few MHz/channel and 100μm and better). Due to the already high particle flux, the amount of material along the beam path has to remain at a minimum, in order to minimize multiple scattering and secondary interactions. Moving radially away from the beam axis, the constraints on the particle rate and spacial resolution can be relaxed, even though larger areas need to be covered. The tracking detectors can be divided into three main groups: the Very Small Area Trackers (VSAT), the Small Area Trackers (SAT) and the Large Area Trackers (LAT). The first are characterized by high flux capabilities and excellent space time resolutions and they consist in eight SciFi stations and three silicon microstrip detectors (see Fig.6). The SAT are placed at beam distances greater than 2.5cm and they are made of three Micromesh Gaseous Structures (Micromegas) [16] and eleven Gas Electron Multiplier (GEM) stations [28], featuring high space resolution and minimum material budget. Eventually the LAT cover the largest region of the spectrometer, providing a good spatial resolution. Particles emerging at large angles are tracked by three Drift Chambers, while straw drift tubes are responsible for covering areas closer to the beam axis. From downstream the RICH counter to the end of the setup, the particles scattered at relatively small angles are detected by fourteen Multi Wire Proportional Chambers (MWPCs). The identification of scattered μ is performed by the two muon filters, which include an absorber layer, preceded and followed by tracker stations with moderate space resolution (Muon Walls). The absorber is fundamental for stopping incoming hadrons. The presence of a muon is confirmed only if compatible signals are retrieved in both trackers, upstream and downstream from the absorber [11].

2.4 COMPASS II

The proposal of the second phase of measurements at COMPASS experimental hall arrived in 2010, with a renewed physics research program and several necessary setup improvements. The proposal contained mainly studies of chiral perturbation theory, "unpolarized" Generalized Parton Distributions (GPDs) and Transverse Momentum Dependent (TMD) parton distributions. The new GPDs theoretical framework allowed the study of the nucleon from a unique point of view: while Parton Distribution Functions (PDFs) only represented the structure of the nucleon as a function of the nucleon momentum fraction carried by a parton of a certain species regardless of the electromagnetic form factor, the GPD was able to embody both information, such that they could be considered as momentum-dissected form factors. In a complementary approach, COMPASS was also planning to study the intrinsic effects of transverse parton momenta described by TMDs, which become visible in DY and SIDIS processes. The final goal of the experiment was indeed to build a more complete 3d-picture of the nucleon, therefore performing studies of nucleon tomography [12].

The concept of GPDs could also help in understanding the original problem that COMPASS decided to face at the time of its creation: how the nucleon 1/2 spin is shared between the contributions of intrinsic and orbital angular momenta of quarks of vairous flavours and gluons. Constraining quark GPDs experimentally, by measuring DVCS or Deep Virtual Meson Production (DVMP), would have represented a good way of constraining the quark components on the nucleon spin budget $\frac{1}{2} = \Sigma_{f=u,d,s} J^f + J^g$.

Parallely with the GPD program, high-statistics data of SIDIS events were to be recorded in order to extract at leading order $\alpha_S(LO)$ the unpolarized strange quark distribution function $s(x)$ as well as fragmentation functions describing how a quark fragments into hadrons $\mu p \to \mu h X$. Another fundamental aspect of COMPASS-II research plan was to concentrate on the transverse momentum of partons. This quantity is a central element for the understanding of the 3-dimensional structure of the nucleon and when it is taken into account, several new functions are required to properly describe the internal structure of the nucleon: intrinsic transverse momentum and transverse momentum naturally couple, therefore their resulting correlations need to be encoded into various parton distribution and fragmentation functions [4].

3

Trigger and DAQ

In every High Energy Physics experiment the amount of scattering events produced in the laboratory has to be the highest possible, in order to be able to eventually produce statistically significant results. Of course COMPASS as well had the precise purpose to produce a considerable amount of interactions, exploiting polarized muons and hadrons scattering on polarized solid and liquid state targets. In the end though, after a multi-level selection of information based on specific physical requirements, only a small percentage of the total produced events are effectively used for the research purposes described in the previous chapter. This process of filtering the overall amount of interactions is of course mandatory, in order to select only the events of interest and to reject the backgrounds. The data reduction happens at different stages in the pipeline of the trigger and data acquisition systems of the experiment.

In the following sections, the flow of data gathered by the experiment's detectors is reported, providing a general explanation of the filtering and selection techniques in use. In particular, the first level trigger system is described, paying particular attention to the experimental setup used and the physical reasons behind each hardware's implementation. The original architecture of the Data Acquisition System (DAQ) system is reported according to the state of the experiment in 2007, in order to be able to compare it to the updated and currently deployed FPGA based continuously running version (iFDAQ).

The Artifical Spill Generator project, which will be treated in detail in Chapter 4, was based on such latter implementation of the DAQ and it has been eventually inserted into its pipeline.

3.1 Muon trigger system

The trigger system itself has to serve several purposes. Firstly, it has to select events of interest in a high rate environment, then it has to provide precise event times for unambiguous association with incident beam muons. Eventually it is also responsible for generating strobe signals for several other detectors in the spectrometer.

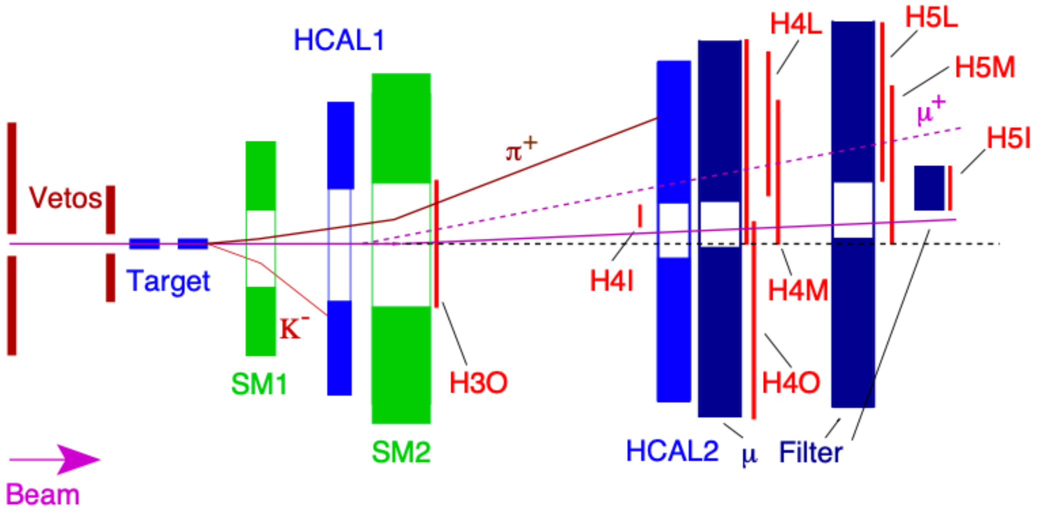


Figure 7: Detector components used in the Trigger information: veto hodoscopes (Sec.3.1.1), spectrometer magnets (green), hadronic calorimeters (light blue) and muon filters (dark blue).

COMPASS trigger system is reported in Fig.7. In view of the high rates in the central region, the hodoscope system is subdivided into four parts: the *inner* (H4I, H5I) and the *ladder* (H4L, H5L) systems covering the quasi-real photon events, the *middle* (H4M, H5M) system selecting quasi-real photon events as well as deep inelastic scattering events and the *outer* (H3O, H4O) system selecting muons up to $Q^2 \simeq 20GeV^2$. The *middle* trigger system uses two planes of the hodoscopes to detect muons scattered with angles between 4 and 12 mrad, by measuring the vertical projection θ_y of the scattering angle in the non-bending plane and checking its compatibility with the target position ("vertical target pointing trigger"). In addition, two other planes measure the horizontal projection θ_x and allow to perform a coarse energy cut.

Fig.7 also shows the concrete and iron absorbers used to identify the muon tracks. In the ladder, middle and outer systems, a 2.4 m thick concrete absorber is used for muon filtering, together with the material of the hadron calorimeter HCAL2 located in front of the first hodoscope. Due to the large element width in the outer system compared to the ladder and middle system, a large lever arm is needed, therefore the first hodoscope of the outer system, H3O, is installed directly behind the second spectrometer magnet. This hodoscope has a much larger central hole, covered by the other systems, to avoid excessive rates in the 7 cm wide strips.

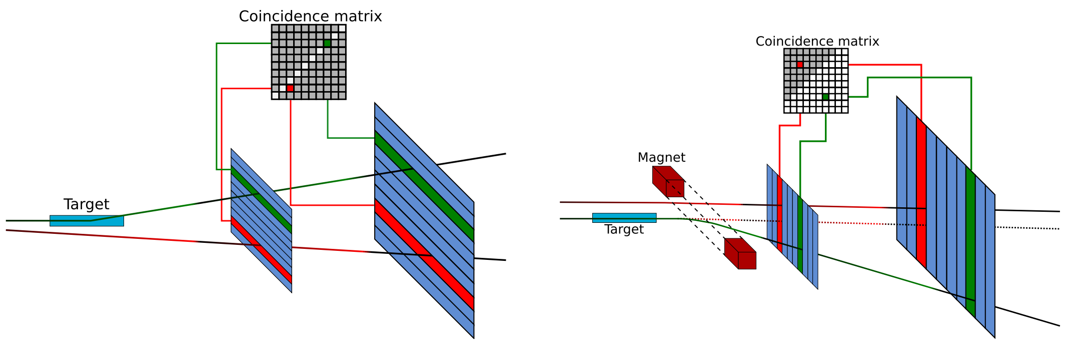


Figure 8: Left: Target pointing trigger. Right: Energy loss trigger.

The first trigger condition ("vertical target pointing") for the measurement of scattered muons is reported in the Left picture of Fig.8. It forms a geometrical trigger, in which a set of two hodoscopes with horizontal slabs before and after the muon absorbers measure the vertical coordinate of the scattered muon at two positions Z1 and Z2 in the magnet field-free region of the spectrometer. By requiring a certain combination of fired slabs in the coincidence matrix the origin of the particle track to the target location can be verified [32].

Due to the nature of the interactions analyzed at COMPASS for the muon beam program, its trigger system has to detect muon scattering events on target nucleons, with a relative energy loss exceeding a selectable value y_{min} and happening at relatively small scattering angles (see Right picture in Fig.8). Moreover, imposing an additional condition of a minimum energy deposit in an hadronic calorimeter, it is possible to reject background events: like scattering on electrons, elastic and quasi-elastic radiative events as well as events from beam halo tracks. The trigger can in fact be considered as a tagger for quasi-real photoproduction events (Fig.7) [17].

A second trigger condition is sometimes applied, depending on the physics program, by measuring the muon energy loss, exploiting the muon track direction behind the two magnets in the bending plane of the dipole spectrometer. Two vertical hodoscopes planes at distances z_1 and z_2 from the target are responsible for measuring the horizontal position x_1 and x_2 of the deflected track. For an isolated muon track, of course being aware of the magnets transverse kick, this information is sufficient to determine the horizontal component of the scattering angle θ_x, its momentum p' and its energy y. In conclusion, the trigger for scattered muons with a momentum below a preselected threshold p'_{max} is realized by requiring a coincidence of elements x_1 and x_2 in the hodoscopes positioned at z_1 and z_2, fulfilling the condition in Eq.3.

$$\frac{\Delta p_x}{p'} z_m = \frac{x_1 \cdot z_2 - x_2 \cdot z_1}{z_2 - z_1} \tag{3}$$

As it can be seen in Fig.9, the position information gathered from the hodoscopes H4 and H5 is used to build a coincidence matrix, in which only certain elements cause the generation of the trigger signal. The additional condition of the energy deposit in an hadronic calorimeter is eventually convoluted with the coincidence matrix signal, causing the proper selection of the scattered muon and the rejection of the muon which did not interact within the target cells [25].

The selection of muon tracks requires an absorber in front of one of the two hodoscopes, in order to reject hadron and electron tracks: for this purpose an absorber of 1.6 m of iron, corresponding to 91 radiation lengths, was chosen. The effects of multiple scattering on the measured track position are minimized if the absorber is placed directly in front of the second hodoscope and all detetctors between the two hodoscopes have a hole in the center that matches the projected size of the trigger hodoscopes [31].

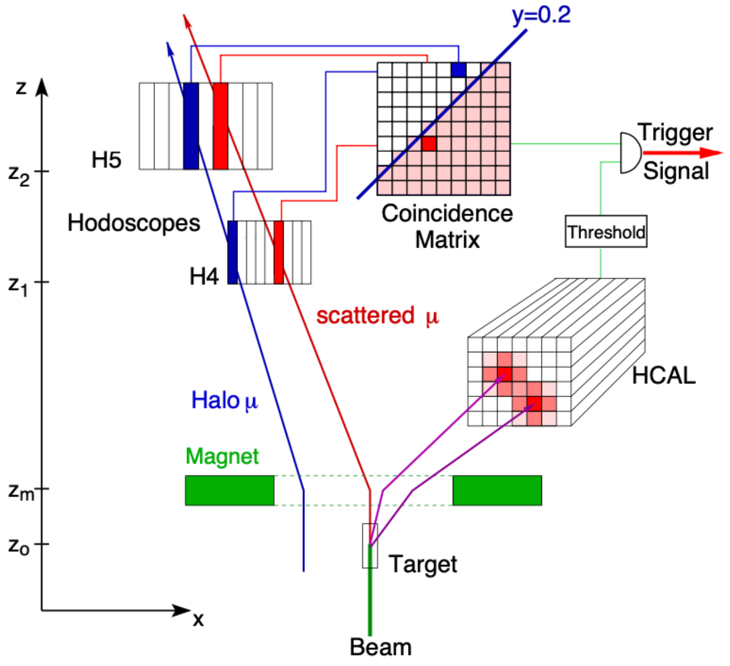


Figure 9: Muon trigger functioning: scattered muon causes the generation of trigger signal (red), halo muon is rejected (blue). The energy deposit condition in an HCAL contributes to the generation of the trigger signal [31].

3.1.1 Veto system

A trigger based only on a coincidence between two hodoscopes planes would result in a rate of order 10^6 per spill, much too high for the DAQ system. Most of these coincidences are caused by muons not interacting in the target since, as it is clear from Fig.10, the muon beam is not always perfectly focused. The particles that do not satisfy certain momentum and position requirements when hitting the target cells are referred to as "Halo Muons" and they need to be discarded as they do not respect the specifics of the events researched by COMPASS [26].

The so-called Veto Trigger System was introduced with the purpose of eliminating such backgrund events and it simply consists in hodoscopes upstream the target, leaving the central region around the beam uncovered. Thanks to the information gathered by such detectors, a large fraction of the unwanted trigger signals can be eliminated by demanding the absence of a signal in the veto system, together with a coincidence in two hodoscope planes.

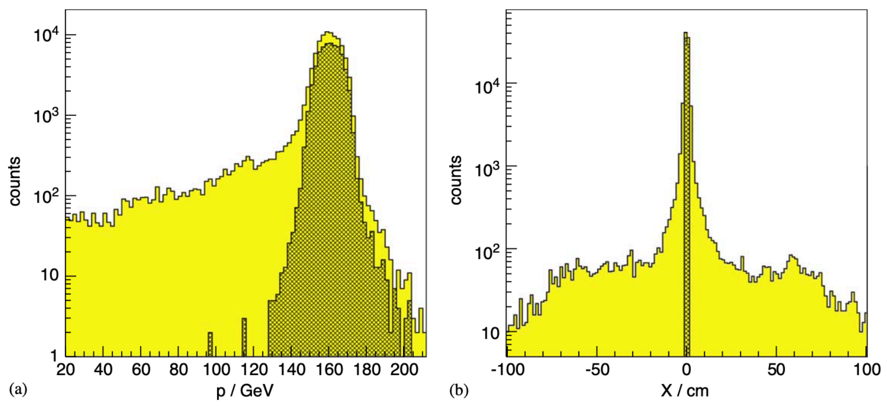


Figure 10: Momentum distribution (left) and horizontal profile at the target center for incoming particles obtained with a random trigger (right). The shaded areas correspond to particles passing through both target cells [31].

The first hodoscope of the veto system, $Veto_1$ is placed at $z_{v_1} = -800cm$ and the second smaller one is located at $z_{v_2} = -300cm$; they are placed at different positions along the beam in order to veto divergent particles passing through the hole in one of them, as it can be seen in Fig.11. The application of $Veto_1$ reduces the rate of triggered signals to 9%, while $Veto_2$ alone reduces it to 46%: the combination of the two instead leads to the 4% of the rate without any vetoed signal. Moreover, in order to be able to veto tracks passing through both holes in $Veto_1$ and $Veto_2$, a further $Veto_{bl}$ was placed at $z = -2000cm$ upstream from the target and its inclusion in the trigger condition leads to a further reduction of 1.4% [31].

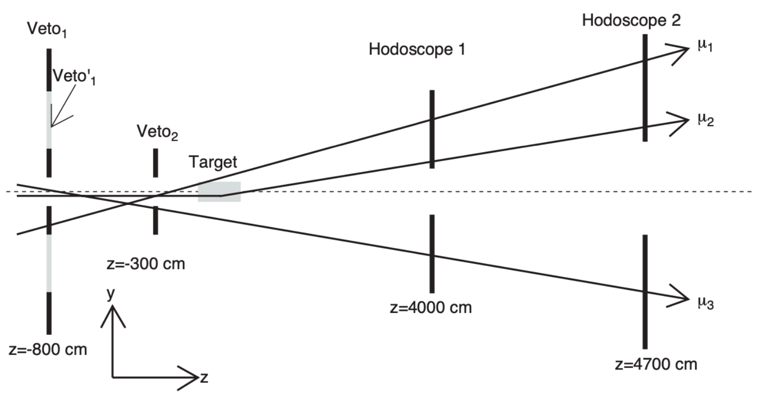


Figure 11: Examples of vetoed (μ_1, μ_3) and not vetoed (μ_2) muon tracks [31].

The trigger conditions described above are the first to be applied to the scattering events happening in COMPASS experimental hall, consistently reducing the total amount of data gathered by the detectors. Nevertheless, additional conditions are of course applied in secondary phases of the data acquisition process. In the following pages, the orignal and current architectures of the DAQ system are presented, providing a general description of the flow of data from frontend electronics to the complete event storage.

3.2 Read-out Electronics and Data Acquisition

The presented DAQ architecture reflects the state of the experiment in 2007, therefore is not fully descriptive of the current functioning of the system. As for many other parts of the spectrometer, the DAQ has been updated several times during the long work life of COMPASS, and even though several components and techniques have been improved, the approach used in the data acquisition process remains the same. One of the fundamental features of the readout electronics and DAQ required since the very first COMPASS proposal, was that they had to serve different physics programs with different detector setups. The amount of channels and the rate of the first level trigger made the design of the system very challenging, requiring LHC type frontend electronics. In order to cope with the high particle fluxes of $2 \cdot 10^8$ muons per spill of 4.8 s, a typical event size of 35 kB, trigger rates of about 10 kHz for the muon beam and a design value of 100 kHz triggers for the hadron beam, a pipelined and nearly dead-time free readout scheme had been adopted [30].

The analogue signals from the various detectors were preamplified, discriminated and digitized in most cases close to the detectors, since the gain was twofold: no loss of signal quality (only short cables required) and cost for cables considerably reduced. The synchronization of the digitizing and readout units was performed by the Trigger Control System (TCS) [19], but its main function was and still is to distribute trigger, time reference and event identification information to the readout driver modules and to generate the strobes for gating some of the Analog-to-Digital Converters.

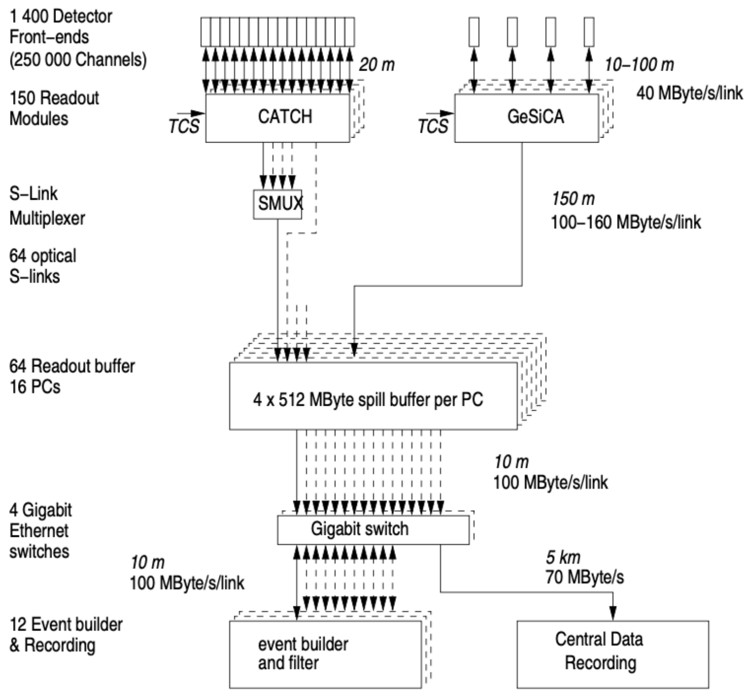


Figure 12: Original COMPASS DAQ system according to 2007 architecture [11].

Upon arrival of the trigger signal, the data were transferred via fast links to CATH [29] and GeSiCa modules, which were also responsible for distributing the trigger signals to the front-ends and for initializing them during the system startup (Fig.12). These readout modules combined the data and transmitted the subevents at a maximum throughput of 160 MB/s to 512 MB readout spill buffer cards. Such electronic components had been developed specifically for COMPASS, while the final event building system was based on high performance PCs and standard Gigabit Ethernet components. The event building took place during both the on and off spill phases, resulting in an average data rate of 70 MB/s. The final events were eventually recorded on tape remotely at CERN's central data recording facility, located in the computer center.

In order to match the high rate capabilities of the trigger system and readout electronics, the data acquisition system had to rely on buffering and parallelism. The regrouping of the data streams was supported by the spill buffer cards and the DAQ computers which formed the event building network. Due to impurities in the trigger system, the output streams contained events which were still useless for the analysis, therefore to save bandwidth, storage space and reconstruction time, the output streams were filtered by an Online Filter (second level trigger). This configuration allowed a simultaneous reading from the detectors and writing to the event building network without overhead or bandwidth losses. The online filter increased the purity of the triggers and allowed for a cost effective reduction of the amount of tapes needed for recording. For the physics program with hadron beam, the online filter was required to reduce both the bandwidth needed for transferring the data to the computer centre as well as the bandwidth needed to record the data on the tape drives. In addition, one could also profit from a reduced CPU time for the reconstruction of the data. At a trigger rate of 10 kHz and with 12 event builders sharing the load, the allowed average decision making time was 4 ms per event. For the two physics programs, two filter algorithms had been developed. In the muon program the presence of a reconstructed beam track was required: the silicon microstrip and scintillating fibre detectors upstream of the target, together with the Beam Momentum Station, had to have recorded a sufficient number of hits from the beam particle. The algorithms were based on the coincidence between the trigger time and the times measured by the aforementioned detectors, respecting their different time resolutions and allowing for the redundancy of the detector systems [11].

3.3 iFDAQ

Due to the increased requirements in 2014 in terms of data acquisition scalability, reliability and data throughput, the COMPASS experiment developed the intelligent FPGA-based Data Acquisition System (iFDAQ), using a novel approach to the event building network. In contrast to traditional event builders which are based on distributed online computers interconnected via an Ethernet Gigabit network, the event building task is now completely executed in hardware by FPGA boards and it exploits, for the configuration of the modules, the IPBUS communication protocol (see Sec.3.3.4) developed at CERN [20]. Such improved stability opened up the possibility to keep the system continuously running without interruption of data flow, reducing time consuming synchronization phases at each start and stop of a run.

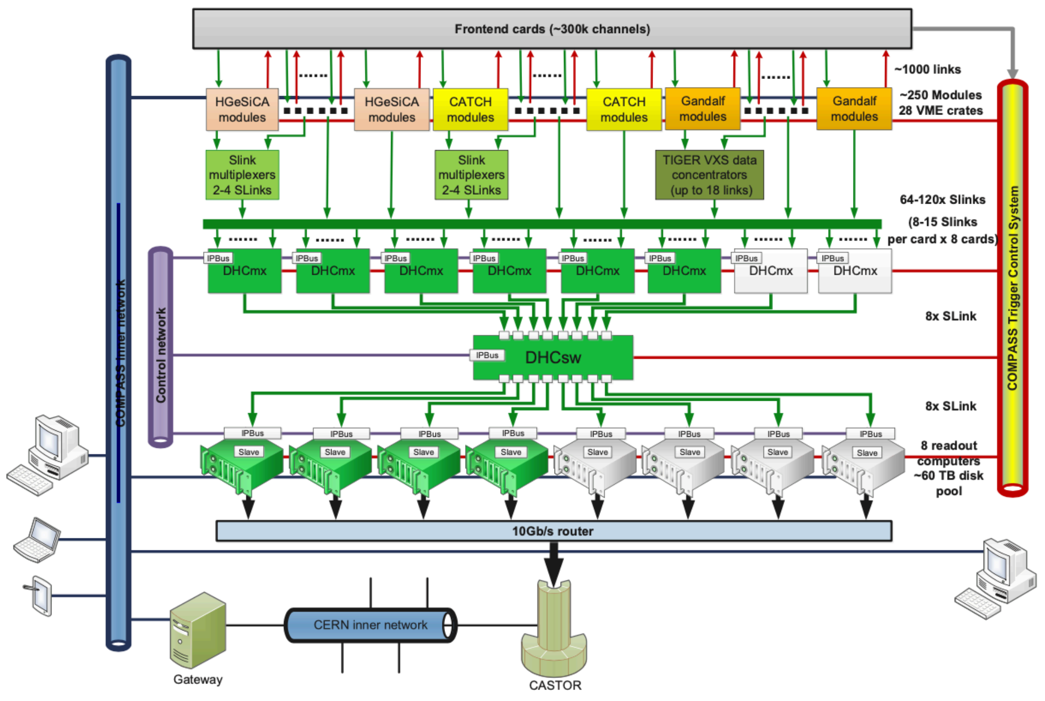

Frontend cards (~300k channels)
~1000 links
HGeSiCA modules
HGeSiCA modules
CATCH modules
CATCH modules
Gandalf modules
Gandalf modules
~250 Modules 28 VME crates
Slink multiplexers 2-4 SLinks
Slink multiplexers 2-4 SLinks
TIGER VXS data concentrators (up to 18 links)
64-120x Slinks
(8-15 Slinks per card x 8 cards)
IPBus
DHCmx
DHCsw
Slave
8x SLink
8x SLink
8 readout computers ~60 TB disk pool
COMPASS Trigger Control System
COMPASS inner network
Control network
10Gb/s router
CERN inner network
Gateway
CASTOR

Figure 13: Topology of the iFDAQ system according to the 2018 setup [24].

The current COMPASS data acquisition system (iFDAQ) has been commissioned in 2014, replacing the previous DAQ, which had been in use since the very first conception of the experiment. In the COMPASS spectrometer, there are in total more than 300,000 detector channels of which the iFDAQ collects hit, time, and, for a subset of detectors, also amplitude information. The data streams emerging from the detector frontends are multiplexed and sent to the FPGA switch, which receives all subevent information and assembles a full event from these fragments. The recombination and merging of event information during the event building process follows a strict pattern whose consistency is verified at each stage. Finally, the data are read out by four readout engine computers and stored on hard drives (see Fig.13) [24].

Since beam time is highly valuable and it is usually provided 24/7 for most of the calendar year, high reliability of the DAQ system is of major importance to High Energy Physics experiments. The overall time period when no physics data can be taken is called the iFDAQ downtime, while the iFDAQ uptime denotes the overall time period when the iFDAQ is stable and ready for a proper data taking. During the data taking going from 2015 to 2017, there were three main sources of instabilities leading to time periods when no physics data could be taken (see Fig.14). The first one WAS a memory access error caused by scrambled data being transferred to the RAM of the readout engines. It was the most time consuming failure, since it requires to reboot all readout engine computers and the recovery procedure takes approximately 10 minutes on average. The second one was an unrecoverable loss of synchronization in the hardware event builder leading to a safe stop of a run. The safe stop of a run might be considered as one of the intelligent elements of the iFDAQ since a safe stop prevents more serious problems which would lead to the higher downtime. The third source of the downtime is based on unknown software crashes being not fully understood. Luckily in the meantime COMPASS has gained a lot more data acquisition stability, currently reaching a 99% and higher DAQ uptime [24].

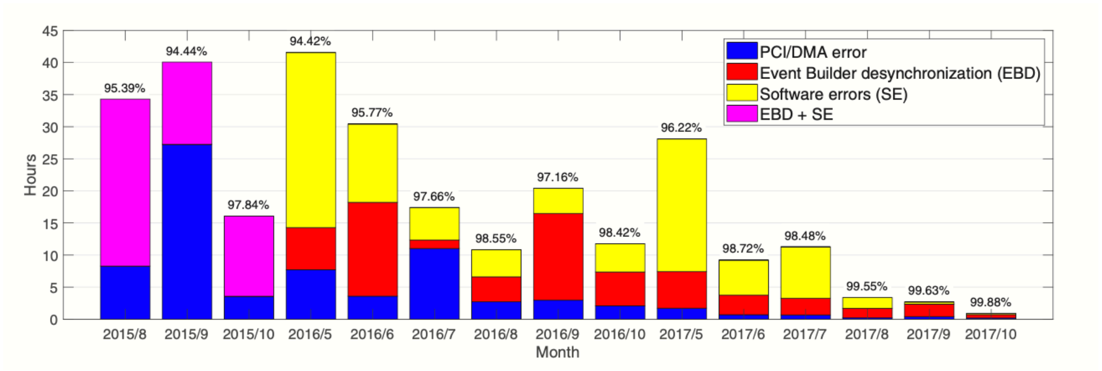


Figure 14: Absolute downtime of the iFDAQ per month and corresponding relative uptime [24].

Another significant contribution to the loss of beam time originates from the time that is needed to initiate a synchronized data flow through the event builder. Establishing synchronous processing of data by all involved hardware nodes is achieved by distributing trigger and spill cycle information to all nodes via the Trigger Control System (TCS) and applying reset commands and timeouts. To synchronize the processes correctly, the iFDAQ needs to take advantage of some proper timing mechanism. The best choice to achieve this is to exploit the signals generated by SPS, which are naturally synchronized with the proper phases of the spills (see Fig.16 in Chapter 4).

3.3.1 Spill Synchronization

The beam for the experiments at the CERN M2 beam line is provided by the SPS, which delivers the particle flux in bursts called spills. Before being able to deliver a particle spill to COMPASS, the SPS has to be filled with proton bunches, the bunches have to be accelerated to the desired energy, and the particle load inside the SPS has to be debunched. Only when the particles circulate homogeneously distributed in the SPS, a spill of stable particle intensity can be delivered. After one filling is completely extracted from the machine, the cycle starts over [24].

Fig.15 shows the proton intensity in the SPS during two cycles, which together form an exemplary SPS super-cycle. For COMPASS, the particles in the SPS are slowly extracted over the course of 4.8 s. After the spill, there are at least 5 s without spill, since filling the SPS with two injections from the Proton Synchrotrone (PS) takes approximately 2 s, and ramping up the energy and debunching (flat top region) takes another 3 s. However, the off-spill period can take significantly longer, depending on the SPS super-cycle.

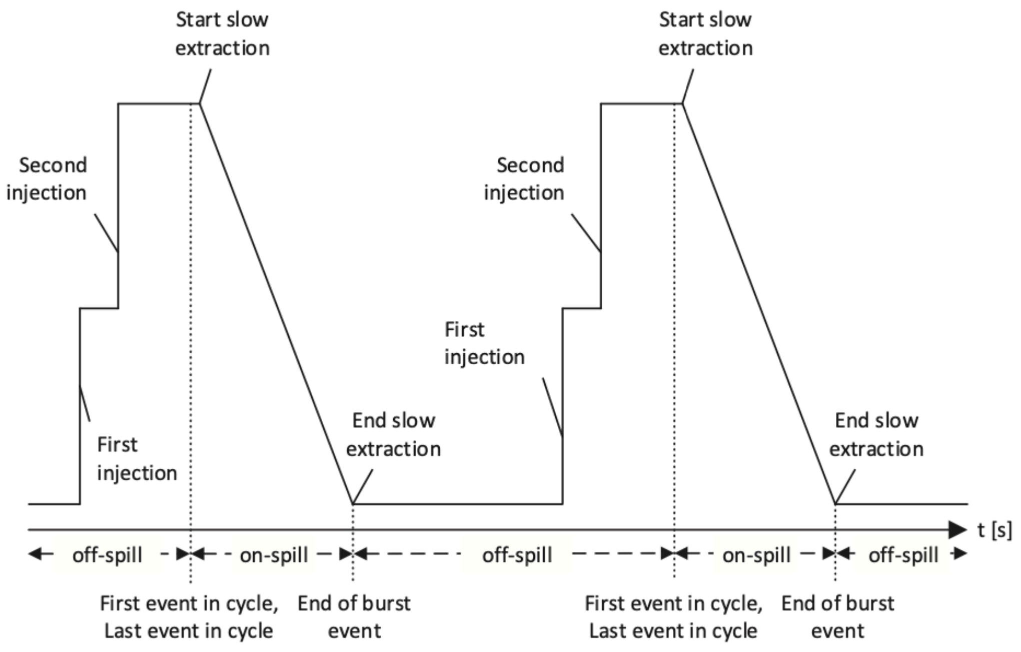


Figure 15: SPS Particle extraction procedure: proton beam intensity vs time. A whole SPS super-cycle is reported [24].

To initiate data flow through the event builder and establish correct timing, three cycles are required (and thus lost) before first physics triggers can be sent to the frontend electronics. The actual synchronization procedure is explained in detail in the next chapter of this work, when the context of the creation of the Artificial Spill Generator project is reported.

3.3.2 Continuously running mode

To safe time-consuming stop and restart of the data flow between two runs, the continuously running mode was introduced. It ensures a smooth transition between two consecutive runs without intervention of the shift crew and without stop of data flow through the event builder. Moreover, the idea of maintaining data flow in the event builder has been extended to periods when no data taking is requested. To do so a new state, the so called Dry Run, was introduced. It maintains the data flow through the whole acquisition chain and provides a monitoring data stream to the online monitoring tools but the acquired events are not written to hard drives. Therefore it serves as verification, monitoring, and diagnostic stage, even in periods when the experiment is not ready for data taking. The consecutive step relevant to real data taking is called Run and its start is possible on the next delivered spill since synchronization is already established. Using the Dry Run state and a smooth transition between runs, the data flow in the iFDAQ is only stopped in case of serious errors or in case of interventions on detectors that require a stop of trigger distribution to the frontends.

To sum it up, the iFDAQ using the continuously running mode is a self-running datataking system where decisions related to the continuously running mode are taken based on delivered events. Before the incorporation of the continuously running mode, the procedure for starting a new run after another run was successfully finished was always connected to a loss of beam time. Firstly, there is loss due to the already mentioned necessary synchronization phases (the start of run procedure requires three spills for the synchronization of the TCS with the SPS cycle and the terminating procedure takes one spill to end up data taking). Secondly, there is beam time loss due to necessary human intervention: before the incorporation of the continuously running mode, a run had to be started manually. Hence an inattentive shift crew could have added a significant part to the beam time loss by not starting the next run right after the previous run is stopped.

Eliminating both factors, the continuously running mode contributes to the efficiency of data taking with a 1.7% gain [24].

3.3.3 iFDAQ Software

The software side of the iFDAQ is used for control and monitoring of the final three layers of the iFDAQ hardware (FPGA multiplexers, FPGA switch, and readout engines) and for reading out physics events from readout engines [5]. It consists in the following processes:

- **Master process.** It runs on a dedicated computer and acts as the communication mediator between the iFDAQ processes. It incorporates supervision of states of the iFDAQ control system and the error handling.
- **Slave-control.** It runs on the readout engine computers and handles monitoring and configuration of the FPGA cards using direct communication through IPBUS.
- **Slave-readout.** It runs on the readout engine computers and handles decoding and verification of physics data.
- **Runcontrol GUI.** A Graphical User Interface (GUI) which provides control of the iFDAQ.
- **MessageLogger.** A process which collects messages from the slave and master processes and stores them in a database.
- **MessageBrowser.** A graphical user interface which allows the user to view messages from the slave and master processes in real time or retroactively browse the messages stored in a database.

All the above processes interact with or are part of the hardware components in the experiment. As it will become clear in Sec., said processes are fundamental for initializing, monitoring and tuning all of hardware components of the DAQ, in particular the Artificial Spill Generator module.

3.3.4 IPBUS Protocol

The communication between the different entities is performed through IPBUS protocol, which is a simple packet-based control protocol for reading and modifying memory-mapped resources within FPGA-based IP-aware hardware [33].

It defines the following operations:

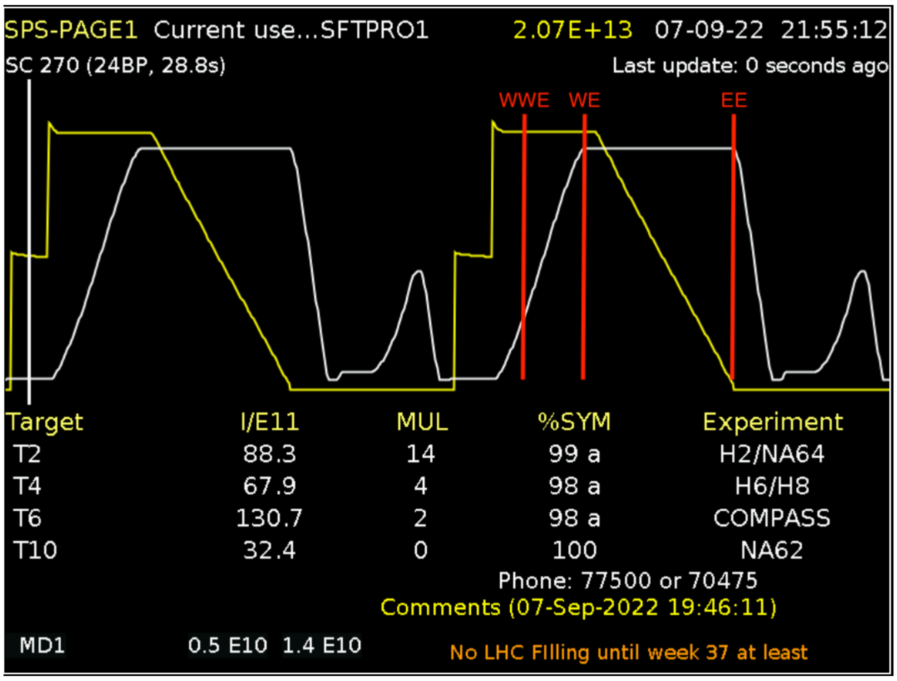
- A **Read** of user-definable depth. Two types are defined: address incrementing (for multiple continuous registers in the address space) and non-address-incrementing.
- A **Write** of user definable depth, again with (non-)incrementing types.
- A **Read-Modify-Write bits (RMWbits)** atomic bit-masked write, defined as $X := (X\&A)|B$, which allows to set/clear a subset of bits within a 32-bit register.
- A **RMWsum** increment operation, defined as $X := X + A$, which is useful fro adding values to a register.

The IPBUS protocol is transactional, meaning that for each read, write or RMW operation, the IPBUS client (typically software) sends a request to the IPBUS device; the device then sends back a response message containing an error code (equal to 0 for successful transactions), followed by the returned data in case of reads. Moreover, in order to minimize latency, multiple transactions can be concatenated into a single IPBUS packet.

The IPBUS suite of software and firmware implements a reliable high-performance control link for particle physics electronics and has nowadays successfully replaced VME control in several large projects [20].

4

Artificial Spill Generator



SPS-PAGE1 Current use...SFTPRO1 2.07E+13 07-09-22 21:55:12

SC 270 (24BP, 28.8s) Last update: 0 seconds ago

Target	I/E11	MUL	%SYM	Experiment
T2	88.3	14	99 a	H2/NA64
T4	67.9	4	98 a	H6/H8
T6	130.7	2	98 a	COMPASS
T10	32.4	0	100	NA62

Phone: 77500 or 70475

Comments (07-Sep-2022 19:46:11)

MD1 0.5 E10 1.4 E10 No LHC FIlling until week 37 at least

Figure 16: Yellow line: intensity of particle flux in the SPS accelerator, as explained in Fig.15 White line: beam energy delivered at the SPS experiments cited in the image. Red lines: synchronization signals [10].

COMPASS is located at the end of the M2 beamline in the north area of CERN and it exploits the particles accelerated by SPS. Together with the spills, three synchronization signals are sent to COMPASS experimental hall (Fig. 16), namely `WWE`,`WE` and `EE`, in order to properly identify the time limits of the particle bunches. The Warning Warning Extraction (`WWE`) is received at the very beginning of the spill, typically ≈ 1 s before particles arrive. The second Warning Extraction (`WE`) sets the actual beginning of the spill and it is received by the experimental hall a few 10 ms before the particles arrive.

This signal actually enables the electronics that are responsible for switching on the Trigger and the DAQ systems of the experiment, allowing the possibility to acquire data. The last signal Extraction End (`EE`) is eventually responsible for signalling the end of the particle flux, therefore it advises the experimental setup that the data taking can be suspended until the next SPS cycle [15].

The instructions are 2μ s Transistor-Transistor Logic (TTL) signals, adjusted each time to the SPS super-cycle timing, and they are converted into Nuclear Instrumentation Module (NIM) signals [14] to be feed into the Field Programmable Gate Arrays (FPGA) board. They usually follow a precise time scheme, but several variations can occasionally occur in the SPS extraction. The flat top region is generally 4.8 s long, while the `WWE` instruction usually comes 1 s before `WE`. In best super-cycle mode for the north area of CERN, the shortest super-cycle is 14.4 s. `WWE,WE` and `EE` are represented in Fig.17 as impulses raising from 0 to 1 but in the incoming description of the Artificial Spill Generator module, due to the inverted logic of the FPGA (see Sec.4.1), they will be treated as falling values from 1 to 0. In the same image, the SPS timing structure is reported with respect to the on/off spill phases as interpreted from COMPASS experiment and DAQ system.

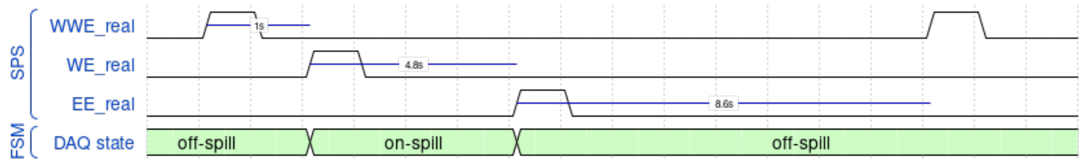


Figure 17: SPS instructions with respect to DAQ spill phase. The DAQ state refers to the status of the Finite State Machine internal to COMPASS DAQ system.

The creation of an Artificial Spill Generator internal to COMPASS framework happens to be necessary for several reasons. The data taking process relies entirely on the `WWE,WE,EE` signals, therefore the necessity of generating them internally and independently from SPS is fundamental.

Unfortunately, the signal generation procedure can sometimes result in the production of an imperfect spill structure. For example, the distribution of timing signals can be stopped during machine developments for maintenance reason or to perform tests. In addition, due to issues with the level converters of timing logic at SPS, some of the three signals can be missing. These events and all the other possible configurations of imperfect signal generation could cause the failure of the DAQ of the experiments downstream from SPS, therefore a controlled generation of instructions must be introduced in order to solve this problem. Moreover, for testing purposes of stand alone DAQs, which does not rely on the beam, an artificial generation of these signals is essential. The possibility of selecting these forms is also extremely important for the proper functioning of the so called Dry-Run mode, which is used to inspect the correct functioning of the detectors and DAQ. Even if the complete absence or imperfect generation of SPS signals should be rare, the experiments must be able to deal with this possibility. For this reason, the artificial instructions can entirely substitute the real signals.

Together with a generation module, the creation of a monitoring system for the parallel instruction signals is mandatory. The two sources of signals (SPS and internal) must be selected accurately from the DAQ control Graphical User Interface (GUI), setting a 1-bit `switch` value to 1 for the Real and to 0 for the Artificial instructions. The transition between the two and the following forwarded result to the DAQ must be coherent. The superposition of signals coming from different sources must always be avoided, since it can result in the representation of a spill structure with false time coordinates, possibly causing a failure of the DAQ or a huge loss of data. The Artificial Spill Generator module was created with the purpose of solving several timing and loading issues. In particular, the main achievement for COMPASS collaboration, which saved a lot of physics data, was the feature of skipping spills. Since the loading of a faulty detector takes a lot of time and has to be executed without the arriving of any SPS signal (which would otherwise re-synchronize the front-end electronics and lead to a DAQ crash), the DAQ has to be in stop state. The introduction of the fast 14.4 s SPS super cycle results in a too short off-spill phase to do the detectors loading, therefore the Artifical Spill generator allows to skip the next spill and saves a considerable amount of time to load the detector in the stop mode of the DAQ.

Eventually, several useful services for the measurement of some important spill parameters were also added, together with the possibility to visualize them directly on the DAQ user interface. Such specific tool was implemented on hardware in order to exploit the low latency and reliability of the FPGA board and eventually to be inserted in the DAQ pipeline of COMPASS experiment. In the following pages, the various components present within the firmware and their logic are presented. The circuit mappings are reported with a deductive approach, starting from macroscopic block diagrams and eventually explaining the detailed waveform diagrams of each entity.

4.1 FPGA and VHDL

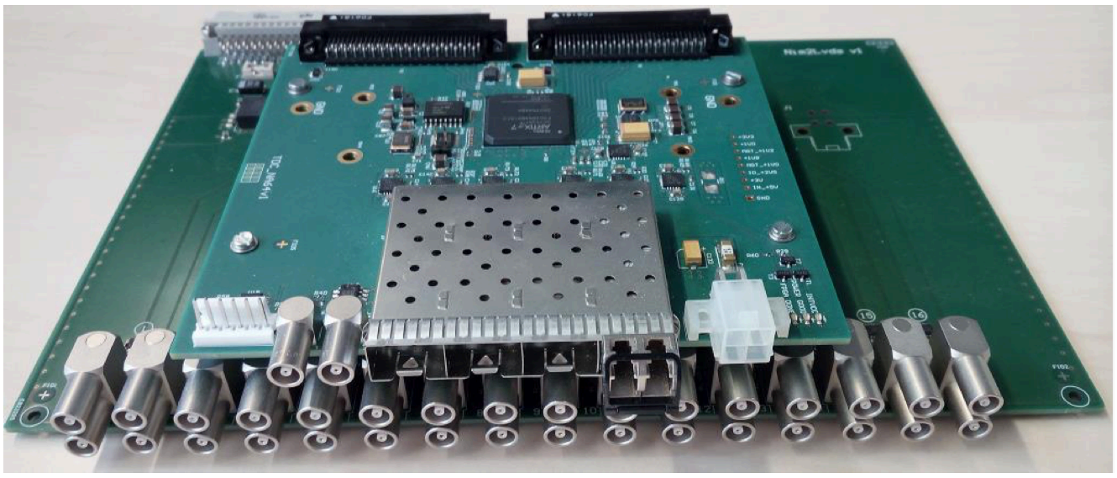


Figure 18: Picture of FPGA board used for the Artificial Spill Generator implementation.

Before moving to the actual description of the firmware, a few fundamental concepts that will be used in the following sections must be introduced. The logic of the Artificial Spill Generator has been built by means of Very high-speed integrated circuit Hardware Description Language (VHDL), eventually implementing the whole firmware on the FPGA board visible in Fig.18. FPGA are semiconductor devices that are based around a matrix of configurable logic blocks, connected via programmable interconnects. They can in fact be reprogrammed to a desired application or functionality requirement after manufacturing. This particular feature distinguishes FPGA from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks [35]. The board in use for this project was an ARTIX-7 FPGA XC7A-35 with 64 LVDS and 2 LEMO channels, 2 NIM Input/Output pins for Trigger signals, 4 SFP cages for data read-out and slow control and 5 V power supply.

VHDL programming language presents several concepts which are typical of its nature and which need to be properly introduced for a better understanding of the rest of this work. First of all, it is mandatory to provide the definitions of the objects exploited within VHDL codes. Nonetheless, also the "variables" used in the codes can belong to different classes and types, which present slightly different characteristics and are used for different purposes. A summary of what is used in the Artificial Spill Generator codes is reported below [18]:

Objects

- **`entity`**: It's a design block of the logic, representing a specific integrated circuit with Input and Output signals. Different entities will contain a different architectures describing its logic behaviour.
- **`component`**: A smaller `entity` can be inserted into the architecture of a bigger one in order to perform a specific task, therefore becoming one of its `components`.
- **`process`**: A `process` describes the logic behaviour of an `entity` inside its architecture. Several processes can be part of a single `entity`, acting either with concurrent statements (active at the same time) of with sequential statements.

Classes

- **`constant`**: An object of class `constant` holds the same value for its entire life. It is set when the object is created and cannot be changed afterward.
- **`variable`**: The only state associated with a variable is its current value, which can be modified with a *variable assignment* statement. When this is executed, the variable immediately assumes its new value and no history is kept about the previous ones.
- **`signal`**: It is used to transport values between different parts of a design, which could be processes and concurrent statements of the same architecture or entities that compose a design hierarchy

Types

- **`integer`**: Usual type containing integer base 10 numbers.
- **`boolean`**: Usual type containing TRUE/FALSE values.
- **`STD_LOGIC`**: 1-bit binary value, mainly used in this work for impulse signals and flags.
- **`STD_LOGIC_VECTOR`**: Fixed length vector of binary values, mainly used in this work as 32-bit measured values.

4.2 General Structure

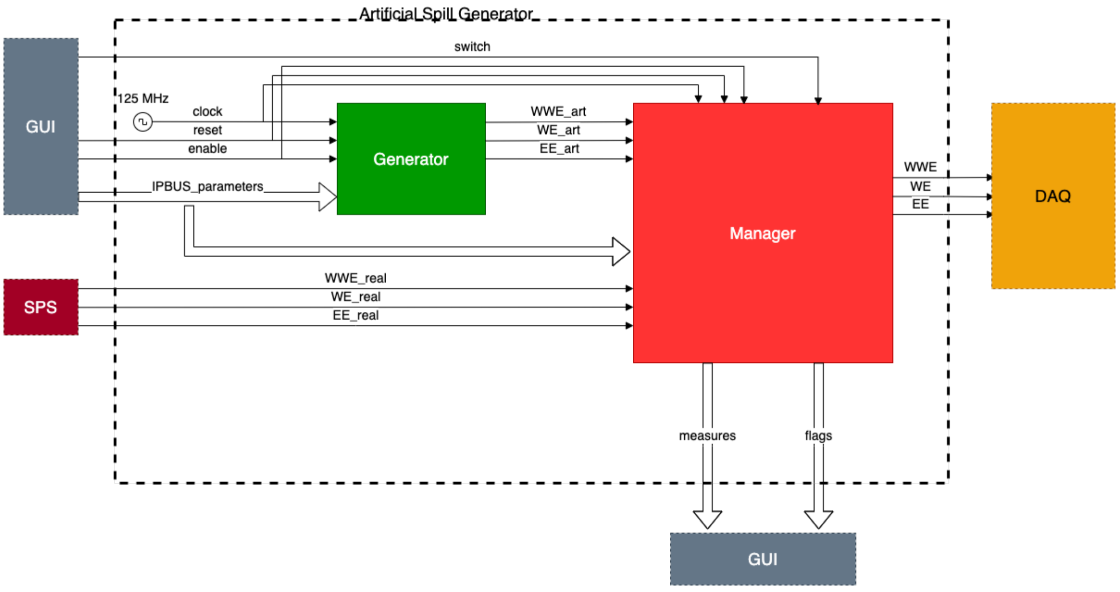


Figure 19: Macroscopic structure of the Artificial Spill Generator firmware.

The macroscopic structure of the Artificial Spill Generator can be divided into two main entities: the Generator and the Manager (Fig.19). The former is responsible for the generation of the artificial signals, which are always present and independent from SPS ones. The latter has the task to manage parallelly the Real and Artificial signals, selecting one of the two according to the `ext_switch` signal and sending the result to the DAQ system.

The whole module can be controlled remotely from the DAQ's GUI via an IPBUS interface [20]. The user is able to tune several parameters, both for the generation and for the management of the signals, which are sent to hardware through IPBUS communication protocol. The 1-bit signals `ext_reset` and `ext_enable` are also set in the GUI, since they are meant for starting, stopping or resetting the whole module. The FPGA board eventually returns a series of measurements and flag values that are useful for the control room to have insights on the phases, the structure and the duration of the spills. The system relies on the 125 MHz clock internally generated by the FPGA, which corresponds to a 8 ns time unit (see Tab.15). In the following pages the behaviour of the logic involved in the above cited macro-structures is described, going into the details of the different entities and paying particular attention to the physical reasons behind the choices of the implementation.

4.3 Generator

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`reset`	1	Input	Boolean for resetting.
`enable`	1	Input	Boolean for enabling.
`WWE_delay_in`	32	Input	Distance between `WWE-WE`.
`WE_delay_in`	32	Input	Distance between `WE-EE`.
`SIGNAL_LENGTH`	8	Input	Duration of the impulses (8 ns units).
`WWE_out`	1	Output	`WWE` signal
`WE_out`	1	Output	`WE` signal.
`EE_out`	1	Output	`EE` signal.

Table 1: Summary of In/Out signals of Generator entity.

The logic implemented in the Generator entity allows the control room user to freely tune the main parameters of the artificially generated spill structure, giving them the possibility to generate impulses of variable length and distance between one another. For example, the `SIGNAL_LENGTH` parameter indicates the duration of each impulse (`WWE,WE,EE`) in terms of 8 ns units[3], while the distances between `WWE-WE`, `WE-EE` and `EE-WWE` are counted in terms of 1 ms units. These parameters are set according to the specifics required; they are then sent to the FPGA via IPBUS protocol and received in input by the Generator. This component also receives the FPGA's 125 MHz `clock` signal, together with the `enable` and a `reset` 1-bit values set in the GUI (see Tab.1).

In order to ease the debugging and to build the logic of the Generator entity in an analytic way, four different smaller components have been defined with specific purposes and they have been optimally linked together in order to reach the final artificial signal generation goal. First of all, a 1 kHz periodic signal generator was defined (in the following called `ms_counter`) in order to reach the 1 ms timescale for the measurement of the intervals between the instructions. Then a `variable_counter` entity was designed, even though originally created for another section of the whole project. Its aim is to raises a 1-bit `end_of_count` flag when a specific maximum value is reached. It was used in this context to keep track of the time intervals between the instructions and send the flag signal to the Finite State Machine (FSM) component. Moreover, an `ipbus_params_register` entity has been defined in order to guarantee a smooth transition between the possible changes in the parameter values. Mentioned entities are firstly presented in detail and eventually connected to one another, in order to better explain the overall functioning of the Generator.

[3]Since the important features of the signals are their rising and falling edges, the length of the impulse does not carry fundamental information. Nonetheless, the possibility to tune such value had to be inserted in the logic, in order to provide a good versatility of the circuit for possible future needs.

4.3.1 MS_COUNTER ENTITY

Signals	Bits	In/Out	Description
clock_in	1	Input	Periodic: 125 MHz.
enable	1	Input	Boolean for enabling
clock_out	1	Output	Periodic: 1kHz.

Table 2: Summary of In/Out signals of ms_counter entity.

The ms_counter entity is responsible for the generation of a periodic 1 kHz signal. In order to reach this goal, it receives in input the 125 MHz clk_i signal together with a 1-bit enable value and it outputs the clk_o wave with a period of 1 ms. As internal signals, the count_m_tick is defined as a 32-bit logic vector, with the aim of counting the number of periods of the fast clock before reaching the desired value. The m_tick is instead the internal signal version of the clk_o, which is initialized to one and updated to its opposite when the time limit is reached (see Tab.2).

With a brief calculation, it is clear that the entity needs to count up to 125.000 fast clock periods in order to reach a 1 ms complete slow clock cycle, therefore a constant variable semiperiod is defined inside the VHDL code as a 32-bit integer value, in order to have a fixed number to compare to the counter's result. The logic needs to be able to update the m_tick signal to its opposite value when the counter has reached the equivalent of a semiperiod of the slow clock ($0.5ms$), counted in terms of $8ns$ units. The constant semiperiod value is therefore fixed at 62500, which in binary format translate to "1111010000100100".

$$n_{periods} = \frac{\frac{T_{1kHz}}{2}}{T_{FPGA}} = \frac{0.5ms}{8ns} = 62500 \rightarrow "1111010000100100"$$

The process inside the ms_counter's logic is only sensitive to the rising edge of the fast clock signal clk_i. When this event happens, if the enable value is set to 0, count_m_tick is reset to 0 and m_tick is reinitialized to 1, while if the enable is instead set to 1, the counter can perform its work. In particular the logic states that if count_m_tick has reached the semiperiod value, then the m_tick needs to be updated to its opposite value and the counter reset. Until this does not happen, count_m_tick is increased of one unit at each clk_i cycle. Eventually, the value of m_tick is assigned to clk_o at every activation of the process, producing the periodic 1kHz signal originally required (Fig.20).

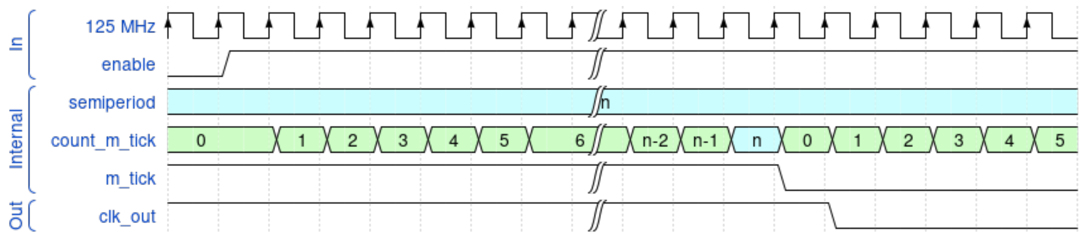


Figure 20: Wave behaviour of the ms_counter component.

4.3.2 VARIABLE_COUNTER

Signals	Bits	In/Out	Description
clock	1	Input	Periodic: 125 MHz.
enable	1	Input	Boolean for enabling.
reset	1	Input	Boolean for resetting.
max_clock_cycles	32	Input	Max number of clock periods.
result	32	Output	Real-time count result.
end_of_count	1	Output	8 ns impulse.

Table 3: Summary of In/Out signals of variable_counter entity.

The variable_counter entity was originally designed to perform the measurements on the time intervals between the spill signals actually forwarded to the DAQ system (see Sec. 4.5.1), but it turned out to be useful also in this context, since its logic has been defined with a good versatility.

In particular, together with the 1 kHz clock, enable and reset signals, this component receives in input the 32-bit logic vector that indicates the maximum number of clock cycles that has to be reached. In this way, a single entity can be applied in multiple contexts: the same would not have been possible with the definition of a constant value inside the component's logic. The variable_counter returns in output the result of the count (only for simulation purposes) and it raises a 1-bit flag value when the comparison with the input signal max_clock_cycles states that the maximum value has been reached. Another difference with respect to the previous entity is that while the ms_counter was built to generate a periodic signal with a specific frequency, the variable_counter is only meant to produce an impulse of the duration of a clock cycle (see Tab.3).

Also in this case, the logic of the component is made out of one single process, which is sensible to the rising edge of the input clock[4]. When reset=1 or enable=0, the end_of_count flag and the inner signal temp_result are both reinitialized to 0. Whenever the opposite combination of enable and result occurs, respectively 1 and 0, the counter performs its duty. By means of an if statement, temp_result is increased by 1 unit at every clock cycle; when the maximum value is reached, the counter is reset and the end_of_count is raised to 1. The temporary counter result is sent in output at each activation of the process, in order to keep proper track of its value. The 1-bit end_of_count flag stays high only for $1ms$ (Fig.21).

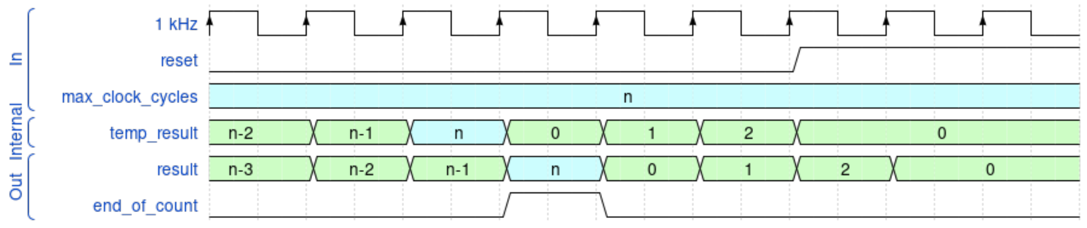


Figure 21: Wave behaviour of the variable_counter component with enable=1.

4.3.3 IPBUS_Register

The creation of an IPBUS_register entity turned out to be fundamental in order to guarantee the smooth transition of the main parameters of the artificial signals. Together with the FPGA's clock signal, the ipbus_params_register receives as input four 32-bit STD_LOGIC_VECTOR signals, set by the user in the

[4]From the definition of the entity, there is no particular prescription on the frequency of the input clock. Nevertheless in this case it has only been used with the $1kHz$ clock generated by ms_counter, as it can be seen in Fig.21. The reason behind this choice will become clear in the following sections.

control room. These numerical values represent the duration of the intervals between `WWE-WE`,`WE-EE`,`EE-WWE` instructions and they are called respectively `WWE_delay_in`, `WE_delay_in` and `EE_delay_in`. The duration of the impulses is also a tunable parameter, therefore the IPBUS register must be able to read this value under the name of `SIGNAL_LENGTH_in`, which is a 8-bit `STD_LOGIC_VECTOR`. After processing these information, the register is responsible for sending the right values to the downstream entities, where the proper signal building is performed. `WWE_delay_out`,`WE_delay_out`, `EE_delay_out` and `SIGNAL_LENGTH_out` are outputted, together with a 1-bit `soft_reset` signal which is responsible for signalling the change of the registered spill parameters (see Tab.4).

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`WWE_delay_in`	32	Input	Time distance between `WWE-WE`.
`WE_delay_in`	32	Input	Time distance between `WE-EE`.
`SIGNAL_LENGTH`	32	Input	Duration of impulses (8 ns units).
`WWE_delay_out`	32	Output	Time distance between `WWE-WE`.
`WE_delay_out`	32	Output	Time distance between `WE-EE`.
`SIGNAL_LENGTH`	32	Output	Duration of impulses (8 ns units).
`soft_reset`	1	Output	8 ns impulse.

Table 4: Summary of In/Out signals of `IPBUS_register` entity.

Even for this component, there is a single active process, sensitive to the rising edge of the FPGA's 125MHz clock received in input. Eight different variables are defined internally to the said process and initialized to 0, with the aim of storing two copies of the four signal parameters, registered at a time difference of a single clock cycle.

At each clock's rising edge, the input parameters are read and stored in `sl_new`,`WWE_new`,`WE_new` and `EE_new` variables, which will always represent the values instantly set in the GUI. A comparison between the old and new versions of the parameters is performed at each activation of the process: if a difference in any of those is retrieved, the values of the new variables are assigned to the old ones and the `soft_reset` flag is raised to 1. If instead no difference in the stored values is percieved, the input parameters are directly transferred to the output 32-bit signals, paying attention to perform a zero padding for SIGNAL_LENGTH, in order to make it a 32-bit long STD_LOGIC_VECTOR. In this situation, the soft_reset signal remains low. Eventually, in order to solve initialization issues, if for any malfunctioning the component would not receive any input data, the soft_reset flag would be raised, with the aim of protecting the following entities of the logic and, in the end, also the DAQ system.

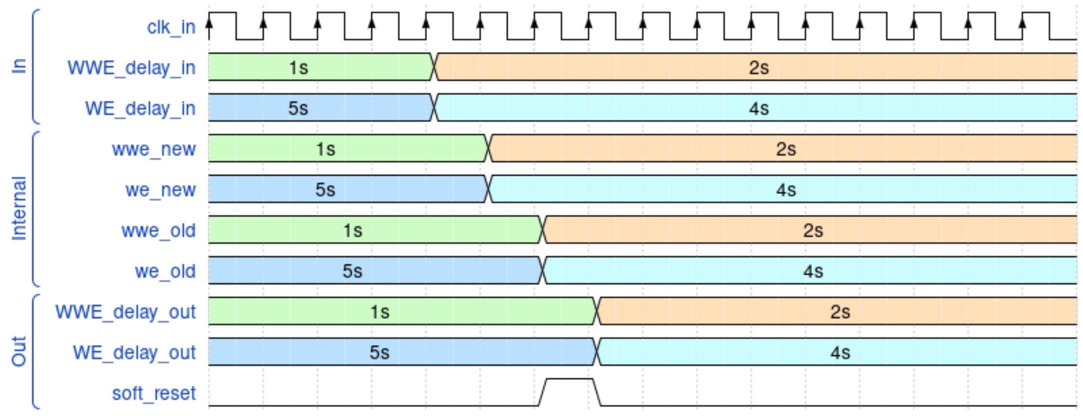


Figure 22: Wave behaviour of IPBUS_register component for the variation of WWE_delay_in and WE_delay_in parameters.

In Fig.22 the general functioning of the component is represented. The case of a synchronized change in the WWE_delay_in and WE_delay_in is reported[5], together with their relative variables and soft_reset impulse generation. EE_delay_in and SIGNAL_LENGTH are not present in the graph just to simplify its interpretation.

[5]These parameters represent the set time interval that should pass respectively between WWE-WE and WE-EE.

4.3.4 Finite State Machine

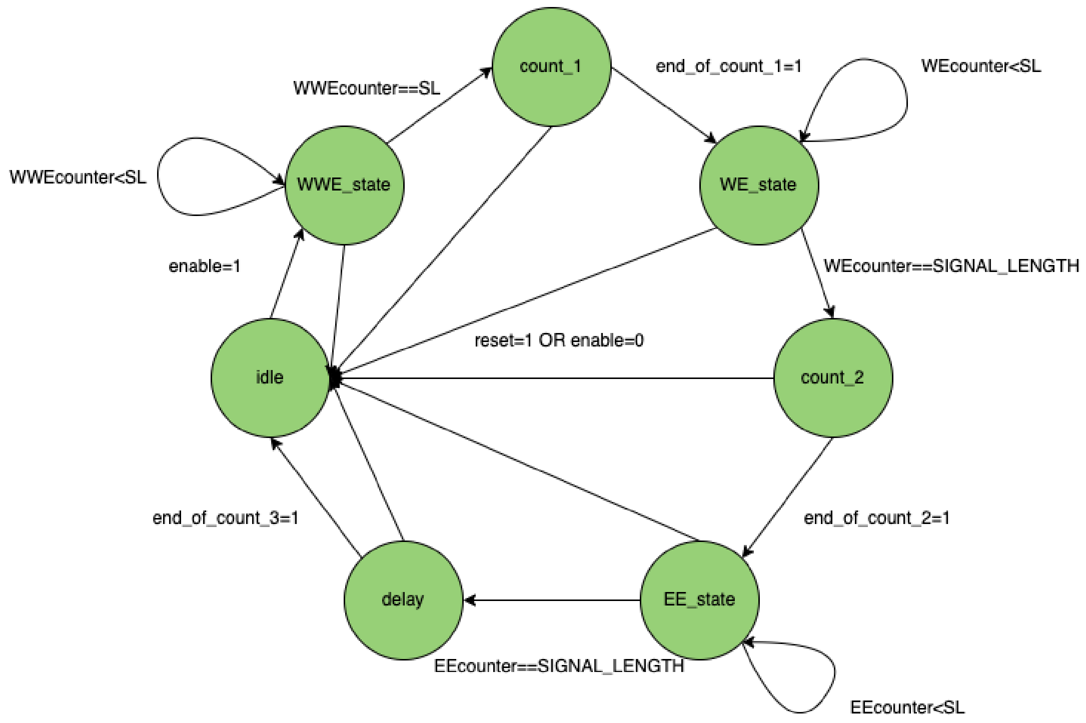


Figure 23: Finite State Machine diagram.

The Finite State Machine process rules the transition of the logic between seven possible states: `idle`, `WWE_state`, `count1`, `WE_state`, `count2`, `EE_state` and `delay` (Fig.23). Together with the 125 MHz clock signal, it receives in input the 1-bit `ext_enable` and the 32-bit `SIGNAL_LENGTH` value coming from the `IPBUS_register`. The `hard_reset` signal coming from the control room is instead convoluted with the `IPBUS_register`'s `soft_reset` by means of an `OR` port: the result of this operation, simply called `reset`, is then forwarded to the FSM[6]. The computed values of the three different `variable_counter` entities and their corresponding `end_of_count` impulses are also received in input by the FSM, since they are meant to count the time intervals between the generated

[6]In this context the collaboration required a different behaviour with respect to the one implemented for the input parameters in the Manager's component (see section 4.3.1). Whenever the spill parameters of the Generator are changed in the control room, the effect must be instantly visible in the logic, hence the resetting of the state machine either via `hard_reset` or `soft_reset`. This means that any change in the parameters will cause an interruption of the signal generation.

artificial instructions. At the same time, three `enable` and `reset` signals for the said counters are returned in output, accurately timed depending on the state in which the machine is found. As the last component in the Generator's architecture, the FSM is responsible for properly generating `WWE_art`,`WE_art` and `EE_art`, which are eventually propagated to the Manager entity (see Tab.5).

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`ext_enable`	1	Input	Boolean for enabling.
`reset`	1	Input	Boolean for resetting.
`SIGNAL_LENGTH`	32	Input	Duration of impulses (8 ns units).
`result_1`	32	Input	Result of `variable_counter_1` count.
`end_of_count_1`	1	Input	`variable_counter_1` 8 ns impulse.
`enable_1`	1	Output	Boolean for enabling `variable_counter_1`.
`reset_1`	1	Output	Boolean for resetting `variable_counter_1`.
`result_2`	32	Input	Result of `variable_counter_2` count.
`end_of_count_2`	1	Input	`variable_counter_2` 8 ns impulse.
`enable_2`	1	Output	Boolean for enabling `variable_counter_2`.
`reset_2`	1	Output	Boolean for resetting `variable_counter_2`.
`result_3`	32	Input	Result of `variable_counter_3` count.
`end_of_count_3`	1	Input	`variable_counter_3` 8 ns impulse.
`enable_3`	1	Output	Boolean for enabling `variable_counter_3`.
`reset_3`	1	Output	Boolean for resetting `variable_counter_3`.
`WWE_art`	1	Output	Artificial `WWE` signal.
`WE_art`	1	Output	Artificial `WE` signal.
`EE_art`	1	Output	Artificial `EE` signal.

Table 5: Summary of In/Out signals of the `FSM` entity within the generator component.

The actual FSM process is activated by the rising edge of the 125 MHz clock and it is initialized to the `idle` state. In this phase every counter is reset and the output instruction signals `WWE_art`, `WE_art` and `EE_art` are held constant at 1. Whenever the `reset` and `enable` equal respectively to 0 and 1, the machine is able to move to the next state. As a safety measure, an `if` statement has been defined in each state, in order to deal with the possible sudden changes of the `ext_reset` and `ext_enable` values. If the condition `reset`=1 OR `enable`=0 should happen, each state would be moved back to `idle`, resulting in an overall re-initialization of the machine.

The WWE_state is the phase in which the first instruction impulse is built (Fig.24). A counter_WWE 32-bit logic vector is used to count up to the input value of SIGNAL_LENGTH, which fixes the duration of the impulse in 8 ns time units. During the count, the WWE_art value is lowered to 0 in order to create the falling edge signal; whenever the time limit is reached it is brought back to 1, stating the end of the WWE_art instruction. At this point the variable_counter1 entity is enabled, which is responsible for keeping track of the distance between WWE-WE instructions during the count_1 phase. The machine is then moved to the next state.

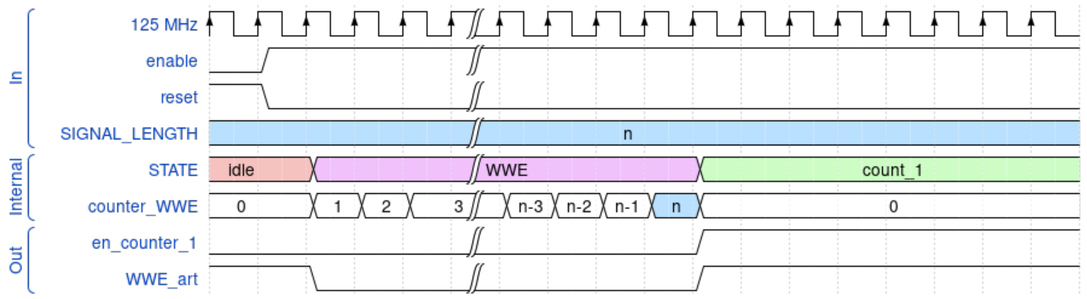


Figure 24: Wave representaation of the idle-WWE-count1 FSM transition.

The count_1 state is maintained until the variable_counter_1 entity produces an end_of_count1 impulse that signals the necessity to start developing the WE_art instruction (Fig.25). In the other transitions reported in Fig.23, the machine alternates between states in which the creation of the impulses is done (WE_state and EE_state) and states in which the logic needs to wait for the proper time value between instructions (count_2 and delay). Eventually, the order of the states follows a loop pattern, with the aim of reproducing the periodicity of the Artificial signals. After the final delay state is reached, which represents the time distance between the EE instruction of the previous spill with the WWE instruction of the following spill, the loop starts back from the initial idle state.

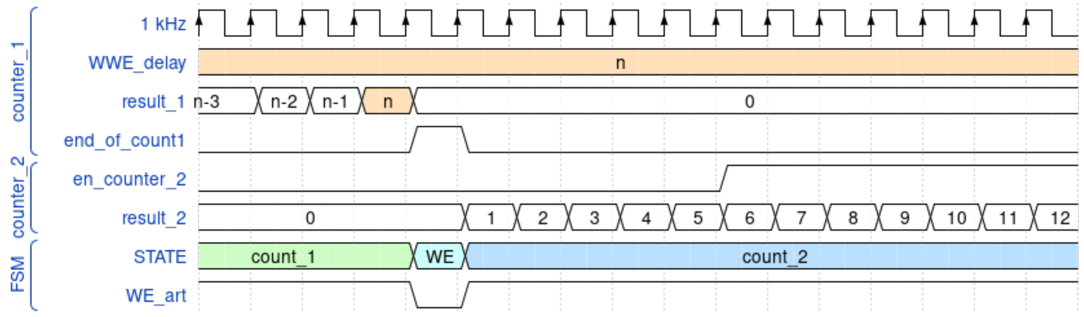


Figure 25: Off-scale wave representation of `count_1-WE-count_2` FSM transition.

The artificial instruction generation is reported in Fig.26 with respect to the FSM states. The picture is not accurately representing the time behaviour of the various signals, since the impulses and the difference between them live on different time scales, respectively ns and ms. An accurate representation of the `WWE,WE,EE` and `idle` states would report them as almost instantaneous with respect to the states in which the `variable_counters` are active (see Fig.51).

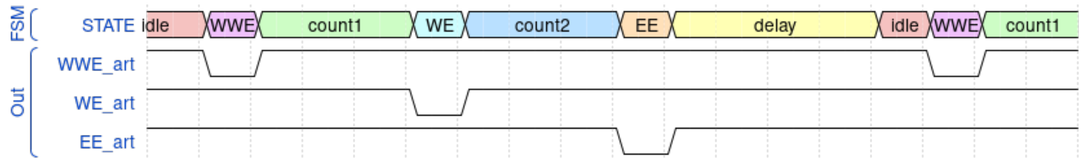


Figure 26: Off-scale representation of instructions w.r.t. states.

4.3.5 Generator's Detailed architecture

The Generator's logic exploits all the previously described components in order to properly generate the artificial instructions. Three different `variable_counter` entities are implemented with the aim of separately keeping track of the time intervals `WWE-WE`, `WE-EE` and `EE-WWE`, while one single copy of `ms_counter` and `IPBUS_register` is used and the FSM deals with the proper instruction generation (see Fig.27). Since different components need to work on different time scales, the FPGA fast `clock` is only received in input by the `IPBUS_register`, the `ms_counter` and the FSM processes.

The `variable_counter` entities instead need to keep track of the time intervals between instructions, therefore they need to perform measurements with 1 ms time units, exploiting the 1 kHz clock signal generated by `ms_counter`.

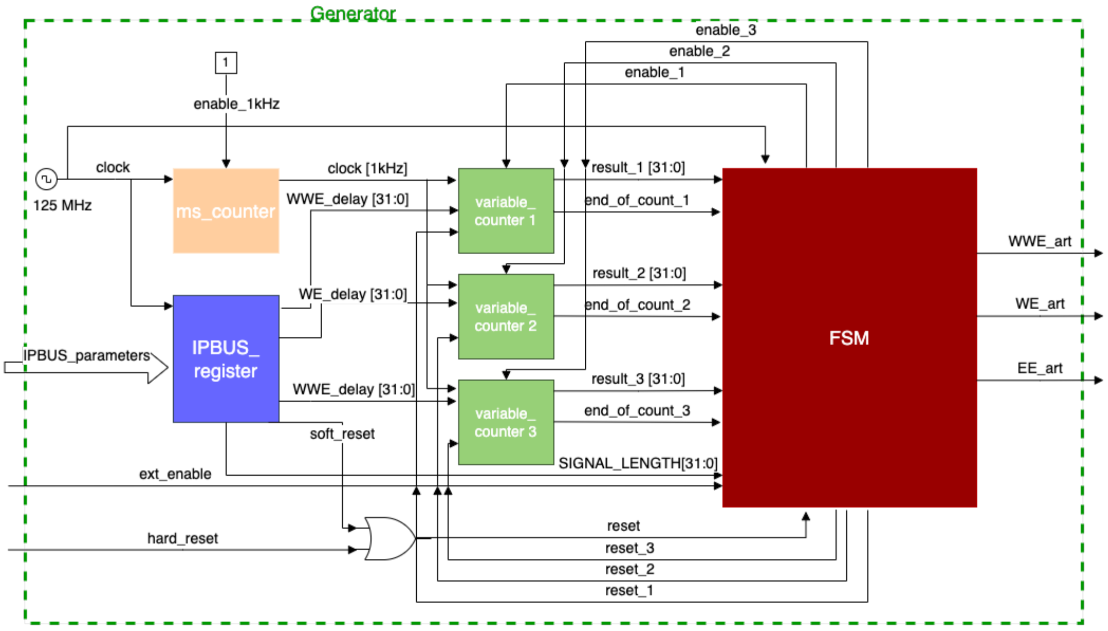


Figure 27: Detailed architecture of the Generator component (see Tab.1).

The spill parameters set in the GUI are sent to the `IPBUS_register` and properly processed; the delay values between `WWE-WE`, `WE-EE` and `EE-WWE` are then forwarded separately to the three different `variable_counter` entities, where they are interpreted as the maximum number of reachable clock cycles. The `SIGNAL_LENGTH` value instead is only read by the FSM process, where specific states are responsible for the impulse generation and the measurement of their duration. Also the `ext_enable` value is only read by the FSM, since it simply works as a binary switch of the artificial signals generation.

The FSM process is the core entity of the Generator component. By means of an organized and strictly ruled ordering of its states, it is responsible for the actual generation of `WWE_art`, `WE_art` and `EE_art`. Depending on the states in which it is found, the machine can output six different internal signals, aimed at resetting or enabling the three `variable_counter` entities. At the same time, the `end_of_count` flags outputted by the `variable_counters` are read by the FSM. This feedback allows the machine to know when the maximum number of clock cycle has been reached and it acts consequently on the change of its states. Defining different entities for different purposes (like in this case, FSM and `variable_counters`) allows the creation of asynchronous processes, it simplifies the debugging of the code and it improves the effectiveness of the architecture.

4.4 Manager

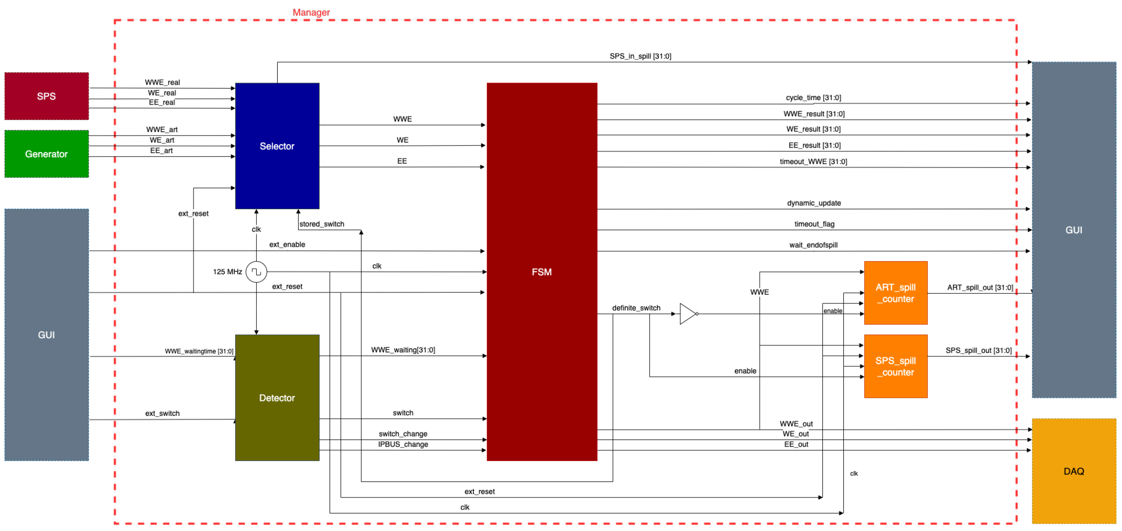


Figure 28: Detailed architecture of Manager component.

The Manager component represents the core logic of the Artificial Spill Generator firmware. It is responsible for monitoring the Artificial and Real instructions, selecting the one to be sent to the DAQ according to the `switch` signal. Even though the initial `switch` value is set from the control room, the Manager is both able to read it and overwrite it, depending on the different necessities of the experiment. In the following, the `switch`=1 value will represent the selection of the instructions coming from SPS, while the opposite `switch`=0 will correspond to the exploitation of the artificially generated signals.

Signals	Bits	In/Out	Description
WWE_real	1	Input	SPS WWE signal.
WWE_art	1	Input	Artificial WWE signal.
WE_real	1	Input	SPS WE signal.
WE_art	1	Input	Artificial WE signal.
EE_real	1	Input	SPS EE signal.
EE_art	1	Input	Artificial EE signal.
clock	1	Input	Periodic: 125 MHz.
switch	1	Input	Signal selection(1:Real, 0:Artificial).
ext_reset	1	Input	Boolean for resetting.
enable	1	Input	Boolean for enabling.
max_waitingtime_WWE	32	Input	Limit of Timeout service.
WWE_final	1	Output	WWE forwarded to DAQ.
WE_final	1	Output	WE forwarded to DAQ.
EE_final	1	Output	EE forwarded to DAQ.
result_WWE	32	Output	Measured distance between WWE-WE.
result_WE	32	Output	Measured distance between WE-EE.
cycle_time	32	Output	Measured distance between spills.
timout_WWE	32	Output	Real-time value for WWE waiting time.
actual_switch	1	Output	Result of switch internal selection.
dynamic_update	1	Output	Flag signalling forced switch.
timeout_flag	1	Output	Flag signalling timeout service.
wait_endofspill	1	Output	Flag: non-update of parameters.
SPS_in_spill_number	32	Output	Received SPS spill signals.
SPS_out_spill_number	32	Output	Processed SPS spill signals.
ART_in_spill_number	32	Output	Processed Artificial spill signals.

Table 6: Summary of In/Out signals of the Manager entity.

Three main entities contribute to the correct functioning of the Manager component, namely the Detector, the Selector and the Finite State Machine. Additional smaller spill_counters are inserted inside the Selector and downstream from the FSM, in order to count the number of received and processed spills. Thanks to the FSM process, several measurements on the spill parameters are performed and can be readout via IPBUS, together with some useful flag values that signal specific characteristics of the said spills. The logic behaviour of the said components is explained in the following sections, together with a detailed description of how they interact with each other.

The Input and Output signals of the Manager are shown in Tab.6, even if their nature will become clear only in the following sections. The whole architecture of the Manager component is reported (see Fig.28), with a precise explanation of its main services and purposes. Eventually, the various measurements performed are contextualized within the logic of the firmware and the requirements of the experiment.

4.4.1 Detector

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`max_waitingtime_WWE_in`	32	Input	Limit of Timeout service.
`switch_in`	1	Input	Incoming `switch` signal.
`max_waitingtime_WWE_out`	32	Output	Limit of Timeout service.
`switch_out`	1	Output	Outgoing `switch` signal.
`ipbus_change`	1	Output	8 ns impulse: changes in IPBUS.
`switch_change`	1	Output	8 ns impulse: change in `switch`.

Table 7: Summary of In/Out signals of the Detector entity.

The Detector is responsible for keeping track of any changes in the values received from the control room, in order to guarantee a smooth transition between them. It receives in input the maximum number of clock cycles that can be measured before the arrival of `WWE` instruction, namely `WWE_waitingtime`, together with the boolean `ext_switch` signal and the FPGA 125 MHz clock. At the same time it returns in output two 1-bit values aimed at signalling the variation of any input parameters, simply called `ipbus_change` and `switch_change`: these will eventually result to be fundamental for the correct management of the spill structure in the FSM process. After being processed, the `ext_switch` and `WWE_waitingtime` values are forwarded to the downstream entities (see Tab.7).

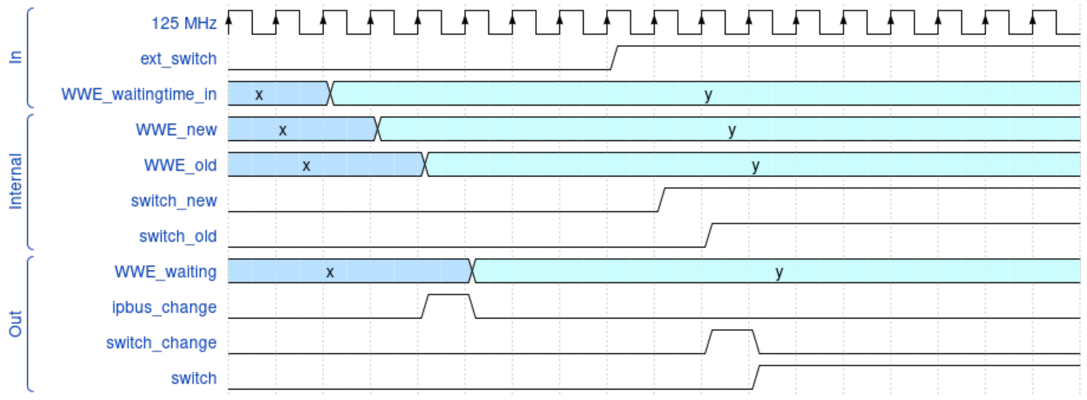


Figure 29: Detector's wave behaviour for `switch` and `WWE_waitingtime` sudden changes.

Four different variables (32-bit `STD_LOGIC_VECTOR` for the IPBUS parameters and 1-bit for the switch value) are defined inside the Detector and initialized to 0, in order to store the old and updated values of the parameters received in input. The process is activated by the rising edge of the 125 MHz clock and its logic resembles the one of the `IPBUS_register` entity described in the previous pages (see section 4.2.3). In particular, every $8ns$ all the input parameters are stored in the `new` variables. The detection of changes for the waiting time and switch value are separated by two different `if` statements, since they conceptually represent different cases for the treatment of the spill structure. Nevertheless, the logic behind the variation detection is the same: whenever any difference between the `old` and `new` variables is present, the formers are assigned with the values of the latters and the 1-bit `change` impulse is raised to 1. If none of the variables is subject to any change with respect to the previous clock cycle, the `old` variables are assigned with the input parameters and the impulses are brought back to 0.

In Fig.29 the wave representation of the Detector's signals is reported. As it is clear from the image, the output values are updated with a three clock cycles delay with respect to the input ones, corresponding to an overall latency of 24 ns.

4.4.2 Selector

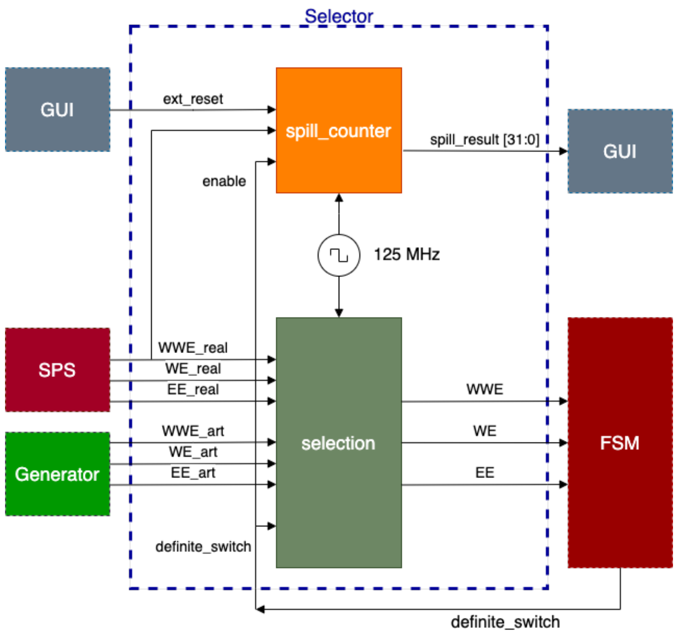


Figure 30: Detailed architecture of Selector component.

Inside the Selector's logic there are two concurrent processes, which paralelly perform the signal selection and the count of the number of incoming SPS spills (Fig.30). The whole entity receives the Real and Artificial instructions in input, which are read by the selection process. The `WWE_real` in particular is also sent to the `spill_counter` component, as it will be fundamental for the proper SPS spill count (see section 4.5.3). The results of the selection are eventually returned in output and sent to the downstream FSM entity. The selection process happens according to the `definite_switch` value received from the FSM, which is the processed result of what it is originally received in input as `ext_switch` by the Detector. Such 1-bit value also works as enabling signal for the `spill_counter`, since it is only meant to count the incoming SPS instructions (`definite_switch`=1=`enable`). It eventually returns in output the measured spill number at every rising edge of the 125 MHz clock (see Tab.8).

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`stored_switch`	1	Input	`switch` selected in FSM.
`ext_reset`	1	Input	Boolean for resetting..
`WWE_real`	1	Input	SPS `WWE` signal.
`WE_real`	1	Input	SPS `WE` signal.
`EE_real`	1	Input	SPS `EE` signal.
`WWE_art`	1	Input	Artificial `WWE` signal.
`WE_art`	1	Input	Artificial `WE` signal.
`EE_art`	1	Input	Artificial `EE` signal.
`WWE_out`	1	Output	Selected `WWE` signal.
`WE_out`	1	Output	Selected `WE` signal.
`EE_out`	1	Output	Selected `EE` signal.
`SPS_in_spill_number`	32	Output	Received SPS spill signals.

Table 8: Summary of In/Out signals of the Selector entity.

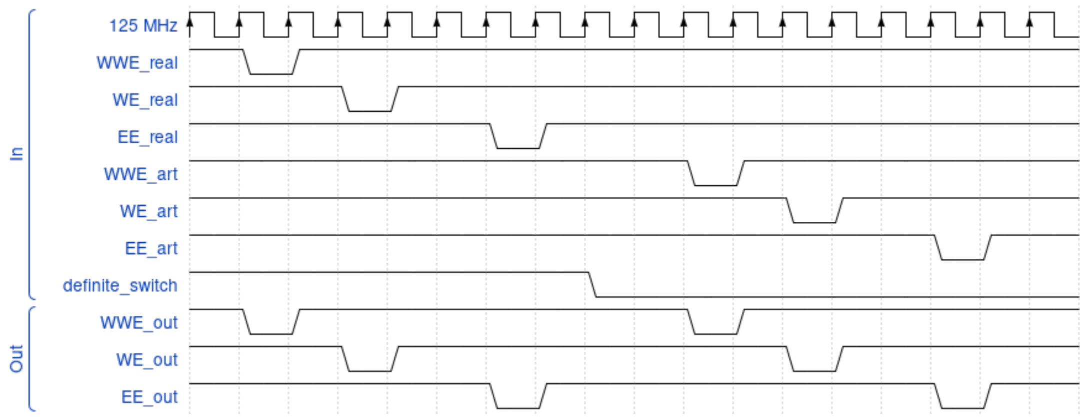


Figure 31: Off-scale Selector's wave representation.

The actual logic of the Selector is extremely simple. It just consists in a single process, activated by the rising edge of the FPGA clock signal. By means of an if statement, the Real signals are selected for the `definite_switch`=1 case, while the Artificial ones are returned in output in the opposite `definite_switch`=0 configuration. To solve initialization issues, since the `definite_switch` signal is looped back from the downstream FSM entity, if none of the above conditions are satisfied, the output `WWE,WE,EE` instructions are constantly set to 1 (which means that, due to the inverted logic, a 0 value is outputted).

In Fig.31 the wave representation of the logic's behaviour is reported. It's important to highlight the fact that the timing of the variation for the `definite_switch` signal is not randomly depicted in the picture. Since it is originally set by the control room's user, it could be possible a priori to have a sudden change of its value during the on-spill phase. Nevertheless, the superposition of different spill structures in the output signals would certainly result in a DAQ failure, since it could not process two subsequent `WWE` or `WE` instructions. The treatment of this issue and the proper timing of the `definite_signal` exchange is one of the main purposes of the FSM, therefore from the Selector's point of view the change of the `definite_switch` value will always conveniently happen after `EE` and before `WWE` instructions.

4.4.3 Finite State Machine

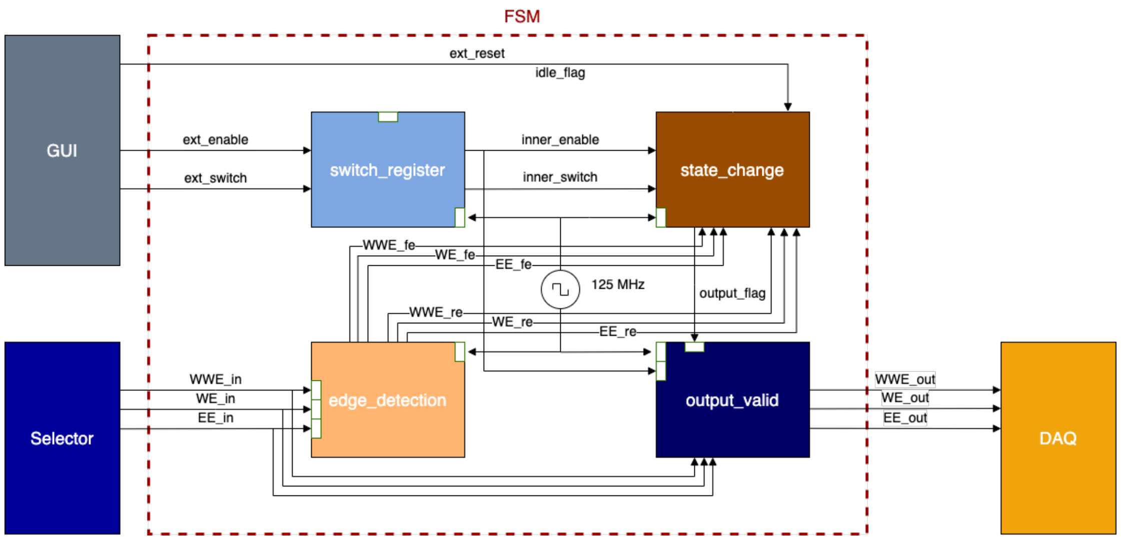


Figure 32: Processes inside Manager's FSM component.

The internal architecture of the FSM entity is made out of four concurrent processes: `switch_register`, `state_change`, `edge_detection` and `output_valid`. They all have specific purposes and they eventually contribute to the correct functioning of the most complex entity in the firmware's architecture. The connections between the cited processes are reported in Fig.32, even though the whole structure of the FSM component is not represented in detail.

state_change exchanges many other signals with different entities in the logic in order to perform specific measurements, but they are not reported in this phase to simplify the description of the circuit (see Tab.9). In the picture, the signals pointing to the white boxes are part of the sensitivity list of the process that receives them, causing an asynchronous functioning of the whole circuit.

Signals	Bits	In/Out	Description
clock	1	Input	Periodic: 125 MHz.
ext_enable	1	Input	Boolean for enabling.
ext_reset	1	Input	Boolean for resetting.
ext_switch	1	Input	switch value set from GUI.
WWE_out	1	Output	WWE signal forwarded to DAQ.
WE_out	1	Output	WE signal forwarded to DAQ.
EE_out	1	Output	EE signal forwarded to DAQ.

Table 9: Summary of In/Out signals of the FSM entity within the Manager's component.

The switch_register process simply works as a controlled gate for the incoming ext_enable and ext_switch. It is activated by the 125 MHz clock and by the idle_flag 1-bit signal generated by the state_change process. When idle_flag=1 the circuit is in the idle state, meaning that since it is in the off-spill phase it is safe to read the incoming switch and enable values and store them in their internal versions inner_switch and inner_enable variables. These will eventually be processed by state_change and they will not be allowed to change again until the next idle state, since their possible mid-spill variation would result in the superposition of instructions coming from different sources (see Tab.10).

Signals	Bits	In/Out	Description
clock	1	Input	Periodic: 125 MHz.
ext_enable	1	Input	Boolean for enabling.
ext_reset	1	Input	Boolean for resetting.
inner_enable	1	Input	Boolean for enabling.
inner_reset	1	Input	Boolean for resetting.

Table 10: Summary of In/Out signals of the switch_register process (Fig.32).

The main purpose of the `edge_detection` logic instead is to guarantee the correct functioning of the `state_change` process, where the actual FSM structure is implemented. It is activated by the FPGA clock and by the incoming `WWE_in,WE_in,EE_in` instructions. At each `clock` rising edge, the instructions are read and stored in their delayed versions `WWE_d,WE_d,EE_d`, which are then compared with the input signals at every activation of the process. As it is represented in Fig.33, the output signals have the structure of impulses, properly timed in order to highlight the presence of falling and rising edges of the instruction values. The main reason why the delayed signals are stored is that it is fundamental to compare the current values of the signals with the ones they were assigned to at the previous clock cycle: in this way, if the comparison detects any difference in them, we can be sure that a rising or lowering edge was present. Since the process is activated at every variation of the clock but the delayed signals are read only at its rising edges, the maximum latency reachable for the transmission of the rising/falling edge information corresponds to a semi-period of the FPGA clock, namely 4 ns. The logic implemented in the circuit follows the formulae $falling\_edge = delayed \wedge \overline{input}$ and $rising\_edge = \overline{delayed} \wedge input$, which have been applied separately to the three instruction signals `WWE,WE,EE`. In Fig.33 the wave behaviour of said process is presented, considering only the `WWE` signal and not the complete set of instructions (see Tab.11).

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`WWE_in`	1	Input	Selected `WWE` signal.
`WE_in`	1	Input	Selected `WE` signal.
`EE_in`	1	Input	Selected `EE` signal.
`WWE_fe`	1	Input	`WWE` falling edge.
`WE_fe`	1	Input	`WE` falling edge.
`EE_fe`	1	Input	`EE` falling edge.
`WWE_re`	1	Input	`WWE` rising edge.
`WE_re`	1	Input	`WE` rising edge.
`EE_re`	1	Input	`EE` rising edge.

Table 11: Summary of In/Out signals of the `edge_detection` process (Fig.32).

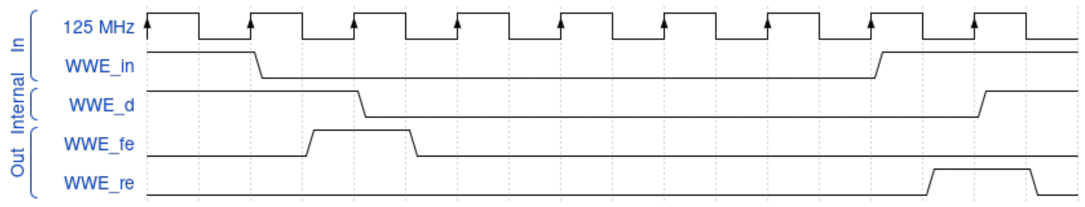


Figure 33: Off-scale `edge_detection` wave representation for `WWE` instruction.

The `output_valid` process works as a controlled gate for outputting the instruction signals `WWE_out,WE_out,EE_out` and eventually send them to the DAQ. It is activated by the rising edge of the 125MHz clock, the `inner_enable` and the `output_flag` signals, the latter being generated by the `state_change` process. At each rising edge of the clock, if the condition `inner_enable`=1 and `output_flag`=1 is satisfied, the `WWE_in,WE_in EE_in` instructions are directly forwarded respectively to `WWe_out,WE_out,EE_out`, otherwise the output values are kept constant at 1 in order to avoid working with undefined signals. In this way the `state_change` process can regulate the signals sent to the DAQ, properly timing them according to the states in which the logic is found (see Tab.12).

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`inner_enable`	1	Input	Boolean for enabling.
`output_flag`	1	Input	Flag for enabling output.
`WWE_in`	1	Input	Selected `WWE` signal.
`WE_in`	1	Input	Selected `WE` signal.
`EE_in`	1	Input	Selected `EE` signal.
`WWE_out`	1	Output	`WWE` signal forwarded to the DAQ.
`WE_out`	1	Output	`WE` signal forwarded to the DAQ.
`EE_out`	1	Output	`EE` signal forwarded to the DAQ.

Table 12: Summary of In/Out signals of the `output_valid` process (Fig.32).

The actual `state_change` process follows the diagram reported in Fig.34, where the transition between the states `idle, pre_warning, measure, stop` is described. It is activated exclusively by the 125MHz clock and it is initialized with the `idle` state. This represents the one state in which the logic is found in between different spills and it will be described in detail in the following sections. In this phase the majority of the circuit's safety checks are performed, the measurements are reset, the `definite_switch` value is chosen, the `idle_flag` is set to 1 and the output is blocked via `output_flag`=0. The next state is only reachable when the `WWE_fe`=1 comes from the `edge_detection` process, signalling the beginning of the spill with the pre-warning instruction (see Tab.13).

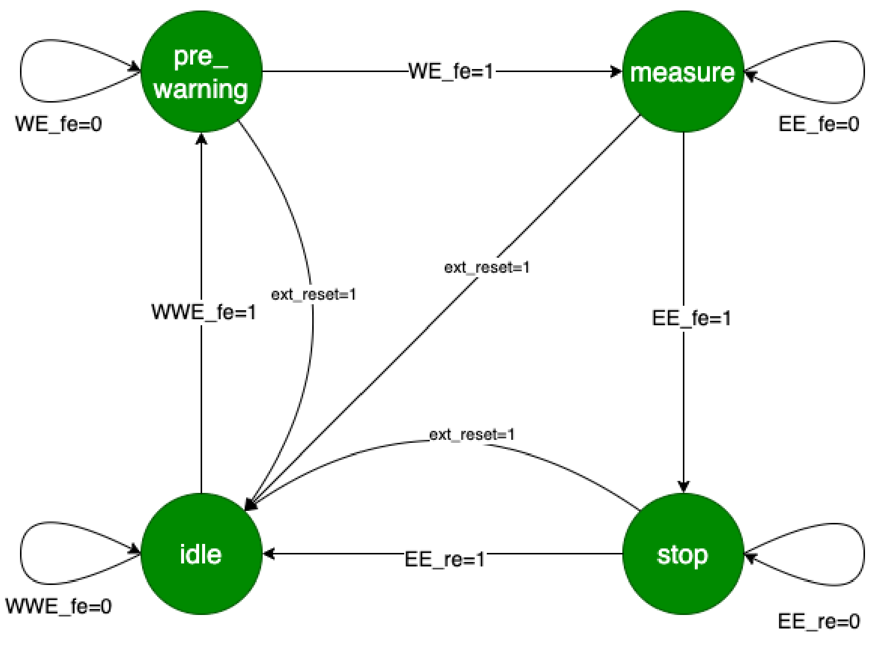


Figure 34: Logic Diagram of the FSM's process `state_change`.

In this state the machine is still waiting for the the arrival of the `WE` instruction, signalled by `WE_fe`=1, which states the actual beginning of the on-spill phase: when this happens the logic is moved to the `measure` state, where the DAQ effectively starts collecting data from the detectors. When the particle spill reaches its end, the `EE_fe` impulse moves the machine to the final `stop` state. The latter is responsible for sending in output the measured values collected throughout the spills and it resets several spill-specific flags which are no longer useful. The machine is moved back to the `idle` state when `EE_re`=1, making this state last only for a short amount of time with respect to the previous ones. While the others are present in between instructions, therefore on the seconds scale, this one is only available for the duration of the single `EE` instruction, corresponding to a SIGNAL_LENGTH value in the scale of micro seconds. The state vs instructions relation is reported in Fig.35, even though clearly off-scale due to graphical impossibilities. A more truthful representation of the impulses can be seen in Fig.51.

Signals	Bits	In/Out	Description
clock	1	Input	Periodic: 125 MHz.
inner_enable	1	Input	Boolean for enabling.
inner_switch	1	Input	Fixed switch signal.
ext_reset	1	Input	Boolean for resetting.
WWE_fe	1	Input	WWE falling edge.
WE_fe	1	Input	WE falling edge.
EE_fe	1	Input	EE falling edge.
WWE_re	1	Input	WWE rising edge.
WE_re	1	Input	WE rising edge.
EE_re	1	Input	EE rising edge.
output_flag	1	Output	Flag for enabling output.

Table 13: Summary of In/Out signals of the state_change process (Fig.32). A complete record of the signals is reported in Tab.6.

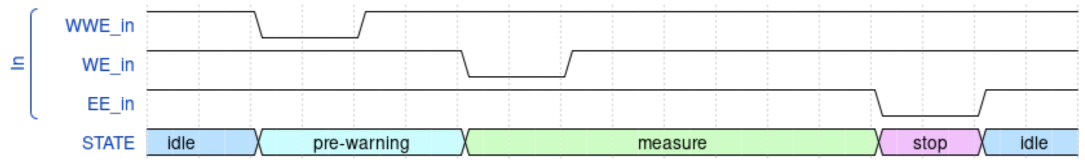


Figure 35: Off-scale representation of instructions vs state_change states.

4.5 ENABLE, SWITCH AND RESET

One of the first requirements of the firmware was to implement the functioning of the ext_enable, ext_switch and ext_reset signals. As it has been stated already, the first is meant to work as an on/off command for the whole circuit, allowing or not its communication with the DAQ, while the second signal represents which instruction source has to be selected by the logic. The last one was instead implemented to give the possibility to completely reset the whole hardware from the control room. Reaching the `state_change` entity, these signals need to be analyzed with particular attention, since they represent the orders from the GUI which could have a larger impact on the correct functioning of the whole circuit and, as a consequence, on the DAQ.

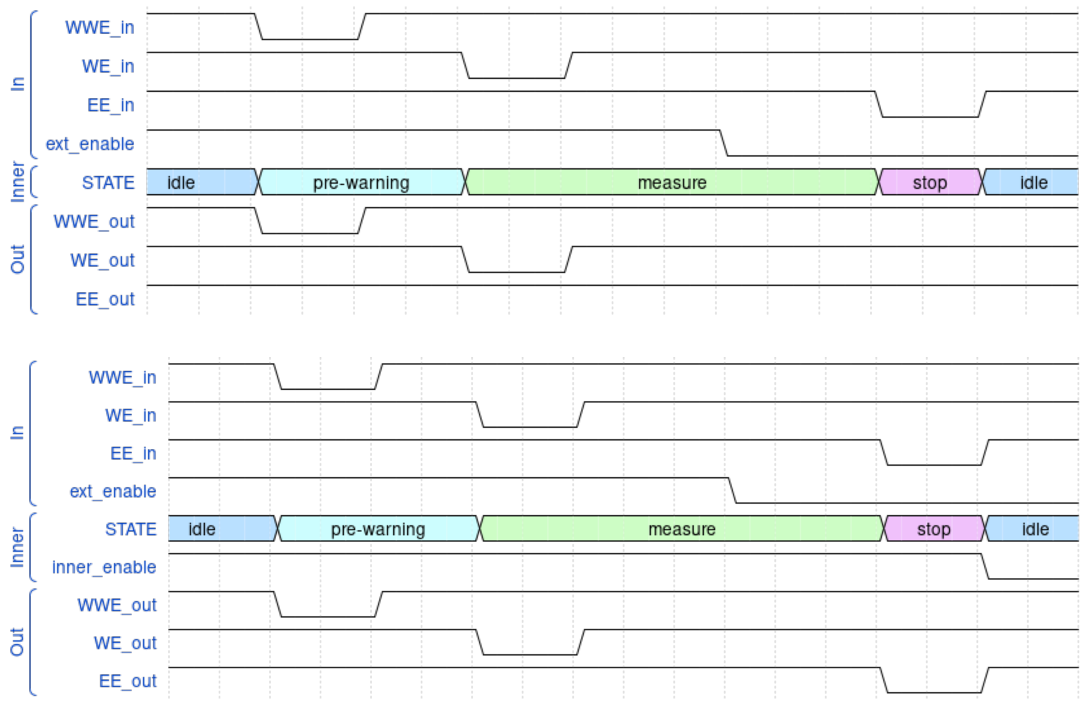


Figure 36: Top: wrong `ext_enable` behaviour, Bottom: correct `inner_enable` behaviour.

Considering the `ext_enable` signal, some care has to be applied when considering its consequences on the instruction transmission. Its main purpose is to switch on and off the whole circuit and the forwarding of the instructions to the DAQ. The timing of such intervention has to be micromanaged. The commands from the GUI are always sent regardless of the spill phase, therefore the circuit could potentially be switched off mid-spill, causing a partial generation of the instructions. Since this would correspond to a certain DAQ failure, the logic must be able to prevent this from happening (Fig.36).

Thanks to the switch_register, the ext_enable signal is read and transferred to the inner_enable only when idle_flag=1. During its states loop, the state_change component keeps referring to the stored inner_enable value, allowing its update only at the next idle state. In this way, the rising or lowering edge of the ext_enable signal happening mid-spill are only processed at the beginning of the next bunch of particles. The downside of this behaviour is that it can potentially represent a data loss in the case in which the ext_enable switches from 0 to 1 in the middle of the spill. Nonetheless, when the opposite situation occurs, it is able to save the data of the last spill, waiting for it to end before processing the ext_enable=0 information. This precaution represents a huge gain for the experiment, since it guarantees a constant functioning of the DAQ. At last, the inner_enable value directly acts on the output_valid entity, allowing or blocking the forwarding of the instructions to the DAQ.

A similar treatment has to be used with the ext_switch value, even though more complications occur due to the different nature of the signal (Fig.37). Just like the ext_enable, also the ext_switch is transferred to its internal version inner_switch thanks to the switch_register entity, exclusively during the idle state. In this way the switch value received before the particles start coming will be kept constant throughout the whole spill. Nonetheless, with the introduction of the Timeout service (which will be explained in detail in the section 4.5)the necessity of storing another processed version of the switch value, namely definite_switch, becomes fundamental. This signal is most of the time just another identical copy of inner_switch, initialized during the idle state. The only moment in which they represent different values is when the Timeout service comes into action, forcing the selection of the Artificial signals instead of the Real ones (inner_switch=1 and definite_switch=0).

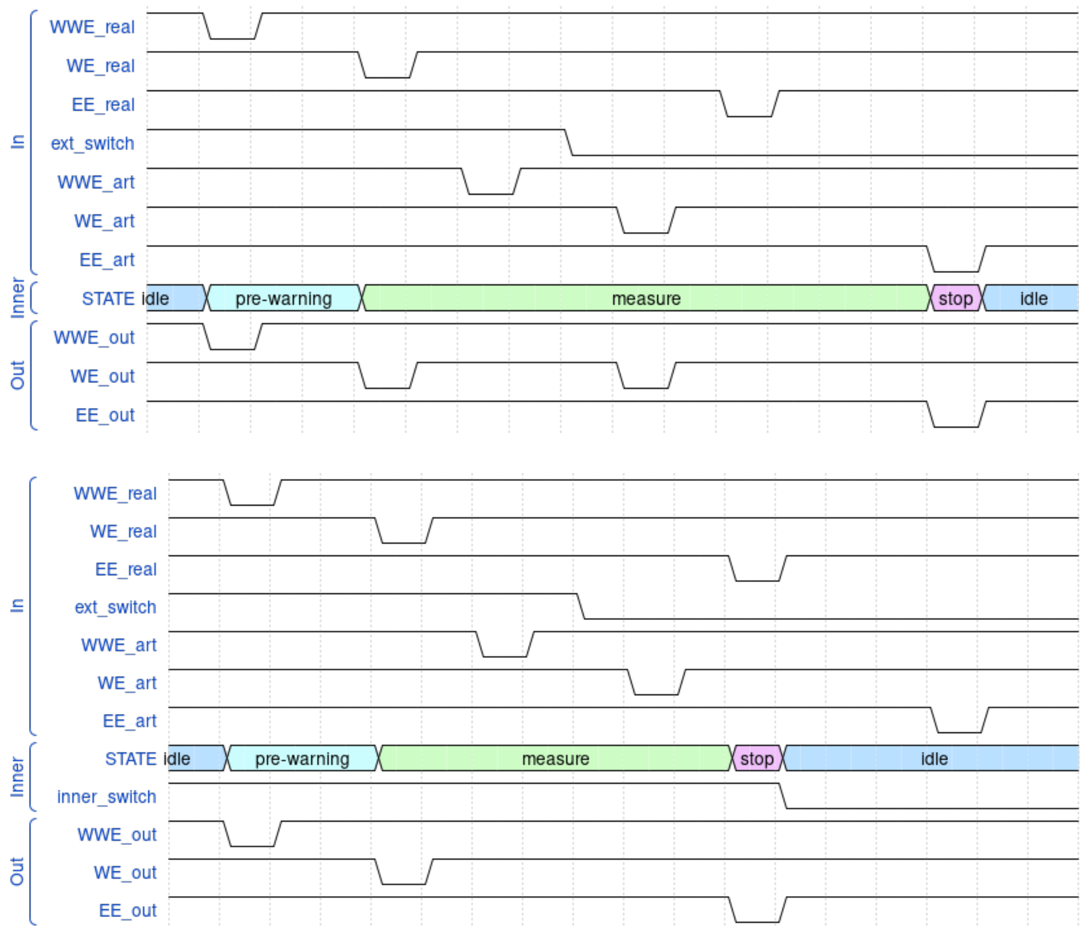


Figure 37: Top: wrong `ext_switch` behaviour, bottom: correct `inner_switch` behaviour

On the other hand, the `ext_reset` is not paired with any internally registered signal. Its purpose is to work as a hard shutdown for the whole circuit, regardless of the spill phase in which the logic is found. The possibility to completely reset the whole hardware had to be implemented, even though it is clearly an action which should be avoided. If the value of the `ext_reset` signal is raised to 1 from the GUI, all the counters in the circuit are reset, together with all the data stored for the spill measurements. At the same time, the `state_change` component is re-initialized to the idle state, regardless of the current spill phase (see Fig.34). Since in this way the crash of the DAQ would be certain, for the optimal functioning of the firmware `ext_reset` will always be set at a constant 0 value.

4.6 Measurements

Together with the development of the Artificial Spill Generator firmware, a series of important measurements were requested by the collaboration. The real-time monitoring of the main spill parameters can give important insights on the experiment's state to the control room during data taking runs. The logic is therefore enriched with a series of components dedicated to these measurements, which are eventually transmitted in output and transferred to the GUI via IPBUS protocol. First of all, the measurement of the effective distances between `WWE-WE` and `WE-EE` instructions is performed. Moreover the `cycle_time`, which is the time distance between different spills measured at each `WWE`'s falling edge, is also continuously monitored. Eventually, for statistical purposes, the circuit collects the number of incoming SPS spill signals and of Real and Artificial instructions effectively forwarded to the DAQ. The following sections contain the detailed explanation of how these measurements are performed and inserted in the previously described logic[7].

4.6.1 Measurement of signal distances

The effective measurement of the time distances between `WWE-WE,WE-EE` instructions provide useful information for both monitoring the state of the experiment and accelerator performance, enriching the data taking process. These values can be useful for hardware synchronization and for proper spill timing visualization, but they can also provide additional material for the event storage and reconstruction. For example, the `WE-EE` interval fixes the beginning and the end of the particles bunches, therefore it can be used as a cardinal trigger information for selecting the appropriate interaction events within the flat-top region of the spill.

[7]The results of the measurements explained in the following paragraphs are eventually reported and visualized in real time in the control room's GUI. Due to the variability of such values (depending on the beam states and SPS timings) and hence to their non fully informative feature, they are not reported in this work.

In order to measure the effective distances between `WWE-WE` and `WE-EE`, a dedicated `counter` entity has been exploited. It receives in input the 1 kHz clock signal, together with the `enable,validate,reset` 1-bit values and it eventually outputs the 32-bit `STD_LOGIC_VECTOR` with the counted result. Its internal logic is extremely simple: it is made out of two concurrent processes, one dedicated to the effective count and one which works as a gated output. The former is activated as usual by the rising edge of the `clock` signal and it is responsible for resetting the count when `reset`=1 and enabling it when `enable`=1, storing the temporary result into an internal 32-bit signal. The latter instead is used to return the final result in output only when `validate`=1.

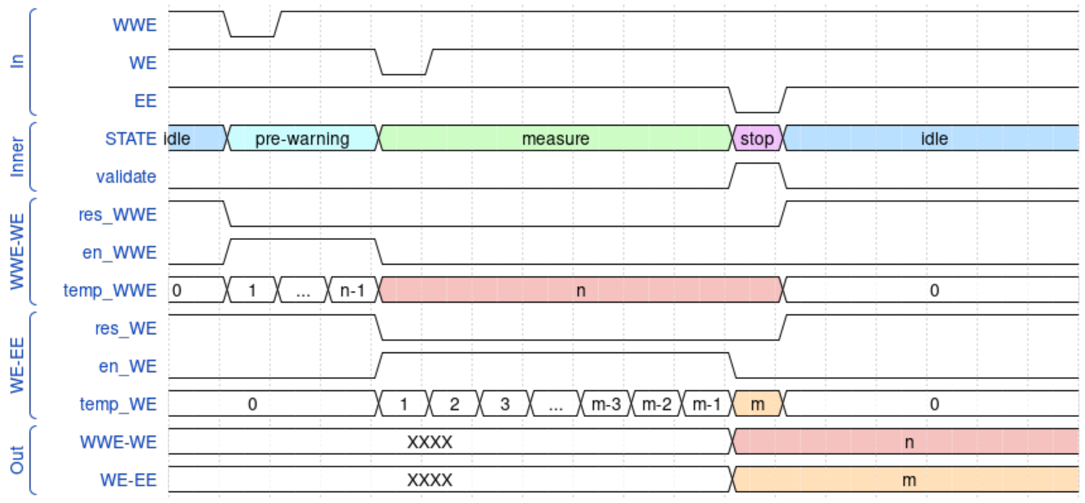


Figure 38: Wave behaviour of instruction distances measurement.

Two copies of such entity are inserted inside the Manager's FSM component, in order to compute separately the `WWE-WE` and `WE-EE` time distances (see Fig.43). The `validate` signal has been constantly set to 1 for the simulation phases, since it was useful to retrieve the counted result in real time, while in the final version of the firmware, it is raised to 1 only when the ultimate result is ready. The 32-bit format for the count of clock periods happens to be enough for the measurement limits for the COMPASS experiment and SPS cycle: a 32-bit binary number covers the decimal representation up to a $2^{32} - 1 = 4294967295$ value, corresponding to the maximum number of countable clock cycles. The period of the 1 kHz clock being 1 ms, the maximum time difference measurable is $4294967295 \cdot 8x10^{-9} = 4294967.295$ s which abundantly covers the nominal value for the SPS cycle time (see Sec.4.5.2).

Together with the `WWE_counter, WE_counter` entities, dedicated internal signals have been defined in order to properly time the `state_change` phases with the said counters. In particular, in the `idle` state all the counted results are reset to 0; only when the machine is found in the `pre-warning` state, so when the first lowering edge of `WWE` occurs, the `WWE_counter` is enabled via `res_WWE=0, en_WWE=1`[8]. At the next state change, the counter is stopped with `en_WWE=0` but not reset and the `WWE-WE` time distance result is stored in memory. The same behaviour is implemented for the measurement of the `WE-EE` distance, exploiting of course a different counter component. Both retrieved values are sent in output during the `stop` state via `validate=1` and the counters are eventually reset (Fig.38).

4.6.2 Cycle time

The so called cycle time is another fundamental parameter for the monitoring of the incoming spill structure: it is defined as the time distance between `WWE` instructions belonging to consecutive spills. Firstly, its measurement happens to be extremely important to compare the effective and nominal SPS cycle time values, in order to be sure that the information received from SPS corresponds to what is actually received by the experiment. Secondarily, such value can work as a checksum with other event-specific time coordinates stored during data taking, which are fundamental for the final event building and analysis. Moreover, when the artificial signals are being used, the cycle time measurement represents a good verification method for the correct functioning of the whole firmware.

[8]As it can be seen in Fig.43, the `enable` signals of the counters are convoluted with the `inner_enable` value by means of an AND port. In this way, said entities need to be switched on by both the control room and the correct `state_change`'s signals.

Such measurement is performed by exploiting the `counter` entity described before, even though for this case additional difficulties must be taken into consideration (see Fig.43). The cycle time represents by definition the time interval that the logic takes to perform a whole loop of states, using as a reference the lowering edge of the `WWE` signal (see Fig.39). This measurement overlaps between two different but contiguous spills, therefore the logic must be adapted in order not to concentrate only on the spill structure, as all the entities described before are doing, but also on storing the time coordinates of different spills happening one after the other. Together with these improvements, the circuit must also take care of the possible `ext_switch` or `ext_enable` differences that might occur between consequent spills, since they could cause a wrong measurement of the cycle time.

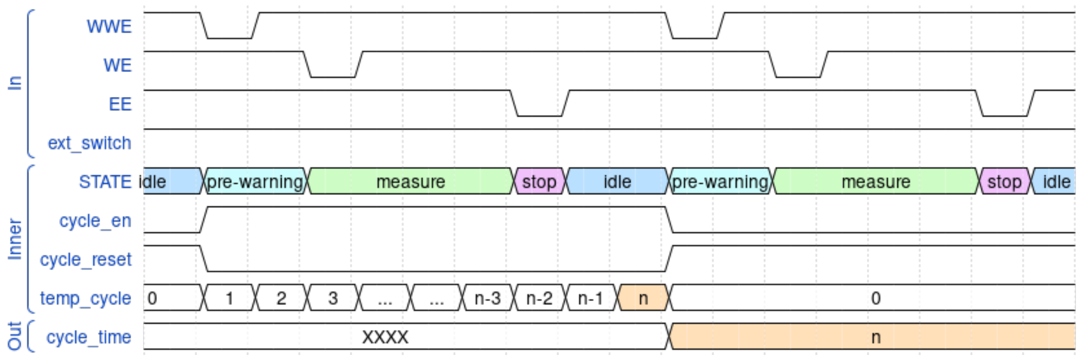


Figure 39: Off-scale wave representation of `cycle_time` measurement with constant `switch` and `enable` signals.

In order to solve the issue of a possible sudden change of `ext_switch` or `ext_enable`, an internal version of such signals is defined: `cycle_switch`, `cycle_enable`. In every state of the Manager's FSM, a comparison between the inner and external versions of the above signals is performed; if any difference is retrieved, the `cycle_counter` component is reset via `res_cycle=1` and `en_cycle=0`. This implementation is necessary because, in the case of two subsequent spills belonging to different sources, the measurement of the cycle time would loose its reason to exist. The distance between `WWE` instructions would represent a time interval with no specific physical meaning, therefore at every variation of the switch, the cycle time result returned in output is reset to 0. A similar behaviour is expected for sudden changes of the `enable` signal: even though the circuit always keeps track of the incoming instructions, it must be able to measure the cycle time only if the instructions are returned in output too. In order to be sure not to store any cycle time values when the `WWE,WE,EE` signals are not processed by the DAQ, at any rising or lowering edge of the `ext_enable`, the logic must be able to return a coherent result and reset itself (see Fig.40).

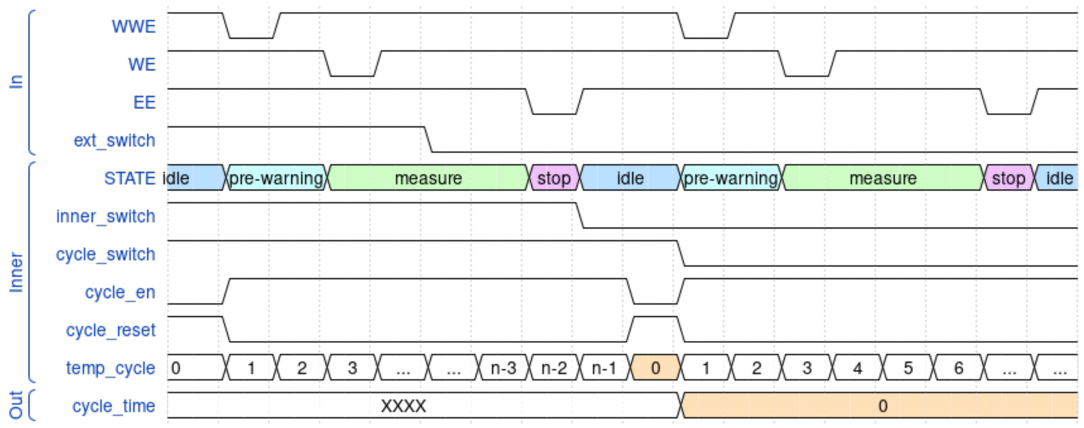


Figure 40: Reset `cycle_time` measurement due to exchange of instruction source.

The `cycle_enable` and `cycle_switch` signals are only read during the `pre-warning` state, when the beginning of the spill is certain and the lowering edge of `WWE` has already occurred. In this context the counter is enabled via `res_cycle=0`, `en_cycle=1` and is kept active for the whole loop of states, if none of the switch or enable values change. Even in this case, the `validate` signal was kept constant at 1 for the simulation phases, in order to have a real-time insight of the functioning of the circuit. In the final version instead, the validate signal is raised to 1 only in the `idle` state, when the `WWE` lowering edge signal has already arrived, but the logic has not yet been moved to the `pre-warning` state. In this way, the difference in time between the stop and start instructions for the counter is only one clock cycle (See Fig.41).

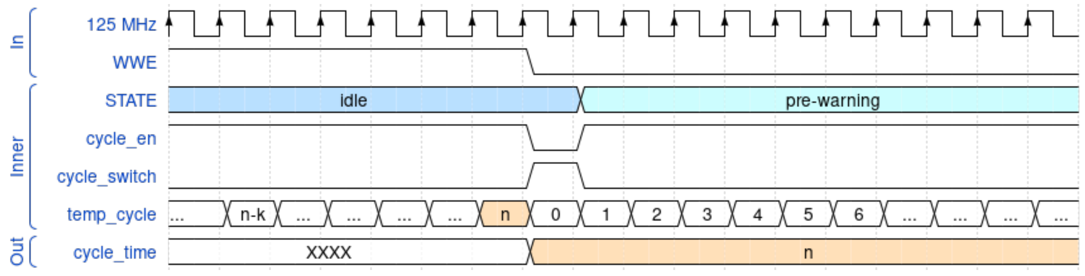


Figure 41: Close-up wave behaviour of cycle time counter w.r.t FSM states.

4.6.3 Number of spills

Signals	Bits	In/Out	Description
`clock`	1	Input	Periodic: 125 MHz.
`enable`	1	Input	Boolean for enabling.
`ext_reset`	1	Input	Boolean for resetting.
`WWE`	1	Input	Incoming `WWE` signal.
`spill_result`	32	Output	Result of spill number count.

Table 14: Summary of In/Out signals of the `spill_counter` entity.

For statistical purposes, the collaboration wanted to be able to monitor the overall number of complete spills received and processed by the experiment. Such count is performed separately, by considering the amount of spills received from SPS and the number of Real and Artificial instructions eventually processed by the DAQ. Therefore a dedicated `spill_counter` entity has been defined and inserted in three different copies within the circuit (see Tab.14).

Said entity receives in input the 125 MHz clock, the `enable` and `reset` signals and only a `WWE` instruction, used to state the beginning of a spill. Two concurrent processes are defined within the `spill_counter` entity: the usual `edge_detection` process highlights the presence of a falling or rising edge of the `WWE` signal, while the counting is performed in the parallel process. Here, the spill number is increased whenever a falling edge of `WWE` is detected, provided that the `reset` and `enable` values are found respectively in 0 and 1. As usual, the opposite configuration would simply result in the complete reset of the counter, setting the `spill_number` back to its initial 0 value. Eventually the result is returned in output in the form of a 32-bit `STD_LOGIC_VECTOR`.

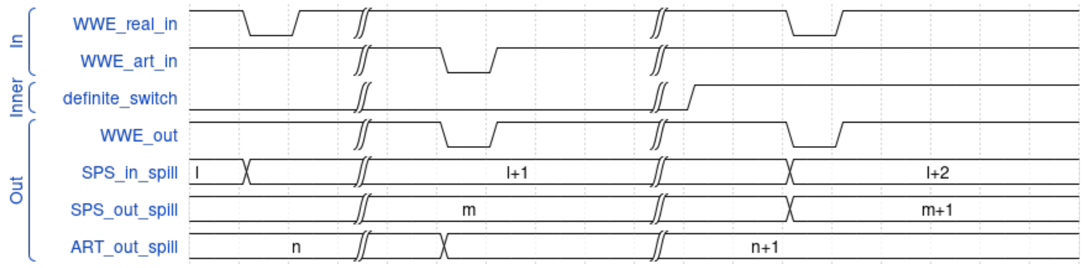


Figure 42: Spill's number counting with respect to received and processed `WWE` instructions.

Since the behaviour of such entity is extremely simple, the difference in the nature of the required measurements has to be found in the position of said component within the circuit, rather than in its internal logic (see Fig.43). For example, the measurement of the incoming Real signals has to be performed in the Manager's Selector component (see Fig.30). In this case, the `WWE_real` signal is intercepted upstream from the selection component, in which the choice between SPS or artificial signals is performed. The counted result is eventually returned to the GUI, bypassing the rest of the firmware's logic.

For the processed SPS and Artificial signals, the components are inserted downstream from the Manager's FSM. The `WWE` signal sent to the DAQ is convoluted to the said entities, in order to make sure to count the processed spills. The differentiation between Artificial and Real signals is performed exploiting the `definite_switch` value, received in input by the two `spill_counter` entities as an `enable` value. Since `definite_switch` represents the source of information effectively processed by the FSM (respectively equal to 1 for SPS and 0 for internal signals), it can be used as a pure `enable` for the Real signal selection, while its logic opposite can enable the selection of the Artificial signals (see Fig.42).

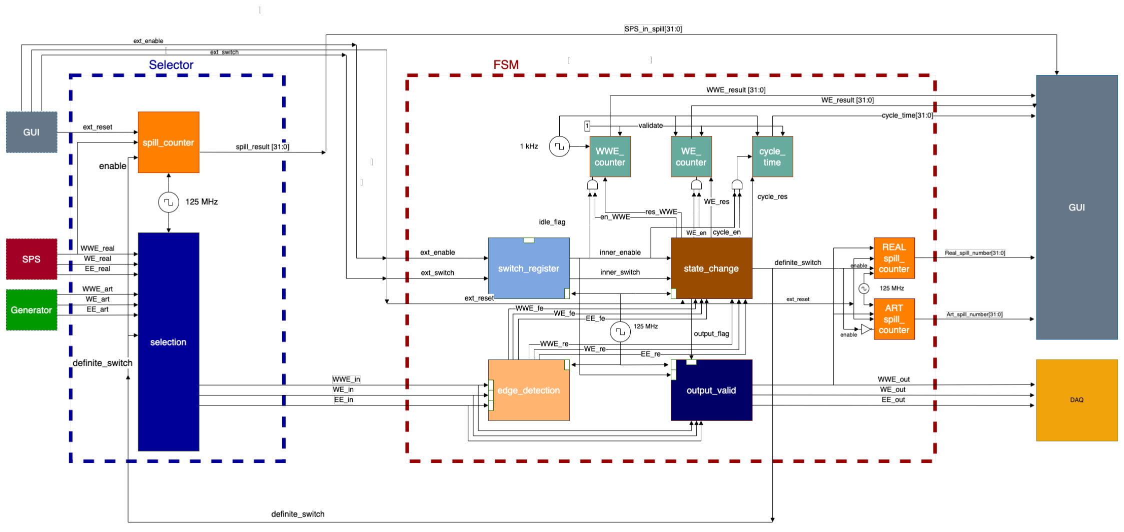
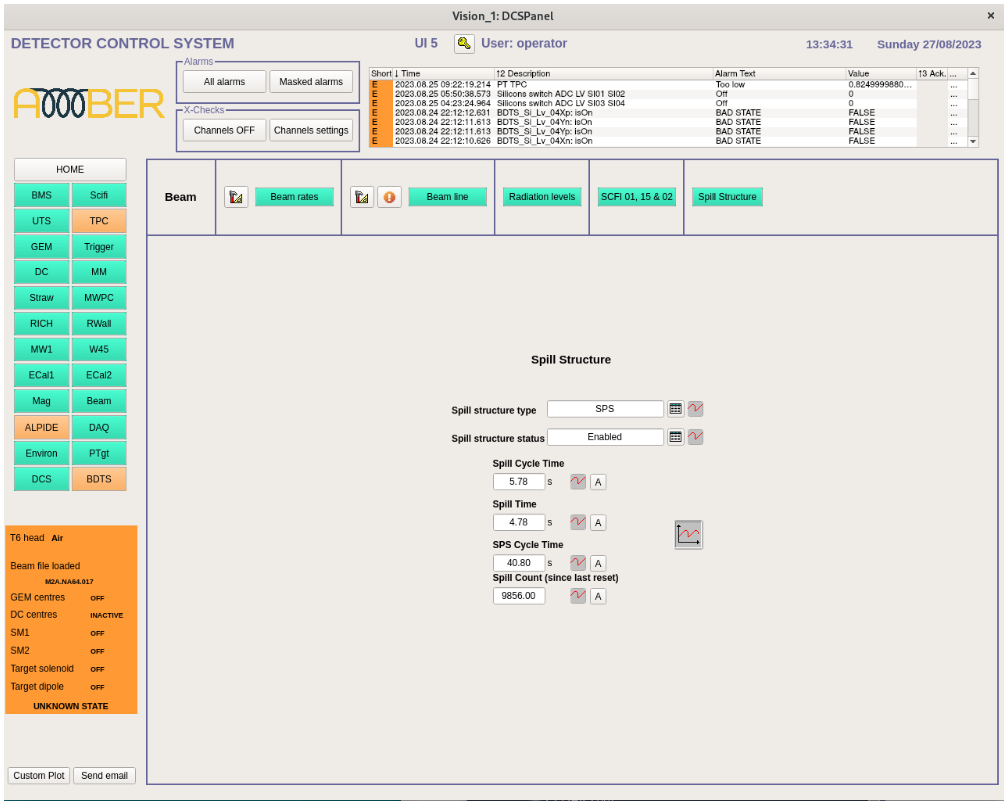


Figure 43: Detailed FSM's architecture with measurement tools.

The results of the measurements explained above can be visualized in real time from the control room's GUI. In the following images, two screenshots of the Detector Control System (DCS) are reported, in order to allow the reader to properly picture the look of the experiment's interface devoted to the DAQ control and monitoring. In particular, Fig.44 shows the panel in which the main parameters are set and visualized. At the center of the picture the "spill structure type" (corresponding to the firmware's `switch` value) can be selected, choosing between SPS or ART sources. The "spill structure status" represent instead the `enable` signal. The values reported below are those measured and retrieved by the Artificial Spill Generator module: "Spill Cycle Time" reports the summed `WWE-WE,WE-EE` values, while the "Spill Time" only reports the `WE-EE` time distance. Eventually, the "SPS cycle time" is the measured `cycle_time` value and the "Spill Count" simply reports the result of the addition of the `SPS_out_spill_number` and `ART_out_spill_number`.



Short	↓ Time	↑2 Description	Alarm Text	Value	↑3 Ack.	...
E	2023.08.25 09:22:19.214	PT TPC	Too low	0.8249999880...		...
E	2023.08.25 05:50:38.573	Silicons switch ADC LV SI01 SI02	Off	0		...
E	2023.08.25 04:23:24.964	Silicons switch ADC LV SI03 SI04	Off	0		...
E	2023.08.24 22:12:12.631	BDTS_Si_Lv_04Xp: isOn	BAD STATE	FALSE		...
E	2023.08.24 22:12:11.613	BDTS_Si_Lv_04Yn: isOn	BAD STATE	FALSE		...
E	2023.08.24 22:12:11.613	BDTS_Si_Lv_04Yp: isOn	BAD STATE	FALSE		...
E	2023.08.24 22:12:10.626	BDTS_Si_Lv_04Xn: isOn	BAD STATE	FALSE		...

T6 head Air

Beam file loaded M2A.NA64.017

GEM centres	OFF
DC centres	INACTIVE
SM1	OFF
SM2	OFF
Target solenoid	OFF
Target dipole	OFF

UNKNOWN STATE

Spill Structure

Spill structure type: SPS

Spill structure status: Enabled

Spill Cycle Time: 5.78 s

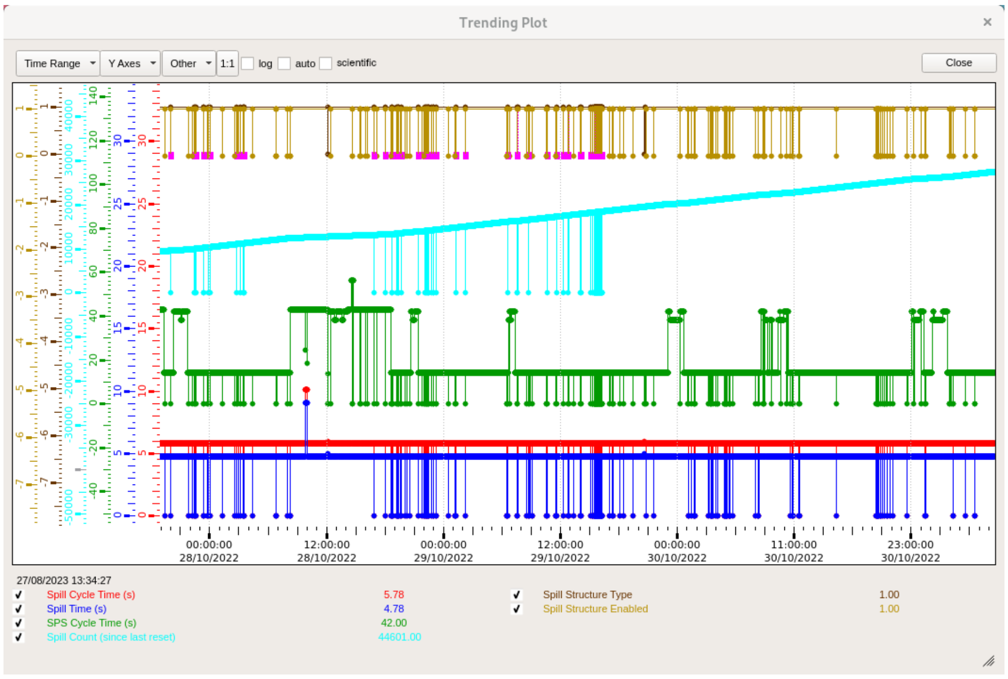
Spill Time: 4.78 s

SPS Cycle Time: 40.80 s

Spill Count (since last reset): 9856.00

Figure 44: DCS Panel: `switch` and `enable` values are set, `WWE-EE`, `WE-EE`, `cycle_time` and `SPS_out+ART_out` are read.

In Fig.45 instead, the above values are reported as a function of time. It is particularly interesting to notice how the Spill Cycle time and Spill Time remain constant most of the time, reflecting the fact that the spills delivered at COMPASS experimental hall maintain always the same structure. A different behaviour is seen for the SPS Cycle Time, which can be discontinuous, due to the particle beam delivery to the other experiments at CERN M2 beam line, and for the Spill Count, which is of course constantly increasing.



27/08/2023 13:34:27

✓	Spill Cycle Time (s)	5.78	✓	Spill Structure Type	1.00
✓	Spill Time (s)	4.78	✓	Spill Structure Enabled	1.00
✓	SPS Cycle Time (s)	42.00			
✓	Spill Count (since last reset)	44601.00			

Figure 45: DCS Panel: spill parameters trending plot.

4.7 Timeout

The main purpose of the Artificial Spill Generator is to guarantee the optimal functioning of the DAQ, constantly providing the necessary spill synchronization signals `WWE,WE,EE`. In order to do so, the hardware module must be able to fully substitute the information received by SPS with the artificially generated one. It can occur that the transmission of the Real signals is for some reason interrupted or ill-processed, therefore the firmware was built with several safety measures that can properly manage this situation, without propagating the issues directly to the DAQ and causing its crash. In order to limit the loss of data and the possible system's failure, the logic is equipped with a monitoring counter that keeps track of the time passed between `EE` and `WWE` SPS instructions: if such measure exceed a fixed threshold, meaning that the SPS transmission is not being correctly received by the experimental hall, the firmware automatically switches to the artificial generator (see Fig.46) and the user is informed via the GUI of the DAQ (Sec.4.6.2).

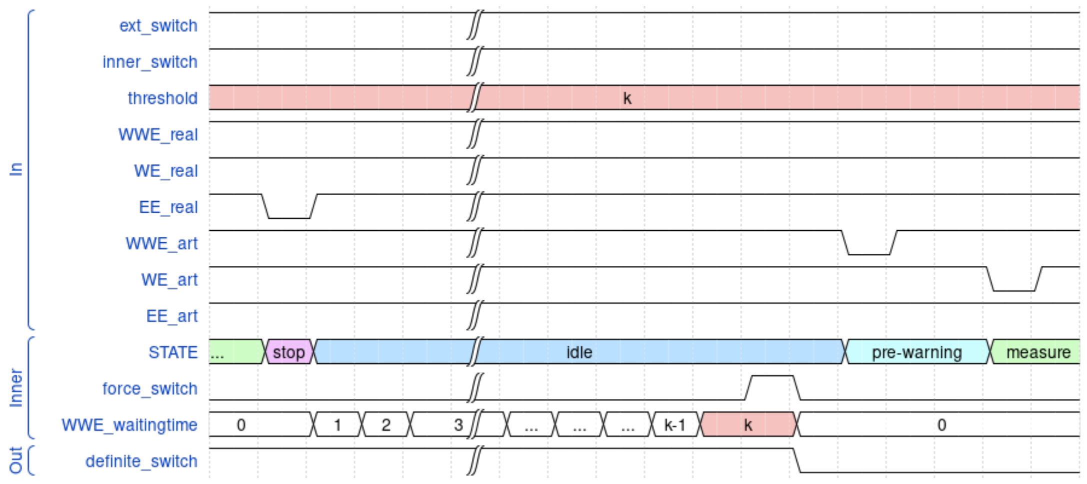


Figure 46: Wave behaviour of Timeout service.

This monitoring process is performed by a `variable_counter` entity (see Section 4.2.2), activated during the `idle` state of the Manager's FSM. This component is responsible for counting the `WWE` waiting time in terms of $8ns$ clock periods and confronting it with the value received in input from the control room. Whenever the counter reaches its maximum value, the component raises a 1-bit `force_switch` flag which consequently acts on the FSM, forcing the `switch` value from 1 (Real signals) to 0 (Artificial signals).

The logic of this process is described in detail in the following pages, paying particular attention to the variety of signals used and their different purposes. Since the overall `switch` signal's behaviour happens to be quite peculiar, a dedicated section has been inserted. The exploitation of a `dynamic_switch_register` for the IPBUS communication protocol completely changes the nature of such signal, providing a better logic versatility as well as a much greater circuit complexity. First of all, the `switch` circuit mapping is going to be described, connecting the nature of each signal with its processing entities. The specific implementation of the `dynamic_switch_register` and its logic behaviour are then reported, eventually concluding the explanation with the precise presentation of the timeout service's logic.

4.7.1 Switch signals

As explained before, the selection of the instruction source is performed by setting the `ext_switch` value from the GUI. Such value is then processed by the `switch_register` entity within the Manager's FSM component, which transfers the delayed information in the `inner_switch` signal exclusively during the `idle` state, *e.g.* in the off-spill phase. The `inner_switch` information is read by the `state_change` process and looped back to the Manager's Selector component, under the name of `definite_switch`, after having it properly processed and controlled with the timeout service. In this way, the `selection` process knows which source to read in order to look for the spill synchronization signals, sending the results down again to the `state_change` entity. At the same time, the `definite_switch` signal is also looped back to the dynamic switch register, which rules the exchange of information via IPBUS protocol (see Fig.47). This particular entity is both responsible for receiving the GUI orders and sending them to the firmware module and for reading the `definite_switch` value from the module and looping it back to the GUI. The detailed behaviour of such entity will be described in the Section 4.6.2.

By looping back the information to the GUI, the control room's user will know that the module has forced itself into reading the artificial signals regardless of the fixed input. In this sense, the insertion of the `dynamic_switch_register` has the aim of mediating between the firmware's independence and the user's lawful requirement to select a specific instruction source. Clearly, any other hierarchy would be counterproductive for the whole system. If the user had no limit in exchanging the `switch` value, he could easily cause the DAQ failure in several possible ways: for example by switching it mid-spill or selecting the Real instruction source when SPS is not correctly transmitting the signals. On the other hand, the Artificial Spill Generator cannot be completely independent in the selection of the instruction generators: the user must always be able to control the spill signals source, in order to switch between the different DAQ modes during data taking.

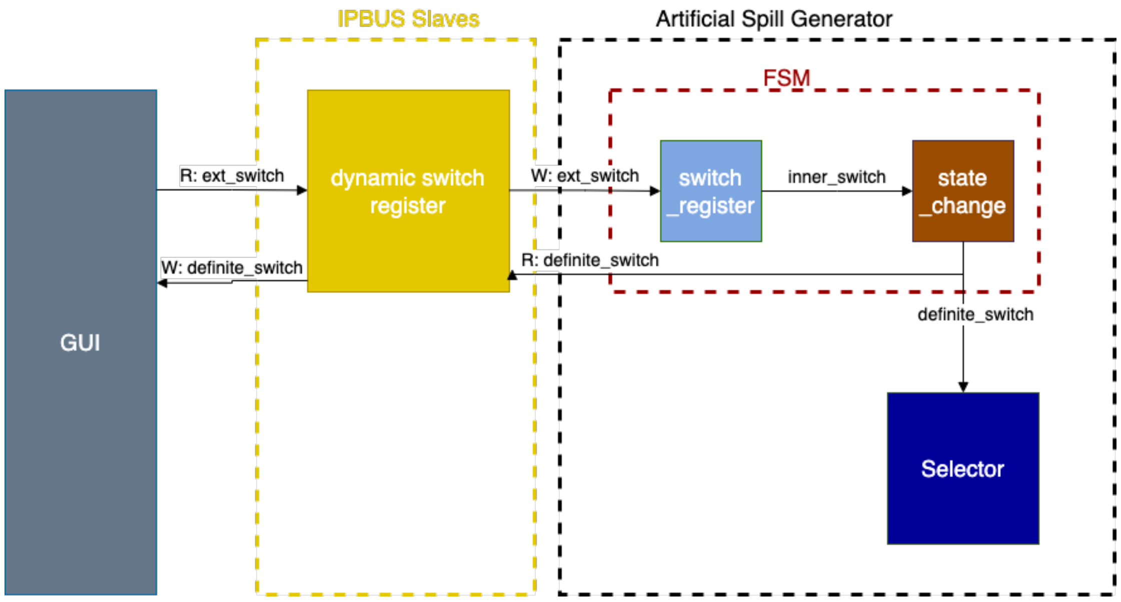


Figure 47: Complete loop of the `switch` signal's information.

4.7.2 Dynamic switch register

In the circuit, all the registers employed in the IPBUS communication protocol exploit the built-in package available in the CERN's IPBUS firmware [9]. In the code reported in Listing 1, two different types are defined. The `ipb_wbus` arrays represent the exchanged information from master to slave, while the `ipb_rbus` vector instead manages the exchange of information in the opposite direction. As it can be seen in the code, these two types present a different structure due to their different natures and purposes. They both rely on a 32-bit `STD_LOGIC_VECTOR` to deliver data and they present a similar purpose 1-bit value which comes before the data itself to enable the writing and the reading processes (respectively `ipb_strobe` and `ipb_ack`). Moreover the `ipb_rbus` is provided with a timeout service in the case in which the data are not properly received from the slaves.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
package ipbus is
-- The signals going from master to slaves
type ipb_wbus is
  record
  ipb_addr: std_logic_vector(31 downto 0);
  ipb_wdata: std_logic_vector(31 downto 0);
  ipb_strobe: std_logic;
  ipb_write: std_logic;
end record;
type ipb_wbus_array is array(natural range <>) of ipb_wbus;

-- The signals going from slaves to master
type ipb_rbus is
record
    ipb_rdata: std_logic_vector(31 downto 0);
    ipb_ack: std_logic;
    ipb_err: std_logic;
    ipb_timeout: std_logic_vector(15 downto 0);
end record;
type ipb_rbus_array is array(natural range <>) of ipb_rbus;

end ipbus;
```

Listing 1: IPBUS types [9]

For IPBUS protocol communication of the majority of the parameters involved in the firmware, `static_registers` are used, even though they only manage the exchange of information in one direction. They are deployed within the circuit either for translating the information set by the GUI to the firmware or for retrieving from the circuit the measured results and make them available in the GUI. In the logic there is a natural distinction between the values set from the user and the ones provided from the firmware itself. Signals like `ext_reset,ext_enable` and `max_WWE_waitingtime` are always and only fixed by the user and interpreted as orders by the circuit. On the other hand, all the measures performed (for example `WWE-WE,WE-EE` time intervals, as well as the `cycle_time` or the `SPS_in,SPS_out,ART_out` spill numbers) and the flags `wait_endofspill,timeout_flag` are always calculated by the circuit and delivered to the GUI. In Listing 2, the code that rules the behaviour of the `static_registers` is reported [1].

[1]The codes reported in Listings 1 and 2 are not written by the author of this thesis. They were provided by COMPASS collaboration, based on the CERN's firmware components publicly available online [9].

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;
library work;
use work.ipbus.all;
entity ipbus_reg is
  port(
    clk       : in  std_logic;
    reset     : in  std_logic;
    ipbus_in  : in  ipb_wbus;           --request
    ipbus_out : out ipb_rbus;           --answer
    read      : out std_logic;
    write     : out std_logic;
    update    : in  std_logic;
    d         : in  std_logic_vector(31 downto 0);
    q         : out std_logic_vector(31 downto 0)
    );
end ipbus_reg;
architecture rtl of ipbus_reg is
  signal reg : std_logic_vector(31 downto 0);
  signal ack : std_logic;
begin
    process(clk) begin
    if rising_edge(clk) then
      write <= '0';
      if ipbus_in.ipb_strobe = '1' and ipbus_in.ipb_write = '1' then
        reg   <= ipbus_in.ipb_wdata; --data are written
        write <= '1';
      elsif update = '1' then
        reg <= d;
      end if;
      if reset = '1' then
        reg <= (others => '0');
      end if;
      ipbus_out.ipb_rdata <= reg; --data are read
      ack                 <= ipbus_in.ipb_strobe and not ack;
    end if;
    end process;
      read                  <= ipbus_in.ipb_strobe and not ipbus_in.ipb_write;
      ipbus_out.ipb_timeout <= X"0000";
      ipbus_out.ipb_ack     <= ack;
      ipbus_out.ipb_err     <= '0';
      q                     <= reg;
end rtl;
```

Listing 2: Modified version of the IPBUS package [8].

The `dynamic_switch_register` has the unique characteristic to be able to work at the same time in both communication directions. It is only used to provide the necessary behaviour of the `switch` signal, which means that in its default mode it must be able to read the fixed value from the GUI and forward it to the firmware. At the same time, in the case in which the timeout service states the absence of SPS transmission, it must also be able to retrieve the forced value from the circuit and overwrite it on the GUI, in order to inform the user of the actual source of instructions processed by the circuit at that time. The necessity of exploiting a single entity, instead of two parallel ones communicating in opposite directions, is that the whole process needs to act on a single copy of the `switch` signal and not on two redundant versions of the same wire.

As it can be seen from Listing 3, the overwriting behaviour is ruled by the `update` value, which states that the `dynamic_switch_register` has to read the `switch` value from the GUI and forward it to the firmware when `update`=0, while it must invert its communication direction when `update`=1. The latter is a 1-bit value generated by the FSM's `state_change` process and looped back to the IPBUS slaves. It consists in a simple copy of the `max_reached` value generated by the timeout's logic, which will be better explained in Section 4.6.3.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;
library work;
use work.ipbus.all;
entity dynamic_reg is
port(
    clk              : in  std_logic;
    reset            : in  std_logic;
    ipbus_in         : in  ipb_wbus;          --request
    ipbus_out        : out ipb_rbus;          --answer
    read             : out std_logic;
    write            : out std_logic;
    update           : in  std_logic;
    d                : in std_logic_vector(31 downto 0);
    q                : out std_logic_vector(31 downto 0)
    );
end dynamic_reg;
architecture Behavioral of dynamic_reg is
    signal reg : std_logic_vector(31 downto 0);
    signal ack : std_logic;
begin
    master_to_slave : process(clk) is
    begin
        if rising_edge(clk) then
            if update='0' then --if input comes from master
                if ipbus_in.ipb_strobe = '1' and ipbus_in.ipb_write = '1' then
                    reg <= ipbus_in.ipb_wdata;
                end if;
            elsif update='1' then
                    reg <= d;
            end if;
        end if;
    end process;
    ipbus_out.ipb_rdata <= reg; --read data from FSM or master according to update
    ipbus_out.ipb_timeout <= X"0000";
    ipbus_out.ipb_ack <= ipbus_in.ipb_strobe;
    ipbus_out.ipb_err <= '0';
    q <= reg;--always output what read
 end architecture;
```

Listing 3: `dynamic_switch_register`' logic.

4.7.3 Timeout and flags

The logic of the timeout service is entirely contained within the FSM's `idle` state and it follows these simple steps. First of all, a 1-bit `max_reached` value is defined internally to the FSM component, in order to signal the reaching of the maximum waiting time for the `WWE` instruction. Secondarily, different cases are evaluated according to the values of the `inner_switch`. Eventually, depending on each specific combination of `max_reached` and `inner_switch`, the 1-bit `definite_switch` signal is assigned with a different value.

The simplest scenario corresponds to `max_reached`=0 and `inner_switch`=0: in this case the input received by the GUI imposes to analyze the artificial signals. As a consequence, the timeout service is not required, since it is only meant to switch the source from Real to Artificial. For this combination, the `output_flag` (which is forwarded to the `output_valid` component to enable the outputting of instructions to the DAQ, see Fig.32) is raised to 1 and the `definite_switch` is assigned to 0, stating the necessity to analyze the spill information received by the Artificial generator. Instead, if the Real signals are required (meaning `inner_switch`=1), it becomes necessary to subdivide again the logic according to the values of the `force_switch` signal generated by the `variable_counter` component. In the case in which `inner_switch`=1 and `force_switch`=0, the `variable_counter` entity needs to start counting the waiting time for `WWE` instruction; until the above conditions are satisfied, the `definite_switch` signal is assigned to 1, as the `inner_switch` suggests.

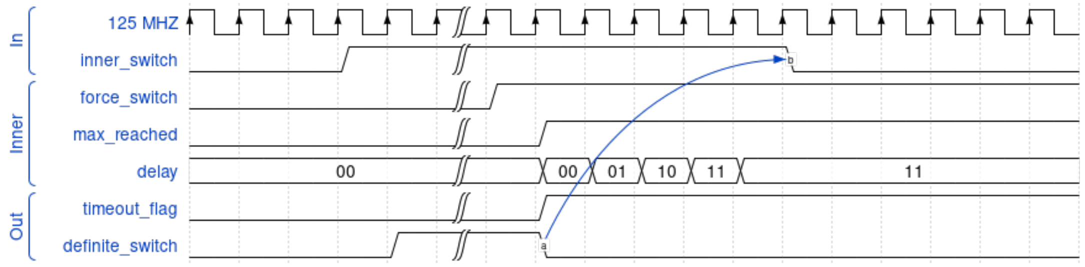


Figure 48: `switch`'s wave behaviour within Timeout service. The blue arrow represents the back propagation of the information through the `dynamic_switch_register`.

When the maximum value of clock periods is reached, meaning that SPS has failed into transmitting the `WWE` instruction on time, the `variable_counter` entity raises to 1 its `force_switch` value. This causes the logic to fall in the `max_reached`=0, `inner_switch`=1 and `force_switch`=1 combination: in this case, the `definite_switch` signal is set to 0, stating the necessity to move the analysis to the artificial signals. At the same time the `max_reached` signal and an `timeout_flag` are raised to 1, the latter being sent to the GUI to inform the user that the `switch` signal has been forced by the firmware and that temporarily no other input will be processed by the module.

When `max_reached`=1, the circuit knows that the `switch` value is no longer the one originally set from the GUI, but better its forced version generated by the logic itself. In this scenario, the `timeout_flag` is kept constant at 1 and the waiting time and cycle time counters are reset to 0. At this point a 4 clock periods dead time (corresponding to 32 ns) needs to be introduced, in order to guarantee the proper functioning of the `dynamic_switch_register`. The IPBUS protocol exchanges information between the GUI and the firmware with a lower frequency with respect to the one of the Artificial Spill Generator (respectively 25 MHz and 125 MHz), therefore the module needs to wait a consistent amount of time before the information is looped back to the dynamic register and overwritten (see Fig.48). After the 32 ns delay, when the updated version of the `inner_switch` finally reaches the logic (`inner_switch`=0) the forcing process has reached its end: the `max_reached` signal is brought back to 0, together with the `timeout_flag` at the next clock cycle.

At this point the module is brought back to its original state, after having switched the source from Real to Artificial. The users can again change the inputs from the GUI and the module would be ready to process them accordingly (see Fig.49 for the detailed architecture).

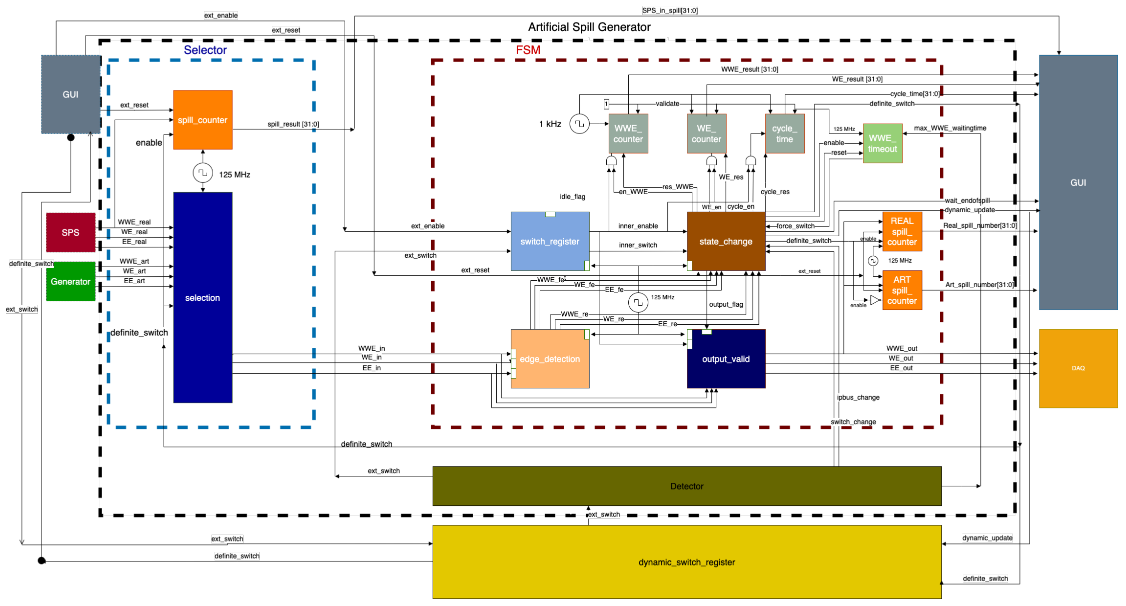
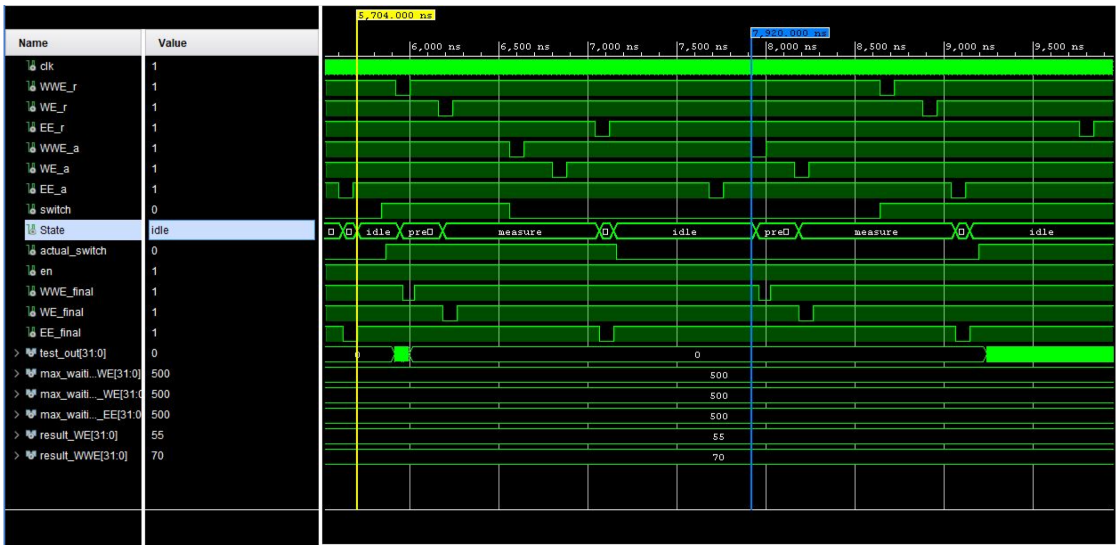


Figure 49: Detailed Manager's architecture with Timeout service and Measurements

4.8 Top Entity

For the implementation of the whole firmware, each entity described in the previous pages was written on separate files. The behaviour of each component was properly simulated with dedicated test benches, exploiting the Xilinx Vivado Suite [34]. In Fig.50 an example of the simulation interface is reported. For simplicity, the signals generated in the corresponding test bench do not follow the effective period of the SPS supercycle, but they act on the μs timescale, since the aim of the simulation was just to verify the effective behaviour of the logic with respect to the SPS instructions structure. The measurements of the time intervals `WWE-WE, WE-EE` and the counting of the waiting time for `WWE` instruction are also visible in the picture.

The smaller entities were inserted into more complicated files as VHDL `components`, thanks to the automatic building hierarchy of the Vivado program. Eventually, the Generator and the Manager were properly inserted into a `top.vhd` file provided by the collaboration, where all the necessary links of signals were performed according to the FPGA specifics and to the other modules already present in the firmware.



Name	Value
clk	1
WWE_r	1
WE_r	1
EE_r	1
WWE_a	1
WE_a	1
EE_a	1
switch	0
State	idle
actual_switch	0
en	1
WWE_final	1
WE_final	1
EE_final	1
test_out[31:0]	0
max_waiti...WE[31:0]	500
max_waiti..._WE[31:0	500
max_waiti..._EE[31:0	500
result_WE[31:0]	55
result_WWE[31:0]	70

Figure 50: Simulation of the firmware's behaviour for the case of superposition of Real and Artificial signals.

As it can be seen in Tab.15, the whole module presents several features and provides different useful services. First of all it can be switched on and off via `ext_enable` and `ext_reset` signals, automatically synchronizing said instructions to the spill phase in which the experiment is found. In addition, it allows the user to choose between two different sources of signal information (Artificial and Real) via the `ext_switch` signal, which is both read and overwritten by the module itself according to the instructions given by the Timeout service. Eventually, it provides the GUI with several useful measurements of fundamental spill parameters, like the time distances between `WWE-WE`,`WE-EE` (respectively `result_WWE`,`result_WE` in Tab.15), the `cycle_time` value and the number of incoming SPS signals (`SPS_in_spill_number`) and processed SPS and Artificial spills (respectively `SPS_out_spill_number`,`ARTR_out_spill_number`). Its contribution to the collaboration allows a safer and more controlled management of the `WWE`,`WE`,`EE` signals and of the relative DAQ usage (see Fig.19).

Signals	Bits	In/Out	Description
clock	1	Input	Periodic: 125 MHz.
ext_switch	1	Input	switch value set from GUI.
ext_enable	1	Input	From GUI: boolean for enabling.
ext_reset	1	Input	From GUI: boolean for resetting.
max_waitingtime_WWE	32	Input	Max value for Timeout.
WWE_real	1	Input	SPS WWE signal.
WE_real	1	Input	SPS WE signal.
EE_real	1	Input	SPS EE signal.
WWE_final	1	Output	WWE forwarded to DAQ.
WE_final	1	Output	WE forwarded to DAQ.
EE_real	1	Output	EE forwarded to DAQ.
result_WWE	32	Output	Distance between WWE-WE.
result_WE	32	Output	Distance between WE-EE.
cycle_time	32	Output	Distance between spills.
timeout_WWE	32	Output	Real-time WWE waitingtime.
dynamic_update	1	Output	Flag signalling forced switch.
actual_switch	1	Output	switch selected by FSM.
timeout_flag_out	1	Output	Flag: Timeout reached.
wait_endofspill	1	Output	Flag: no update of parameters.
SPS_in_spill_number	32	Output	Received SPS spills.
SPS_out_spill_number	32	Output	Processed SPS spills.
ART_out_spill_number	32	Output	Processed Artificial spills.

Table 15: Summary of In/Out signals of the top entity.

4.9 Simulation and Tests

The firmware was simulated, tested and synthetized with the tools available in Vivado's software, thanks to which the FPGA board was eventually programmed through a simple USB connection. Once the bitstream had been downloaded into the FPGA, the GUI and its interaction with the firmware were simulated by means of Pyhton scripts, exploiting the IPBUS Ethernet based protocol. The script reported in Listing 4 rules the communication from the server to the FPGA. In these simple lines each input parameter, which will eventually be set by the user in the control room, is transferred to the FPGA by means of the ipbus_write customized function. WE_DELAY,SPILL_LENGTH,SPILL_PAUSE respectively represent the time distances between WWE-WE,WE-EE,EE-WWE instructions interpreted by the Generator, while `MAX_TIME` represents the maximum waiting time of the timeout service and `ENABLE` and `SPILL_SWITCH` are simply different names for the usual `ext_enable`,`ext_switch` values.

```
 1 !/usr/bin/python
 2 import sys
 3 import uhal
 4 from argparse import ArgumentParser
 5 DEBUG=1
 6 uhal.setLogLevelTo(uhal.LogLevel.WARNING)#Open Connection
 7 manager=uhal.ConnectionManager("file://connection.xml")
 8 hw = manager.getDevice("arts_amber")
 9 def ipbus_write(register,value,DEBUG=0): #Write IPBUS Register
10     hw.getNode(register).write(value)
11     hw.dispatch()
12     if DEBUG:
13         var = hw.getNode(register).read()
14         hw.dispatch()
15         print("%s = 0x%8.8x (%i)"%(register,var,var))
16         hw.dispatch()
17         return var
18     else:
19         return 0
20 
21 print ("===== SETTING ART. SPILL GENERATOR PARAMETERS====")
22 ipbus_write("TF_LENGTH",624,DEBUG) #624 = 100kHz Art. Trigger
23 ipbus_write("SIGNAL_WIDTH",10,DEBUG) # in 16 ns units
24 ipbus_write("WE_DELAY",2,DEBUG) #in ms
25 ipbus_write("SPILL_LENGTH",1,DEBUG) #in ms
26 ipbus_write("SPILL_PAUSE",9,DEBUG) #in ms
27 ipbus_write("MAX_TIME",625000000,DEBUG) #in 16 ns units, 10 sec
28 ipbus_write("ENABLE",1,DEBUG)
29 if len(sys.argv)>1: #0=Internal Signals, 1=SPS Signal
30     ipbus_write("SPILL_SWITCH",int(sys.argv[1]),DEBUG)
31 else:
32     print("Default Internal Spill Generator")
33     ipbus_write("SPILL_SWITCH",0,DEBUG)
```

Listing 4: Python code for Writing on FPGA via IPBUS protocol.

The nodes are retrieved by the code according to the `connection.xml` file, as it can be seen from line 7 in Listing 4. Such file represents the mapping of read and written variables and their corresponding registers withing the firmware. Each variable has a different name, which corresponds to a different address in the logic. The registers in the circuit were created with specific integer addresses, reported in the connection file in their corresponding hexadecimal format.

A different code was used to read the information from the FPGA (see Listing 6). By running multiple times Listings 4 and 6, different configurations of the firmware were simulated and tested, in order to check its behaviour in an heterogeneous set of possible scenarios.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<node id="TOP">
    <node id="STATUR_REG"     address="0x00000087" permission="r">
    <node id "UCF_STATUS"     mask="0x1" permission="r" />
    <node id "LOCKED_MAIN"    mask="0x2" permission="r" />
    <node id "TCS_SYNC"       mask="0x4" permission="r" />
    <node id "IDELAYCTRL_RDY" mask="0x8" permission="r" />
</node>
    <node id="SIGNAL_WIDTH"      address="0x00000088" permission="rw" />
    <node id="TF_LENGTH"         address="0x0000008A" permission="rw" />
    <node id="WE_DELAY"          address="0x0000008C" permission="rw" />
    <node id="SPILL_LENGTH"      address="0x0000008E" permission="rw" />
    <node id="SPILL_PAUSE"       address="0x00000090" permission="rw" />
    <node id="SPILL_SWITCH"      address="0x00000096" permission="rw" />
    <node id="MAX_TIME"          address="0x00000098" permission="rw" />
    <node id="ENABLE"            address="0x0000009A" permission="rw" />
    <node id="WWE_TIME"          address="0x0000008B" permission="r" />
    <node id="WE_TIME"           address="0x00000097" permission="r" />
    <node id="ACTUAL_SWITCH"     address="0x00000099" permission="r" />
    <node id="TEST_OUT"          address="0x0000008F" permission="r" />
    <node id="DYNAMIC0"          address="0x000000C8" permission="rw" />
    <node id="DYNAMIC_UPDATE"    address="0x00000092" permission="rw" />
    <node id="OPEN_6"            address="0x00000094" permission="rw" />
    <node id="test2"             address="0x00000091" permission="rw" />
    <node id="fw"                address="0x00000093" permission="rw" />
    <node id="IP"                address="0x00000095" permission="rw" />
    <node id="MAC"               address="0x0000008d" permission="rw" />
```

Listing 5: File `connection.xml`: linking registers addresses to their variable names.

A second phase of tests was performed in the laboratory, before deploying the Artificial Spill Generator module in the experiment. The Real signals were simulated analogically by exploiting a dual timer, a multiplexer and a delay generator, accurately tuning the time distances between the impulses in order to test the majority of possible scenarios and the relative functioning of the firmware. The actual instruction generation was checked with a classic oscilloscope (see Fig.51), which allowed the verification of the structure of the spill instructions as well as the comparison between the measurements retrieved with IPBUS protocol and their actual values.

```python
!/usr/bin/python
import sys
import uhal
from argparse import ArgumentParser
DEBUG=1
#Open Connection
uhal.setLogLevelTo(uhal.LogLevel.WARNING)
manager = uhal.ConnectionManager("file://connection.xml")
hw = manager.getDevice("arts_amber")
#Read IPBUS Register
def ipbus_read(register):
var = hw.getNode(register).read()
hw.dispatch()
prnt ("%s = 0x%8.8x (%i)"%(register,var,var))
print("====ART. SPILL GENERATOR PARAMETERS====")
ipbus_read("TF_LENGTH")#624=100kHz Art.Trigger
ipbus_read("SIGNAL_WIDTH")#in 16ns units
ipbus_read("WE_DELAY")#in ms
ipbus_read("SPILL_LENGTH")#in ms
ipbus_read("SPILL_PAUSE")#in ms
ipbus_read("ACTUAL_SWITCH")
ipbus_read("ENABLE")
MAX_TIME=ipbus_read("MAX_TIME")
print("MAX_TIME: %.6f ms, %.6f sec"%(MAX_TIME*16.0/1000000.,MAX_TIME*16.0/1000000000.0))
WWE_time=ipbus_read("WWE_time")
print("WWE_time: %.6f ms, %.6f sec"%(WWE_time*16.0/1000000.,WWE_time*16.0/1000000000.0))
WE_time=ipbus_read("WE_time")
print("WE_time: %.6f ms, %.6f sec"%(WE_time*16.0/1000000.,WE_time*16.0/1000000000.0))
```

Listing 6: Python code for Reading from FPGA via IPBUS protocol.

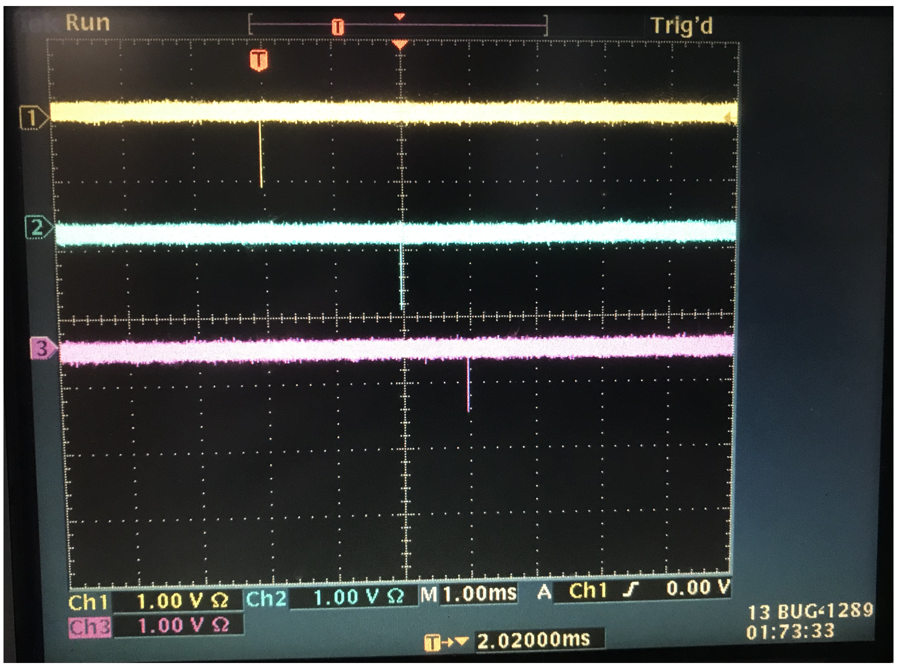


Figure 51: Effective WWE,WE,EE representation from the oscilloscope. The timescale used is not the nominal SPS one.

5 Conclusions

After having passed the whole set of laboratory tests, the Artificial Spill Generator has been deployed and programmed into FPGA board, which was eventually inserted into the DAQ pipeline of COMPASS, and now AMBER, collaborations. The Artificial Spill Generator firmware was exploited during the very end of COMPASS 2022 data taking and it is currently deployed also inside AMBER's DAQ system.

It contributed with a consistent gain in terms of DAQ up time and, as a consequence, with an increase in data taking efficiency and therefore physics data recorded by the experiment. Having a smooth transition between Dry Run and Run modes guarantees an optimal exploitation of the beam time, allowing a variable flux of data in the event building system without storing any on tape. Moreover, the possibility to visualize in real time from GUI the measured spill parameters, allows the collaboration to have fundamental insights on the status of the experiment, DAQ and beam. Eventually, the proper management of the two sources of synchronization signals allows an accurate and safe treatment of the incoming spills, preventing the DAQ from crashing because of the superposition of said structures.

Regarding the possible future improvements of the firmware, additional features could be inserted in order to ease the work of the shift crew in the control room. An automatic synchronization and tuning of spill parameters, with respect to SPS timing scheme and its particle delivery to the experiments of the M2 beam line, could avoid several human interventions and manual parameters setting of the artificial signals. Moreover, the number of spills skipped and not processed by the DAQ as a result of the safe stop implemented by the logic, could still be counted and visualized in real time in the GUI, providing useful statistical information for the amount of lost data.

References

[1] P. Abbon **andothers**. "The COMPASS setup for physics with hadron beams". **in**: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 779 (2015), **pages** 69–115. ISSN: 0168-9002. URL: `https://www.sciencedirect.com/science/article/pii/S0168900215000662`.

[2] B Adams **andothers**. "COMPASS++/AMBER: Proposal for Measurements at the M2 beam line of the CERN SPS Phase-1: 2022-2024". **in**: (2019). URL: `https://cds.cern.ch/record/2676885`.

[3] B. Adams **andothers**. *Letter of Intent: A New QCD facility at the M2 beam line of the CERN SPS (COMPASS++/AMBER)*. 2019. arXiv: `1808.00848 [hep-ex]`.

[4] C. Adolph **andothers**. "Final COMPASS results on the deuteron spin-dependent structure function and the Bjorken sum rule". **in**: *Physics Letters B* 769 (**june** 2017), **pages** 34–41. URL: `https://doi.org/10.1016%2Fj.physletb.2017.03.018`.

[5] M Bodlak **andothers**. "Monitoring tools of COMPASS experiment at CERN". **in**: *Journal of Physics: Conference Series* 664.8 (**december** 2015), **page** 082054. URL: `https://dx.doi.org/10.1088/1742-6596/664/8/082054`.

[6] A. BRESSAN. "Physics results from COMPASS". **in**: *Spin 2004*. WORLD SCIENTIFIC, **august** 2005. URL: `https://doi.org/10.1142%2F9789812701909_0006`.

[7] CERN. *Amber Main Page*. 2023. URL: `https://amber.web.cern.ch`.

[8] CERN. *IPBUS firmware component: ipbus_reg_v.vhd*. 2017. URL: `https://github.com/ipbus/ipbus-firmware/blob/master/components/ipbus_slaves/firmware/hdl/ipbus_reg_v.vhd`.

[9] CERN. *IPBUS firmware Github repository*. 2023. URL: `https://github.com/ipbus/ipbus-firmware`.

[10] CERN. *SPS Page 1*. URL: `https://op-webtools.web.cern.ch/vistar/vistars.php?usr=SPS1`.

[11] CERN. “The COMPASS Experiment at CERN”. **in**: 2007.

[12] The COMPASS Collaboration. *COMPASS-II Proposal: Questions and Answers*. techreport. Geneva: CERN, 2010. URL: `https://cds.cern.ch/record/1289162`.

[13] COMPASS. *Compass Main Page*. 2023. URL: `https://wwwcompass.cern.ch/compass/`.

[14] Louis Costrell **andothers**. “Standard NIM Instrumentation System”. **in**: (**may** 1990). URL: `https://www.osti.gov/biblio/7120327`.

[15] CERN Engineering Department. *Secondary Beam Areas Users Guide*. 2017. URL: `https://sba.web.cern.ch/sba/BeamsAndAreas/H2/H2_faq.html`.

[16] Ioanis Giomataris **andothers**. “Micromegas: a high-granularity position-sensitive gaseous detector for high particle-flux environments”. **in**: *Nucl. Instrum. Methods Phys. Res., A* 376.1 (1996), **pages** 29–35. URL: `https://cds.cern.ch/record/299159`.

[17] Muon Trigger Group. *Muon Trigger Documentation*. June 3, 2002. URL: `https://wwwcompass.cern.ch/compass/detector/trigger/muon-trigger/triggerdoc.pdf`.

[18] Ricardo Jasinski. *Effective Coding with VHDL: Principles and Best Practice*. The MIT Press, 2016. ISBN: 0262034220.

[19] I. Konorov **andothers**. “The trigger control system for the COMPASS”. **in**: *2001 IEEE Nuclear Science Symposium Conference Record (Cat. No.01CH37310)*. **volume** 1. 2001, 98–99 vol.1. DOI: `10.1109/NSSMIC.2001.1008418`.

[20] C. Ghabrous Larrea **andothers**. “IPbus: a flexible Ethernet-based control system for xTCA hardware”. **in**: *Journal of Instrumentation* 10.02 (**february** 2015), **page** C02019. URL: `https://dx.doi.org/10.1088/1748-0221/10/02/C02019`.

[21] CHEOPS: LoI. “Charm Experiment with Omni-Purpose Setup”. **in**: 1995.

[22] HMC: LoI. “Semi-inclusive muon scattering from a polarized target”. **in**: 1995.

[23] G.K. Mallot. “The COMPASS spectrometer at CERN”. **in**: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 518.1 (2004). Frontier Detectors for Frontier Physics: Proceedin, **pages** 121–124. ISSN: 0168-9002. URL: `https://www.sciencedirect.com/science/article/pii/S0168900203027499`.

[24] Ondej, Subrt **andothers**. “The Continuously Running iFDAQ of the COMPASS Experiment”. **in**: *EPJ Web Conf.* 214 (2019), **page** 01032. URL: `https://doi.org/10.1051/epjconf/201921401032`.

[25] Jorg Pretz. *Geometrical Trigger Acceptance*. February 20, 2002. URL: `https://wwwcompass.cern.ch/compass/detector/trigger/muon-trigger/triggeracc.ps`.

[26] Jorg Pretz. *The Veto System*. February 18, 2002. URL: `https://wwwcompass.cern.ch/compass/detector/trigger/muon-trigger/veto.ps`.

[27] COMPASS: Proposal. “Common Muon and Proton Apparatus for Structure and Spectroscopy”. **in**: 1996.

[28] Fabio Sauli. “The gas electron multiplier (GEM): Operating principles and applications”. **in**: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 805 (2016). Special Issue in memory of Glenn F. Knoll, **pages** 2–24. ISSN: 0168-9002. URL: `https://www.sciencedirect.com/science/article/pii/S0168900215008980`.

[29] Thomas Schmidt. “A Common Readout Driver for the COMPASS Experiment”. 2002. URL: `http://cds.cern.ch/record/1295180`.

[30] Lars Schmitt **andothers**. “The DAQ of the COMPASS experiment”. **in**: *Nuclear Science, IEEE Transactions on* 51 (**july** 2004), **pages** 439–444. DOI: `10.1109/TNS.2004.829386`.

[31] “The COMPASS trigger system for muon scattering”. **in**: **volume** 550. 1. 2005, **pages** 217–240.

[32] Malte Christian Wilfert. “Investigation of the spin structure of the nucleon at the COMPASS experiment”. **in**: *Dissertation zur Erlangung des Grades Doktor der Naturwissenschaften am Fachbe- reich 08 - Physik, Mathematik und Informatik der Johannes-Gutenberg-Universität Mainz D77* (March 2017).

[33] Thomas Stephen Williams. *IPbus A flexible Ethernet-based control system for xTCA hardware*. techreport. Geneva: CERN, 2014. URL: https://cds.cern.ch/record/2020872.

[34] Xilinx. *Vivado Suite overview page*. 2023. URL: https://www.xilinx.com/products/design-tools/vivado.html.

[35] Xilinx. *Xilinx FPGA*. 2023. URL: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

AMBER	Apparatus for Meson and Baryon Experimental Research
CERN	European Council for Nuclear Research
CHEOPS	CHarm Experiment with Omni-Purpose Setup
COMPASS	Common Muon and Proton Apparatus for Structure and Spectroscopy
DAQ	Data Acquisition System
DCS	Detector Control System
DVCS	Deep Virtual Compton Scattering
DVMP	Deep Virtual Meson Production
DY	Drell-Yan
ECAL	Electromagnetic Calorimeter
FPGA	Field Programmable Gate Arrays
FSM	Finite State Machine
GEM	Gas Electron Multiplier
GPDs	Generalized Parton Distributions
GUI	Graphical User Interface
HCAL	Hadronic Calorimeter
HEMP	Hard Exclusive Meson Production
HMC	Hadron Muon Collaboration
LAS	Large Angle Spectrometer

LAT	Large Area Trackers
LHC	Large Hadron Collider
LoI	Letter of Intent
MF	Muon Filter
Micromegas	Micromesh Gaseous Structures
MW	Muon Wall
MWPCs	Multi Wire Proportional Chambers
NIM	Nuclear Instrumentation Module
PDFs	Parton Distribution Functions
PGF	Photon Gluon Fusion
PS	Proton Synchrotrone
QCD	Quantum Chromodynamics
RICH	Ring Imaging Cherenkov Detector
RMWbits	Read-Modify-Write bits
SAS	Small Angle Spectrometer
SAT	Small Area Trackers
SciFi	Scintillating Fibers
SIDIS	Semi-Inclusive Deep Inelastic Scattering
SPS	Super Proton Synchrotrone
TCS	Trigger Control System
TTL	Transistor-Transistor Logic
TMD	Transverse Momentum Dependent

VHDL Very high-speed integrated circuit Hardware Description Language

VSAT Very Small Area Trackers

Acknowledgments

Usually I am not really into this kind of things. I've always thought that public thanks were a useless bourgeois formality, together with many other things. After this last year though, I understood that nothing should ever be taken for granted, therefore in this case being grateful is mandatory. First of all, my gratitude goes to those who, in the practice, helped me conclude this path. Especially to Andrea Triossi, who followed me step by step in the production of the thesis, with patience, trust and good disposition, which are human qualities too often underestimated. Most of this I owe it to Benjamin Moritz Veit, very careful supervisor, tireless worker and kind man who, with extreme patience, taught me more than anybody else has ever done. A special thank you of course goes to all of those friends and family members who, in a way or another, contributed to my growth during these years, constantly dwelling in my life and preventing me from going crazy: Alessio, Anna Rita, Arianna, Chiara, Costanza, Eugenia, Ginevra, Giulia, Matteo, Matilde, Paolo, Simone e Sabrina. This thesis is the result of a definitely too long year, to the protagonists of which I want to dedicate this work, together with all of my affection and gratitude.

To my parents. For the emotional support, for the comprehension and for the faith.
To Rachele. For having always been an inspiration, a role model of genius and madness, a friend, a sister and a cousin.
To the Cian Cianin-ers, Margherita and Valentina. For having been unaware parents, for the food, for the whine and for the love.
To my grand parents. For having been careful watchers, discreet witnesses and Home guardians.
To Anna and the Avogadro. For the friendship and the music.
To my Summer Student friends, Doga, Aksel e Maike. For having been and stayed, at the same place and at the same time, together.

A final greeting goes to those who are now way far away from here. For the emptiness.

Ringraziamenti

Di solito non mi presto volentieri a questo genere di cose. Ho sempre pensato che i ringraziamenti fossero, insieme a tante altre cose, un'inutile formalità borghese. Dopo quest'ultimo anno però ho capito che è bene non dare mai nulla per scontato e dunque in questo caso, essere grati è doveroso. Per prima cosa, la mia riconoscenza va a chi mi ha concretamente aiutato a concludere questo percorso, primo fra tutti Andrea Triossi che con pazienza, fiducia e disponibilità, qualità umane troppo spesso sottovalutate, mi ha seguito passo passo nella produzione della tesi. Molto di tutto questo lo devo inoltre a Benjamin Moritz Veit, attento supervisore, lavoratore instancabile e uomo gentile, che con estrema cura mi ha insegnato più di quanto abbia mai fatto chiunque altro. Un ringraziamento speciale va ovviamente anche a tutti quegli amici e parenti che, in un modo o nell'altro, hanno contribuito alla mia crescita in questi anni, abitando costantemente la mia vita ed impedendomi di impazzire: Alessio, Anna Rita, Arianna, Chiara, Costanza, Eugenia, Ginevra, Giulia, Matteo, Matilde, Paolo, Simone e Sabrina. Questa tesi è il risultato di un anno decisamente troppo lungo, ai cui protagonisti dedico questo lavoro, insieme a tutto il mio affetto e la mia gratitudine:

Ai miei genitori. Per il supporto emotivo, per la comprensione e per la fede.
A Rachele. Per essere sempre stata d'ispirazione, modello di genio e follia, amica, sorella e cugina.
Alle Cian Cianin-ers, Margherita e Valentina. Per essere state genitrici involontarie, per il cibo, per il vino e per l'amore.
Ai miei nonni. Per essere stati osservatori attenti e testimoni discreti, custodi di Casa.
Ad Anna e agli Avogadro. Per l'amicizia e per la musica.
Ai miei amici Summer Student, Doga, Aksel e Maike. Per esserci trovati, nello stesso luogo nello stesso momento, insieme.

Un saluto infine va anche a chi è ormai molto lontano da qui. Per il vuoto.